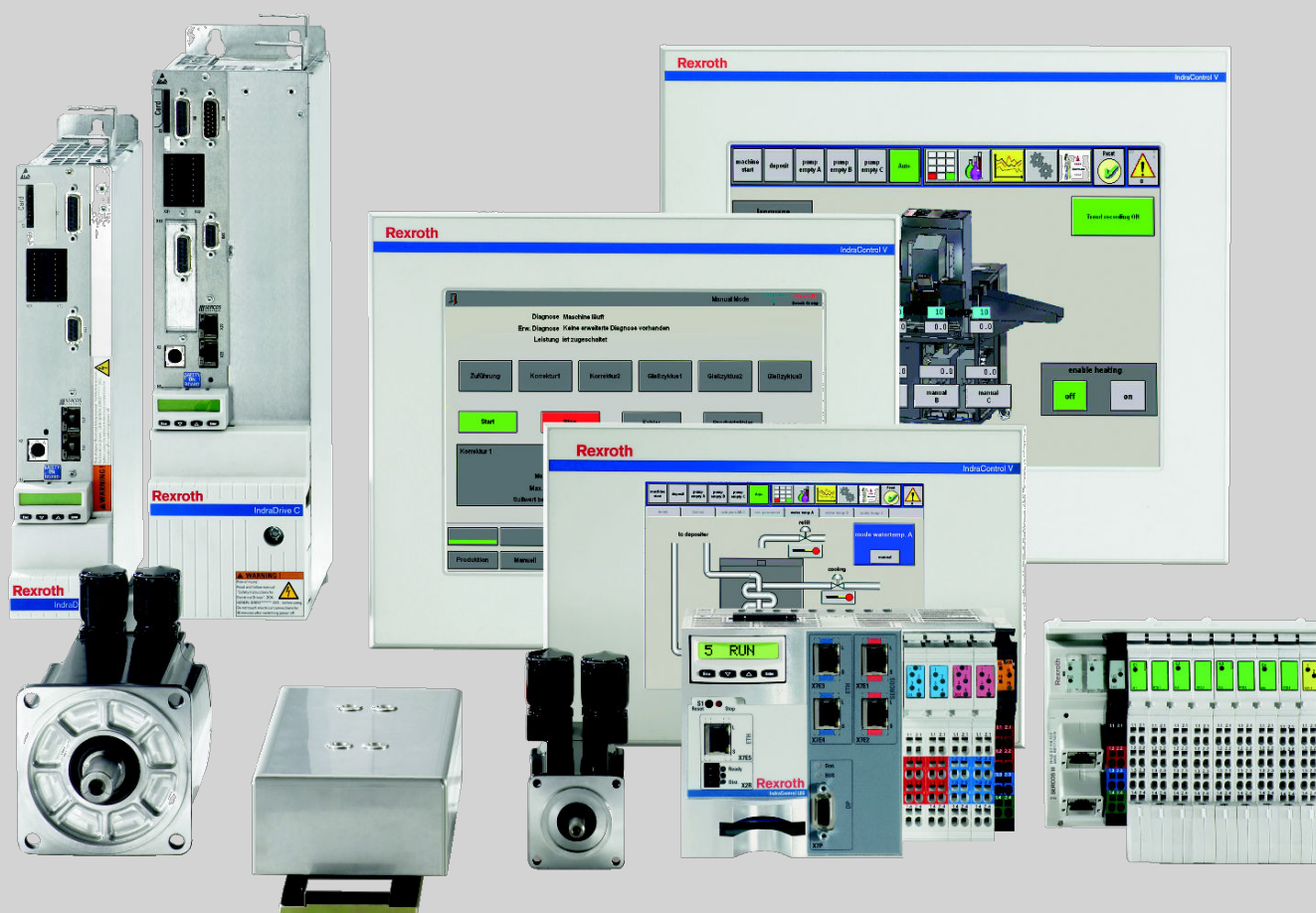


IndraWorks 15VRS

PLC Programming System IndraLogic 2G

Application Description
R911396137

Edition 01



Title	IndraWorks 15VRS PLC Programming System IndraLogic 2G
Type of Documentation	Application Description
Document Typecode	DOK-IWORKS-IL2GPRO*V15-AP01-EN-P
Internal File Reference	RS-23254ab1d57f74b60a347e862c73003a-1-en-US-3
Change Record	Edition 01, 2020-11 Refer to tab. 1-1 "Change Record" on page 1
Copyright	© Bosch Rexroth AG 2020 All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Liability	The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.
Editorial Department	Engineering Automation Systems PLC Engineering, GeKü (MiNi/MePe)

Table of Contents

	Page
1 About this documentation.....	1
1.1 Validity of the documentation.....	1
1.2 Using safety instructions.....	2
1.2.1 Structure of the safety instructions.....	2
1.2.2 Explaining signal words and safety alert symbol.....	2
1.2.3 Symbols used.....	4
1.2.4 Explaining the signal alert symbol on the device.....	4
1.3 Required and supplementing documentation.....	4
1.4 Names and abbreviations.....	5
2 Creating and configuring projects.....	7
2.1 Creating “empty project”.....	8
2.2 Adding objects.....	8
2.3 Modifying PLC compiler version.....	9
2.4 Opening IndraLogic 2G project.....	10
2.5 Configuring projects.....	11
2.5.1 Retrieving and editing library information.....	11
2.5.2 Applying project settings.....	13
3 Exporting and transferring projects.....	15
3.1 Transferring projects.....	15
4 Comparing PLC objects.....	17
4.1 Comparison: General module folder.....	18
4.2 Comparison: Logic.....	19
4.3 Comparison: Application.....	20
4.4 Comparison: Task configuration.....	23
4.5 Comparison: Task.....	24
4.6 Comparison: Folder (Differences with Respect to the IndraWorks Folder).....	25
4.7 Comparison: Data Types	25
4.8 Comparison: Global Variable List, Persistent Variables, Network Variables.....	27
4.9 Comparison: Network Variable Lists, Receiver End.....	28
4.10 Comparison: Programs, Function Blocks, Functions.....	29
4.11 Comparison: Interfaces, interface methods, interface properties.....	36
4.12 Comparison: Library manager.....	37
4.13 Comparison: Library Info.....	38
4.14 Comparison: Data Server.....	39
4.15 Comparison: Data source (DataServer).....	41
4.16 Comparison: Symbol configuration.....	42
4.17 Comparison: Visualization manager.....	42
4.18 Comparison: Visualization.....	42
4.19 Comparison: Text List.....	43
4.20 Comparison: Image Pool.....	44

	Page
4.21 Comparison: Recipe Manager.....	44
4.22 Comparison: Recipe Definition.....	45
4.23 Comparison: Devices (onboard I/O, Inline I/O, field buses, ...).....	46
5 Comparing PLC objects online.....	51
5.1 Overview on the comparison.....	51
5.2 Detailed comparison.....	52
5.3 Options.....	54
5.4 Error entries in the message window.....	54
6 Protecting and saving projects.....	57
6.1 Setting up write protection.....	59
6.2 Setting up user management.....	60
6.3 Protecting objects in the project via access rights.....	61
6.4 Logging in via user account and password manager.....	62
6.5 Saving the project.....	64
7 Multilingual PLC comments.....	65
8 Configuring an I/O link.....	69
8.1 Project Explorer and device editor.....	69
8.2 Devices in the Project Explorer	70
8.3 Configuring devices and I/O mapping.....	73
9 Programming applications.....	83
9.1 Specifying identifiers.....	83
9.2 Declaring variables	83
9.2.1 Using the declaration editor.....	85
9.2.2 Using the dialog "Declare variable".....	86
9.2.3 Declaring arrays.....	86
9.2.4 Declaring global variables.....	88
9.3 Generating source code in IEC.....	89
9.3.1 FBD/LD/IL.....	89
Programming in Function Block Diagram (FBD).....	92
Programming in Ladder Diagram (LD).....	93
Programming in Instruction List (IL).....	95
9.3.2 Continuous Function Chart (CFC).....	96
Programming in the CFC editor.....	97
9.3.3 Structured Text (ST) and Extended Structured Text (ExST).....	101
Programming in Structured Text (ST).....	101
9.3.4 Sequential Function Chart (SFC).....	102
Programming in Sequential Function Chart (SFC).....	102
9.4 Calling function block, function or method with external implementation.....	107
9.5 Using input assistants.....	107

	Page
9.6 Using pragmas.....	110
9.7 Using library function blocks.....	112
9.8 Managing text in a text list.....	113
9.8.1 Managing static text in global text lists.....	117
9.8.2 Managing dynamic text in a text list.....	120
9.9 Using image pools.....	121
9.10 Programmed access to I/Os.....	122
9.10.1 Variable configuration - VAR_CONFIG.....	123
9.10.2 AT declaration.....	125
9.11 Checking syntax and analyzing code.....	126
9.11.1 Checking syntax.....	126
9.11.2 Analyzing code statically.....	127
9.12 Orienting and navigating.....	129
9.12.1 Using the cross-reference list to find places of use.....	129
9.12.2 Finding declaration - "Go to definition".....	130
9.12.3 Setting and using bookmarks.....	131
9.13 Project-comprehensive "Find and replace".....	132
9.14 Refactoring.....	132
9.15 Task configuration.....	136
9.15.1 Creating task configurations.....	137
9.16 Data persistence.....	137
9.16.1 Retaining data using persistent variables.....	140
9.16.2 Preserving data with retain variables.....	142
9.16.3 Preserving data with recipes.....	142
9.16.4 Declaring VAR PERSISTENT variables.....	143
9.16.5 Saving the values of a persistent variable list in a recipe.....	144
9.17 Using function block for implicit check.....	145
9.18 Object-oriented programming.....	145
9.18.1 Extension of a function block.....	145
9.18.2 Implementing interfaces (IMPLEMENTS).....	147
9.18.3 Extending an interface.....	149
9.18.4 Method call.....	150
10 Working with control networks.....	153
10.1 Network and addressing.....	153
10.1.1 Network topology.....	153
10.1.2 Communication in the Control Link via Gateway.....	154
10.2 Symbol configuration.....	156
10.3 Network variables.....	158
10.3.1 Configuring a network variable exchange.....	160
10.4 Data Source Manager.....	162
10.4.1 Basics of the data source manager.....	162
10.4.2 Adding initial data source.....	164
10.4.3 Editing data source variables.....	167
10.4.4 Editing communication.....	169
10.4.5 Updating data interface.....	170

Table of Contents

	Page
10.4.6 Using remote data.....	171
10.5 Subordinate safety control.....	172
11 Transferring applications to the PLC.....	175
11.1 Configuring the connection to the PLC.....	175
11.2 Generating application code.....	175
11.3 Loading application code — Logging in and starting the PLC.....	176
11.4 Generating boot applications.....	177
12 Testing and debugging.....	179
12.1 Testing in simulation mode.....	179
12.2 Using breakpoints.....	179
12.3 Stepping a program.....	181
12.4 Forcing and writing variables.....	183
12.5 Resetting applications.....	185
12.6 Sequence control.....	186
12.7 Determining the current processing position with the call stack.....	189
12.8 Checking task list.....	189
13 Application at runtime.....	191
13.1 Monitoring values.....	191
13.1.1 Calling monitoring in programming object.....	191
13.1.2 Using watch lists.....	196
13.2 Changing values with recipes.....	198
13.3 Data recording with “Trace”.....	200
13.3.1 Getting started with trace.....	203
13.3.2 Creating the trace configuration.....	204
13.3.3 Controlling data recording.....	207
13.3.4 Accessing all traces of the control.....	207
13.3.5 Navigating in the trace data.....	207
13.3.6 Managing trace.....	207
13.3.7 Showing statistics.....	208
13.4 Monitoring tasks.....	208
13.5 Reading PLC logbook.....	208
13.6 Error analysis using core dump.....	208
13.7 PLC shell to query information.....	210
13.8 PLC operation control via system variables.....	211
14 Updating an application on the PLC.....	213
14.1 Executing an online change.....	213
14.2 Executing a download.....	214
15 Copying files to/from PLCs.....	217

	Page
16 Creating and managing libraries.....	219
16.1 Information for library developers.....	219
16.2 Adding a library to the application.....	220
16.3 Adding a library to the repository.....	221
16.4 Exporting library file.....	221
17 Reference “Programming”.....	223
17.1 Programming languages and editors.....	223
17.1.1 Declaration Editor.....	223
17.1.2 Common functionalities in graphical editors.....	224
17.1.3 Structured Text and Extended Structured Text (ExST).....	225
ST editor.....	225
ST editor in online mode.....	226
ST expressions.....	226
Assignments	227
ST assignment operator.....	227
ST assignment operator for outputs.....	227
ExST assignment 'S='.....	228
ExST assignment 'R='.....	228
ExST assignment “:=” as expression.....	229
Assignment operator 'REF'.....	230
Statements.....	230
ST statement “IF”.....	230
ST statement “FOR”	231
ST statement “CASE”.....	232
ST statement “WHILE”.....	232
ST statement “REPEAT”	233
ST statement “RETURN”.....	234
ST statement “JMP”.....	234
ST statement “EXIT”.....	235
EXST statement “CONTINUE”.....	235
ST - Calling function blocks.....	235
ST - Comments.....	235
17.1.4 Sequential function chart (SFC).....	236
SFC (sequential function chart) editor.....	236
SFC editor in online mode.....	236
Processing sequence in SFC.....	237
Qualifiers for SFC actions.....	239
Implicit variables.....	240
SFC flags.....	241
Elements.....	245
SFC elements “steps” and “transition”	245
SFC element “Action”	247
SFC element “Branching”.....	249
SFC element “Jump”.....	251
SFC element “Macro”	251

Table of Contents

	Page
SFC element properties.....	252
17.1.5 Function block plan/ladder diagram/statement list (FBD/LD/IL).....	254
FBD/LD/IL Editor.....	254
FBD/LD/IL editor in online mode.....	258
Modifiers and operators in IL.....	259
Elements.....	263
FBD/LD/IL element "Network".....	263
FBD/LD/IL element "Function block".....	264
FBD/LD/IL element "Assignment".....	264
FBD/LD/IL element "Function block with EN/ENO".....	264
FBD/LD/IL element "Input".....	264
FBD/LD/IL element "Jump label".....	265
FBD/LD/IL element "Jump".....	265
FBD/LD/IL element "Return".....	265
FBD/LD/IL element "Line branching".....	265
FBD/LD/IL element "Execute".....	266
LD element "Contact".....	266
LD element "Coil".....	267
LD element "Line branching start/end".....	268
Closed line branching.....	268
17.1.6 Continuous Function Chart (CFC) and page-oriented CFC.....	269
CFC editor.....	270
Page-oriented CFC editor.....	272
CFC editor in online mode.....	273
Elements.....	277
CFC element "Page".....	277
CFC element "Control point".....	277
CFC element "Input".....	277
CFC element "Output".....	278
CFC element "Function block".....	278
CFC element "Jump".....	278
CFC element "Label".....	278
CFC element "Return".....	278
CFC element "Composer".....	279
CFC element "Selector".....	279
CFC element "Comment".....	279
CFC element "Connection mark source/target".....	279
CFC element "Function block input".....	280
CFC element "Function block output".....	280
17.2 Variables.....	280
17.2.1 Local Variables - VAR.....	280
17.2.2 Input variables - VAR_INPUT.....	280
17.2.3 Output variables - VAR_OUTPUT.....	281
17.2.4 Input and output variable - VAR_IN_OUT.....	281
17.2.5 Global variables - VAR_GLOBAL.....	285
17.2.6 Temporary variable - VAR_TEMP.....	286

	Page
17.2.7 Static variables - VAR_STAT.....	286
17.2.8 External variables - VAR_EXTERNAL.....	286
17.2.9 Instance variables - VAR_INST.....	287
17.2.10 Constant variables - VAR CONSTANT.....	287
17.2.11 Persistent variables - PERSISTENT.....	288
17.2.12 Retain Variable - RETAIN.....	290
17.2.13 Configuration variable - VAR_CONFIG.....	291
17.2.14 SUPER.....	291
17.2.15 THIS.....	293
17.3 Operators.....	294
17.3.1 Operator "ADD".....	298
17.3.2 Operator "MUL".....	298
17.3.3 Operator "SUB".....	299
17.3.4 Operator "DIV".....	299
17.3.5 Operator "MOD".....	300
17.3.6 Operator "MOVE".....	301
17.3.7 Operator "INDEXOF".....	301
17.3.8 Operator "SIZEOF".....	301
17.3.9 Operator "NOT".....	302
17.3.10 Operator "AND".....	302
17.3.11 Operator "OR".....	302
17.3.12 Operator "XOR".....	303
17.3.13 Operator "AND_THEN".....	303
17.3.14 Operator "OR_ELSE".....	304
17.3.15 Operator "SHL".....	304
17.3.16 Operator "SHR".....	305
17.3.17 Operator "ROL".....	305
17.3.18 Operator "ROR".....	306
17.3.19 Operator 'SEL'.....	307
17.3.20 Operator "MAX".....	308
17.3.21 Operator "MIN".....	308
17.3.22 Operator "LIMIT".....	308
17.3.23 Operator "MUX".....	309
17.3.24 Operator "GT".....	309
17.3.25 Operator "LT".....	310
17.3.26 Operator "LE".....	310
17.3.27 Operator "GE".....	310
17.3.28 Operator "EQ".....	311
17.3.29 Operator "NE".....	311
17.3.30 Operator "ADR".....	311
17.3.31 Operator "Content operator".....	312
17.3.32 Operator "BITADR".....	313
17.3.33 Operator "CAL".....	313
17.3.34 Operator 'BOOL_TO'.....	314
17.3.35 Operator 'TO_BOOL'.....	315
17.3.36 Operator "TO_<xxx>".....	315

Table of Contents

	Page
17.3.37 Operator "<INT type>_TO_<INT type>"	316
17.3.38 Operator 'REAL_TO-/ LREAL-TO'	316
17.3.39 Operator 'TIME_TO/TIME_OF_DAY_TO'	317
17.3.40 Operator 'DATE_TO/DT_TO'	318
17.3.41 Operator 'STRING_TO'	319
17.3.42 Operator 'TRUNC'	320
17.3.43 Operator 'TRUNC_INT'	321
17.3.44 Operator "ABS"	321
17.3.45 Operator "SQRT"	321
17.3.46 Operator "LN"	322
17.3.47 Operator "LOG"	322
17.3.48 Operator "EXP"	322
17.3.49 Operator "EXPT"	323
17.3.50 Operator "SIN"	323
17.3.51 Operator "COS"	324
17.3.52 Operator "TAN"	324
17.3.53 Operator "ASIN"	324
17.3.54 Operator "ACOS"	325
17.3.55 Operator "ATAN"	325
17.3.56 Operator '_DELETE'	326
17.3.57 Operator "__ISVALIDREF"	327
17.3.58 Operator '_NEW'	327
17.3.59 Operator "__QUERYINTERFACE"	329
17.3.60 Operator "__QUERYPOINTER"	330
17.3.61 Operators "__TRY", "__CATCH", "__FINALLY", "__ENDTRY"	331
17.3.62 Operator '_VARINFO'	332
17.3.63 Operator '_CURRENTTASK'	334
17.3.64 Operator '_COMPARE_AND_SWAP'	335
17.3.65 Operator '_XADD'	336
17.3.66 Operator 'TEST_AND_SET'	336
17.3.67 Operator - Global namespace	337
17.3.68 Operator - Namespace for Global Variable Lists	337
17.3.69 Operator - Library namespace	338
17.3.70 Operator - Enumeration namespace	338
17.3.71 Operator 'INI'	338
17.4 Operands	339
17.4.1 BOOL constants	339
17.4.2 TIME constants	339
17.4.3 DATE constant	341
17.4.4 DATE_AND_TIME constant	341
17.4.5 TIME_OF_DAY constant	342
17.4.6 Number constants	342
17.4.7 REAL/LREAL constant	343
17.4.8 String constant	344
17.4.9 WSTRING constants	345
17.4.10 Typed literals	346

	Page
17.4.11 Access to variables in arrays, structures and function blocks.....	346
17.4.12 Bit access to variables	347
17.4.13 Addresses.....	348
17.4.14 Functions.....	350
17.5 Data types.....	351
17.5.1 "BOOL" data type.....	352
17.5.2 Integer data types.....	352
17.5.3 Data type 'REAL' / 'LREAL'.....	352
17.5.4 "STRING" data type.....	353
17.5.5 Time data types.....	353
17.5.6 "UNION" data type.....	354
17.5.7 Data type 'ANY' und 'ANY <type>'.....	355
17.5.8 "LTIME" data type.....	358
17.5.9 "WSTRING" data type.....	358
17.5.10 Data type 'BIT'.....	359
17.5.11 Special data types "__UXINT" and "__XWORD".....	359
17.5.12 Data type 'POINTER'.....	359
17.5.13 Data type 'REFERENCE'.....	361
17.5.14 Data type 'ARRAY'.....	363
17.5.15 Data type ' __VECTOR'.....	367
17.5.16 Data type 'STRUCT'.....	373
17.5.17 Enumerations.....	374
17.5.18 Subrange types.....	377
17.6 Pragmas.....	378
17.6.1 Message pragmas.....	378
17.6.2 Attribute pragmas.....	379
User-defined attributes.....	379
Attribute "call_after_global_init_slot".....	380
Attribute 'call_after_init'.....	380
Attribute 'call_after_online_change_slot'.....	381
Attribute 'call_before_global_exit_slot'.....	382
Attribute "call_on_type_change".....	382
Attribute "const_replaced", attribute "const_non_replaced".....	383
Attribute "dataflow".....	383
Attribute "displaymode".....	384
Attribute 'estimated-stack-usage'.....	385
Attribute "ExpandFully".....	388
Attribute "global_init_slot".....	388
Attribute 'hide'.....	389
Attribute "hide_all_locals".....	390
Attribute "initialize_on_call".....	391
Attribute "init_namespace".....	391
Attribute "init_on_onlchange".....	392
Attribute "instance-path".....	392
Attribute 'io_function_block', 'io_function_block_mapping'.....	393
Attribute "linkalways".....	394

Table of Contents

	Page
Attribute 'monitoring'.....	394
Attribute "namespace".....	397
Attribute 'no_assign', attribute 'no_assign_warning'.....	398
Attribute "no_check".....	399
Attribute "no_copy".....	399
Attribute 'no-exit'.....	400
Attribute "noinit".....	400
Attribute "no_instance_in_retain".....	400
Attribute "no_virtual_actions".....	401
Attribute 'pingroup'.....	403
Attribute 'pin_presentation_order_inputs/outputs'.....	404
Attribute "obsolete".....	405
Attribute "pack_mode".....	405
Attribute "qualified_only".....	408
Attribute "reflection".....	409
Attribute 'to_string'.....	409
Attribute "strict".....	410
Attribute "subsequent".....	410
Attribute "symbol".....	410
Attribute "warning disable", attribute "warning restore".....	412
Attribute 'enable_dynamic_creation'.....	412
Attribute 'ProcessValue'.....	412
17.6.3 Conditional pragmas.....	413
17.6.4 Region pragma.....	418
17.7 Identifier.....	419
17.8 Shadowing rules.....	424
17.9 Keywords	427
17.10 'FB_Init', 'FB_Reinit' and 'FB_Exit' methods.....	427
17.11 Compiler error messages.....	431
17.11.1 Compiler errors C0001.....	432
17.11.2 Compiler errors C0002.....	432
17.11.3 Compiler errors C0003.....	433
17.11.4 Compiler errors C0004.....	433
17.11.5 Compiler errors C0005.....	433
17.11.6 Compiler errors C0006.....	434
17.11.7 Compiler errors C0007.....	434
17.11.8 Compiler errors C0008.....	434
17.11.9 Compiler errors C0009.....	435
17.11.10 Compiler errors C0010.....	435
17.11.11 Compiler errors C0011.....	435
17.11.12 Compiler errors C0013.....	435
17.11.13 Compiler errors C0015.....	436
17.11.14 Compiler errors C0018.....	436
17.11.15 Compiler errors C0020.....	437
17.11.16 Compiler errors C0022.....	437
17.11.17 Compiler errors C0023.....	437

	Page
17.11.18 Compiler errors C0026.....	438
17.11.19 Compiler errors C0027.....	438
17.11.20 Compiler errors C0030.....	438
17.11.21 Compiler errors C0031.....	439
17.11.22 Compiler errors C0032.....	439
17.11.23 Compiler errors C0035.....	439
17.11.24 Compiler errors C0037.....	440
17.11.25 Compiler errors C0038.....	440
17.11.26 Compiler errors C0039.....	441
17.11.27 Compiler errors C0040.....	441
17.11.28 Compiler errors C0041.....	442
17.11.29 Compiler errors C0043.....	442
17.11.30 Compiler errors C0044.....	442
17.11.31 Compiler errors C0045.....	443
17.11.32 Compiler errors C0046.....	443
17.11.33 Compiler errors C0047.....	444
17.11.34 Compiler errors C0048.....	444
17.11.35 Compiler errors C0049.....	444
17.11.36 Compiler errors C0050.....	445
17.11.37 Compiler errors C0061.....	445
17.11.38 Compiler errors C0062.....	445
17.11.39 Compiler errors C0064.....	446
17.11.40 Compiler errors C0066.....	446
17.11.41 Compiler errors C0068.....	446
17.11.42 Compiler errors C0069.....	447
17.11.43 Compiler errors C0070.....	447
17.11.44 Compiler errors C0072.....	448
17.11.45 Compiler errors C0074.....	448
17.11.46 Compiler errors C0075.....	448
17.11.47 Compiler errors C0076.....	449
17.11.48 Compiler errors C0077.....	449
17.11.49 Compiler errors C0080.....	449
17.11.50 Compiler errors C0081.....	450
17.11.51 Compiler errors C0082.....	450
17.11.52 Compiler errors C0084.....	450
17.11.53 Compiler errors C0086.....	451
17.11.54 Compiler errors C0087.....	451
17.11.55 Compiler errors C0089.....	452
17.11.56 Compiler errors C0090.....	452
17.11.57 Compiler errors C0091.....	452
17.11.58 Compiler errors C0094.....	453
17.11.59 Compiler errors C0096.....	453
17.11.60 Compiler errors C0097.....	454
17.11.61 Compiler errors C0098.....	454
17.11.62 Compiler errors C0101.....	454
17.11.63 Compiler errors C0102.....	455

Table of Contents

	Page
17.11.64 Compiler errors C0104.....	455
17.11.65 Compiler errors C0114.....	455
17.11.66 Compiler errors C0116.....	455
17.11.67 Compiler errors C0117.....	456
17.11.68 Compiler errors C0118.....	456
17.11.69 Compiler errors C0119.....	456
17.11.70 Compiler errors C0120.....	457
17.11.71 Compiler errors C0122.....	457
17.11.72 Compiler errors C0122.....	457
17.11.73 Compiler errors C0125.....	458
17.11.74 Compiler errors C0126.....	458
17.11.75 Compiler errors C0130.....	458
17.11.76 Compiler errors C0131.....	459
17.11.77 Compiler errors C0132.....	459
17.11.78 Compiler errors C0136.....	459
17.11.79 Compiler errors C0139.....	460
17.11.80 Compiler errors C0140.....	460
17.11.81 Compiler errors C0141.....	460
17.11.82 Compiler errors C0142.....	461
17.11.83 Compiler errors C0143.....	461
17.11.84 Compiler errors C0144.....	461
17.11.85 Compiler errors C0145.....	462
17.11.86 Compiler errors C0149.....	462
17.11.87 Compiler errors C0161.....	462
17.11.88 Compiler errors C0162.....	463
17.11.89 Compiler errors C0168.....	463
17.11.90 Compiler errors C0169.....	463
17.11.91 Compiler errors C0174.....	463
17.11.92 Compiler errors C0175.....	464
17.11.93 Compiler errors C0177.....	464
17.11.94 Compiler errors C0178.....	464
17.11.95 Compiler errors C0182.....	465
17.11.96 Compiler errors C0185.....	465
17.11.97 Compiler errors C0189.....	465
17.11.98 Compiler errors C0190.....	466
17.11.99 Compiler errors C0190.....	466
17.11.100 Compiler errors C0195.....	466
17.11.101 Compiler errors C0196.....	466
17.11.102 Compiler errors C0197.....	467
17.11.103 Compiler errors C0198.....	467
17.11.104 Compiler errors C0199.....	467

	Page
17.11.105 Compiler errors C0201.....	468
17.11.106 Compiler errors C0203.....	468
17.11.107 Compiler errors C0204.....	468
17.11.108 Compiler errors C0205.....	469
17.11.109 Compiler errors C0206.....	469
17.11.110 Compiler errors C0208.....	469
17.11.111 Compiler errors C0211.....	469
17.11.112 Compiler errors C0212.....	470
17.11.113 Compiler errors C0216.....	470
17.11.114 Compiler errors C0217.....	470
17.11.115 Compiler errors C0218.....	470
17.11.116 Compiler errors C0219.....	471
17.11.117 Compiler errors C0221.....	471
17.11.118 Compiler errors C0222.....	471
17.11.119 Compiler errors C0224.....	472
17.11.120 Compiler errors C0227.....	472
17.11.121 Compiler errors C0228.....	472
17.11.122 Compiler errors C0234.....	473
17.11.123 Compiler errors C0235.....	473
17.11.124 Compiler errors C0236.....	474
17.11.125 Compiler errors C0237.....	474
17.11.126 Compiler errors C0238.....	474
17.11.127 Compiler errors C0239.....	474
17.11.128 Compiler errors C0240.....	475
17.11.129 Compiler errors C0241.....	475

Table of Contents

	Page
17.11.130 Compiler errors C0242.....	476
17.11.131 Compiler errors C0243.....	476
17.11.132 Compiler errors C0380.....	476
17.11.133 Compiler errors C0509.....	477
17.11.134 Compiler errors C0511.....	477
18 Reference user interface.....	479
18.1 Objects.....	479
18.1.1 Object 'General module folder'.....	479
18.1.2 "Application" object.....	480
18.1.3 "Data source manager" object.....	482
18.1.4 "Data source" object - "Variables" tab.....	483
18.1.5 "Data source" object - "Type mappings" tab.....	485
18.1.6 "Communication" tab.....	486
18.1.7 "Data source" object - "General and Diagnosis" tab.....	488
18.1.8 Object "DUT".....	488
18.1.9 "Device" object and generic device editor.....	490
Generic device editor.....	490
Tab "Parameters".....	491
"Applications" tab.....	491
Tab "Backup and restore".....	492
"Synchronized files" tab.....	494
"Files" tab.....	495
"Log" tab.....	495
'PLC settings' tab.....	497
"Parameters" tab.....	499
"PLC shell" tab.....	502
Tab '<Device name> I/O mapping'.....	504
Tab '<Device name> IEC objects'.....	506
"Users and groups" tab.....	507
"Access rights" tab.....	508
"Task list" tab.....	510
"Status" tab.....	511
"Information" tab.....	511
18.1.10 "GlobalTextList" object.....	512
18.1.11 "GVL" object - Global Variable List.....	512
18.1.12 'Persistent variable list' object.....	514
18.1.13 "Image pool" object.....	515
18.1.14 Object 'Library manager'.....	516
18.1.15 "Global Network Variable List (sender)" object.....	519
18.1.16 "Global Network Variable List (receiver)" object.....	519
18.1.17 Object "POU".....	520

	Page
"Program" object (PROGRAM).....	522
"Function block" object (FUNCTION_BLOCK).....	523
"Function" object (FUNCTION).....	525
"Interface" object (INTERFACE).....	529
"Method" object	532
"Interface method" object.....	534
"Interface property" object.....	534
Object 'Property'.....	537
Object "Action".....	540
'Transition' object.....	541
18.1.18 Monitoring the PLC program of the application at runtime.....	542
POU templates.....	543
18.1.19 "Project settings" object.....	545
18.1.20 "Library info" object.....	545
18.1.21 "Recipe manager" object.....	549
18.1.22 "Recipe definition" object.....	552
18.1.23 "Text list" object.....	553
18.1.24 "Symbol configuration" object.....	553
18.1.25 "Task configuration" object.....	562
"Properties" tab.....	562
"System events" tab.....	562
"Monitoring" tab.....	564
'Variable use' tab.....	565
'Task groups' tab.....	565
18.1.26 "Task" object.....	565
"Configuration" tab.....	566
18.1.27 "Trace" object.....	568
18.1.28 "Access control" object.....	571
18.2 Menu commands.....	572
18.2.1 Menu 'Edit'.....	573
Standard commands.....	573
Search and replace.....	573
'Find' command.....	573
'Replace' command.....	574
Command 'Search down'.....	575
Command 'Search up'.....	575
Command 'Search down' (selection).....	576
Command 'Search up' (selection).....	576
Bookmark.....	576
'Toggle bookmark' command.....	576
'Next bookmark' command.....	576
'Previous bookmark' command.....	577
'Delete bookmark' command.....	577
Insert file as text.....	577
Command 'Insert file as text'.....	577
Browse.....	577

Table of Contents

	Page
'Go to definition" command.....	577
Command 'Go to Reference'.....	578
Command 'Go to instance'.....	578
'Browse cross references' command.....	578
"Dump call tree" command.....	578
Refactoring.....	579
'Rename refactoring' command.....	579
Command 'Add variable'.....	580
Command 'Remove variable'.....	582
Command 'Rearrange variables...'.....	582
Command 'Refresh affected connections'.....	583
Advanced.....	583
'Show space' command.....	583
"Show indentation help lines" command.....	583
Command 'Expand all'.....	584
'Expand all' command.....	584
Command 'Uncomment selected lines'.....	584
Command 'Remove uncomment selected lines'.....	585
Command 'Go to'.....	585
Command 'Make upper case'.....	586
Command 'Make lower case'.....	586
Command 'Corresponding parenthesis'.....	586
Command 'Highlight up to corresponding parenthesis'.....	586
Input assistance	586
'Input wizard' command.....	587
Declare variable.....	588
Command 'Declare variable'.....	588
18.2.2 Menu 'View'.....	590
Project Explorer.....	590
Library.....	590
Other windows.....	590
"Tools" command.....	590
"Cross reference list" command.....	590
Command "Call tree".....	592
Command "Browse cross-references in classic view".....	594
"Element properties" command.....	594
'Memory' command.....	595
"Full screen" command.....	595
"Properties" command.....	596
"Startup screen" command.....	596
18.2.3 Menu 'Project'.....	596
"Project settings" command.....	596
"User management" command.....	596
Configuring the user management.....	597
"PLC compiler version" command.....	602
Command 'PLC project comparison'.....	603

	Page
“Add” command.....	605
“Add folder” command.....	605
Command “Display PLC objects of an external IndraWorks project”	606
“Edit object (offline)” command.....	608
Command ‘Set active application’	608
"Edit I/O mapping" command.....	609
Command ‘Import I/O mapping from CSV’	610
"Export I/O mapping to CSV" command.....	610
“Replace Global Variable List (GVL) in XML format” command.....	610
18.2.4 Menu ‘Create’	611
"Build" command.....	611
"Generate code" command.....	611
"Clean" command.....	612
"Rebuild" command.....	612
“Generate runtime system files” command.....	612
"Clean all" command.....	613
18.2.5 Menu ‘Debug’	614
Menu ‘Login’	614
Command ‘Logout’	617
Command ‘Create boot application’	617
Command ‘Log out current device user’	617
Command ‘Load’	618
Command ‘Online change’	619
Command ‘Start’	620
Command ‘Stop’	620
Command ‘Single cycle’	621
Command ‘Multiple download’	621
Command ‘Reset warm’	622
Command ‘Reset cold’	623
Command ‘Reset origin’	624
Command ‘Breakpoints’	625
Command ‘New breakpoint’	626
Command ‘Edit breakpoint’	627
Command ‘Switch breakpoint’	627
Command ‘Enable breakpoint’	627
Command ‘Disable breakpoint’	627
Command ‘Call list’	627
Command ‘Monitoring’ - ‘Monitoring list <n>’	628
Command ‘Monitoring’ - ‘Show all forces’	628
Command ‘Add all forces to monitoring list’	629
Command ‘Procedure step’	629
Command ‘Single step’	630
Command ‘Execute to return’	630
Command ‘Execute to cursor’	630
Command ‘Determine next statement’	631
Command ‘Show next statement’	631

Table of Contents

	Page
Command 'Force values'.....	631
Command 'Write values'.....	632
Command 'Cancel forcing for all values'.....	633
Command 'Switch sequence control mode'.....	633
Command 'Operating mode'.....	634
Command 'Core dump'.....	634
Command 'Create core dump'.....	634
Command 'Load core dump'.....	635
Command 'Load device logbook from core dump'.....	635
Command 'Close core dump'.....	636
Command 'Display mode'.....	636
18.2.6 Menu 'Tools'.....	636
Libraries.....	636
"Library repository" command.....	636
Generating a Library in IndraWorks.....	638
"Check for library compatibility" command.....	641
"Customize" command.....	641
"Options" command.....	641
18.2.7 Menu 'AS'.....	642
"Init step" command.....	642
"Insert step-transition" command.....	642
"Insert step-transition after" command.....	643
"Parallel" command.....	643
"Alternative" command.....	643
"Insert branch" command.....	644
"Insert branch right" command.....	644
"Insert action association" command.....	645
"Insert action association after" command.....	646
"Insert jump" command.....	646
"Insert jump after" command.....	647
"Insert macro" command.....	647
"Insert macro after" command.....	647
"Zoom into macro" command.....	647
"Zoom out of macro" command.....	648
'Insert after' command.....	648
18.2.8 Menu 'CFC'.....	648
"Edit worksheet" command.....	648
"Edit page size" command.....	648
"Negate" command.....	649
"Command EN/ENO" command.....	649
"None" command.....	649
"R-Reset" command.....	649
"S-SET" command.....	649
"Ref= (reference assignment)" command.....	650
Command 'Send to front'.....	650
Command 'Send to back'Symbol:.....	651

	Page
Command 'One ahead'.....	651
Command 'One back'.....	651
Command 'Order by data flow'.....	652
Command 'Order by topology'.....	652
"Set execution order" command.....	653
"Edit parameters" command.....	653
"Save prepared parameters to project" command.....	654
"Connect selected connections" command.....	654
"Unlock connection" command.....	655
"Show next collision" command.....	655
"Select linked connections" command.....	655
"Reset connections" command.....	656
"Remove unconnected connections" command.....	656
"Add input connection" command.....	656
"Add output connection" command.....	656
"Route all connections" command.....	656
"Remove control point" command.....	657
"Create control point" command.....	657
"Connection mark" command.....	657
"Create group" command.....	658
"Ungroup" command.....	658
Command 'Prepare function block element for forcing'.....	658
'Force FB input' command.....	658
Command 'Use attributed member as input'.....	659
18.2.9 Menu 'FBD/LD/IL'.....	661
"Insert network" command.....	661
"Insert network below" command.....	661
"Comment in/out" command.....	661
"Insert assignment" command.....	661
'Insert function block' command.....	662
"Insert function block with EN/ENO" command.....	662
"Insert empty function block" command.....	662
"Insert function block with EN/ENO" command.....	662
"Insert jump" command.....	663
"Insert label" command.....	663
"Insert return" command.....	663
"Insert input" command.....	663
"Insert coil" command.....	664
"Insert set coil" command.....	664
"Insert reset coil" command.....	664
"Insert contact" command.....	664
"Insert contact (right)" command.....	664
"Insert contact in parallel (below)" command.....	665
"Insert contact in parallel (above)" command.....	665
'Toggle parallel mode' command.....	665
"Insert negated contact" command.....	666

Table of Contents

	Page
"Insert negated contact in parallel (below)" command.....	666
"Insert contacts" command: "Insert below".....	666
"Insert contacts" command: "Insert above".....	666
"Insert contacts" command: Add right (after).....	667
"Insert IL line after" command.....	667
"Delete IL line" command.....	667
"Negation" command.....	667
"Edge detection" command.....	668
"Set/reset" command.....	668
"Set output connection" command.....	668
"Insert line branching" command.....	668
"Insert line branching above" command.....	668
Command "Insert line branching below" command.....	669
"Set line branching start point" command.....	669
"Set line branching end point" command.....	669
"Update parameters" command.....	669
"Remove unused FB call parameters" command.....	670
"Repair block" command.....	670
"Display as function block diagram" command.....	670
"Display as ladder diagram" command.....	671
"Display as instruction list" command.....	671
"Go to" command.....	671
18.2.10 Menu 'Library'.....	671
"Add library" command.....	672
"Try to reload library" command.....	673
"Properties" command.....	673
"Placeholder" command.....	675
"Library info" command.....	676
18.2.11 Menu 'Image pool'.....	676
"Insert image".....	676
18.2.12 "Declarations" menu.....	677
"Edit declaration header" command.....	677
"Shift downwards" command.....	677
"Shift upwards" command.....	677
18.2.13 Menu 'Declarations' (persistence).....	678
"Rearrange list and clean gaps".....	678
"Save current values to recipe" command.....	678
"Restore values from recipe" command.....	679
Command 'Add all instance paths'.....	679
18.2.14 Menu 'Recipes'.....	680
"Insert variable" command.....	680
"Add a new recipe" command.....	680
"Remove recipe" command.....	680
"Load Recipe" command.....	681
"Save recipe" command.....	681
"Read recipe" command.....	681

	Page
"Write recipe" command.....	682
"Load and write recipe" command.....	682
"Read and save recipe" command.....	683
"Remove variable" command.....	683
"Update structured variables" command.....	683
18.2.15 Menu 'Text list'.....	684
"Add language" command.....	684
"Create global text list" command.....	684
"Export all as text" command.....	685
"Export all as Unicode text" command.....	685
"Insert text" command.....	685
"Import/export text lists" command.....	686
"Remove language" command.....	687
"Remove unused text list entries" command.....	687
"Check visualization text IDs" command.....	687
"Update visualization text IDs" command.....	688
"Add text list support" command.....	688
"Remove text list support" command.....	688
18.2.16 Menu 'Trace'.....	688
"Add variable" command.....	688
'AutoFit' command.....	689
"Compress" command.....	689
"Configuration" command.....	689
"Cursor" command.....	689
'Download trace' command.....	690
"Export symbolic trace config" command.....	691
'Load trace' command	692
"Zoom with mouse" command.....	692
Command 'Convert to multi-channel'.....	692
Command 'Convert to single-channel'.....	693
Command 'Online list'.....	694
"Reset trigger" command.....	695
"Reset to default view" command.....	695
"Save trace" command.....	695
"Start trace" command.....	696
"Stop trace" command.....	696
"Stretch" command	696
Command 'Upload trace'.....	697
"Statistics" command.....	697
18.2.17 Miscellaneous.....	698
"Simulation" command.....	698
"Add to watch list" command.....	699
"Implement interfaces" command.....	699
Command 'Limit results to current declaration'.....	699
"Stop execution on handled exceptions" command.....	700
18.3 Dialogs.....	700

Table of Contents

	Page
18.3.1	Dialog "Select function block"..... 700
18.3.2	"Breakpoint properties" dialog'..... 701
18.3.3	"Prepare value"..... 701
18.3.4	Dialog 'New breakpoint'..... 702
18.3.5	Dialog "Monitoring range"..... 704
18.3.6	"Properties" dialog..... 704
	"Properties" - "General" dialog..... 705
	"Properties" - "Boot application"..... 705
	"Properties" - "Build" dialog..... 705
	"Properties" - "Access control" dialog..... 706
	"Properties" - "Bitmap" dialog..... 707
	"Properties" - "Application build options" dialog..... 707
	The "Properties" - "Network variables" dialog..... 708
	"Properties" - "Network properties" dialog..... 709
	"Properties" dialog - "CFC execution order" tab..... 709
	"Properties" - "SFC settings" dialog..... 712
	"Properties" - "Linking to file" dialog..... 712
	"Properties" - "Image Pool" dialog..... 712
	"Properties" - "Options" dialog..... 713
18.3.7	"Project settings" dialog..... 714
	"Project settings" - "SFC" dialog..... 714
	"Project settings" - "Compiler options" dialog..... 715
	"Project settings" - "Compiler warnings" dialog..... 716
	"Project settings" dialog - "Static analysis light"..... 716
	"Project settings" dialog - "Visualization"..... 719
18.3.8	"Properties" - "Text list"..... 720
18.3.9	"Options" dialog..... 720
	"Options" dialog - "General Settings"..... 720
	"Options" - "CFC editor" dialog..... 721
	"Options" - "Declaration editor" dialog..... 722
	Dialog 'Options' - 'Device editor' 722
	"Options" - "FBD, LD and IL" dialog..... 723
	"Options" dialog - "IndraLogic 1x converter"..... 725
	"Options" - "Libraries" dialog..... 726
	"Options" - "Refactoring" dialog..... 726
	"Options" dialog - "Safe FBD options"..... 727
	"Options" - "SFC editor" dialog..... 728
	"Options" dialog - "Smart coding "..... 730
	"Options" - "Syntax Highlighting" dialog..... 731
	"Options" dialog - Text editor..... 731
	"Options" dialog - Visualization..... 734
	"Options" dialog - "Visualization style"..... 735
18.3.10	"Trace configuration" dialog..... 736
	"Advanced trace settings" dialog..... 736
	'Trace configuration' dialog..... 737

	Page
19 Libraries.....	743
19.1 Guidelines to create libraries.....	743
19.2 "Recipe management" library.....	743
20 IndraLogic 2G visualization.....	757
20.1 Visualization overview.....	757
20.1.1 Displaying a preview of the library visualization.....	759
20.1.2 Displaying and labeling visualization elements.....	759
20.1.3 Using the symbol library in the visualization.....	763
20.2 Commands.....	765
20.2.1 "Add visual element" command.....	766
20.2.2 Command "Interface editor".....	766
20.2.3 Command "Keyboard usage".....	766
20.2.4 Command "Element list".....	766
20.2.5 Command "Activate keyboard usage".....	766
20.2.6 Alignment.....	767
Command "Align left".....	767
Command "Align top".....	767
Command "Align right".....	767
Command "Align bottom".....	767
Command "Center vertically".....	767
Command "Center horizontally".....	767
Command "Equalize horizontal distance".....	767
Command "Increase horizontal distance".....	767
Command "Decrease horizontal spacing".....	768
Command "Remove horizontal distance".....	768
Command "Equalize vertical distance".....	768
Command "Increase vertical distance".....	769
Command "Decrease vertical distance".....	769
Command "Remove vertical distance".....	769
Command "Equalize width".....	770
Command "Equalize height".....	770
Command "Equalize size".....	770
Command "Adjust size to grid".....	770
20.2.7 Order.....	771
"On level up" command.....	771
Command "Place on top".....	771
Command "One level down".....	771
Command "Send to back".....	771
20.2.8 Command "Group".....	771
20.2.9 "Ungroup" command.....	772
20.2.10 "Frame selection" commands.....	772
20.2.11 Command "Background".....	773
20.2.12 Command "Select all".....	774
20.2.13 Command "Deselect all".....	774
20.2.14 "Configure trace..." command.....	774

Table of Contents

	Page
20.2.15	“Insert elements for trace control” command..... 774
20.2.16	“Update frame parameters in <opened visualization>” command..... 774
20.2.17	“Multiply visualization element” command..... 775
20.3	Editor..... 777
20.3.1	“Tools” view..... 778
20.3.2	“Properties” view..... 780
20.3.3	“Interface editor” tab..... 782
	Basic information on the interface editor..... 782
	Updating the frame parameters..... 784
	Pragmas..... 787
	“VAR_IN_OUT_AS_POINTER” attribute..... 787
	Attribute 'parameterstringof'..... 790
20.3.4	Tab 'Keyboard configuration'..... 791
20.3.5	Tab “Element list”..... 793
20.3.6	“Add visualization” dialog..... 794
20.3.7	“Configure IDs and entries” dialog..... 794
20.4	Visualization elements..... 795
20.4.1	Configuring elements - General information..... 795
	User input when editing an element..... 795
	Using text and language 796
	Input configuration..... 800
	Gradient editor..... 814
	Using semitransparent colors..... 815
	Multiplying an element..... 817
	Dragging an IEC variable to an element..... 819
20.4.2	'Basis'..... 821
	Element “Rectangle”, “Rounded rectangle” and “Ellipse”..... 821
	‘Line’ element..... 830
	Element ‘Polygon’, ‘Polyline’, ‘Bézier curve’..... 838
	“Circle segment” element..... 847
	Image and its dialog..... 856
	Element 'Image'..... 856
	Dialog “Input assistant”..... 865
	Element 'Frame'..... 866
20.4.3	‘General control elements’..... 876
	Element “Text”..... 876
	“Combo box, integer” element..... 878
	“Combo box array” element..... 882
	Element “Tab control element”..... 884
	Element 'Button'..... 887
	“Group box” element..... 896
	“Table” element..... 897
	“Text field” element..... 908
	Element “Scroll bar”..... 914
	Element “Slider”..... 919
	“SpinControl” element..... 922

	Page
“Invisible input” element.....	926
“Progress bar” element.....	929
“Radio button” element.....	931
“Checkbox” element.....	932
20.4.4 'Measuring devices'.....	934
“Bar diagram” element, “Bar diagram freely configurable”.....	934
Element “Pointer instrument 90°”.....	939
Element “Pointer instrument 180°”.....	943
Element “Pointer instrument 360°”.....	947
“Potentiometer” element.....	951
“Histogram” element.....	955
20.4.5 'Lamps/switches/images'.....	960
“Image slider” element.....	960
“Lamp” element.....	963
Element “dip switch”, “power switch”, “push switch”, “push switch LED” and “rocker switch”.....	966
“Rotary switch” element.....	969
20.4.6 'Special control elements'.....	972
“Trace” element and their dialogs.....	972
“Trace” element.....	972
“Trace configuration, recording settings” dialog.....	981
“Trace configuration, variable setting” dialog.....	983
“Advanced trace settings” dialog.....	985
Dialog 'Edit Appearance Settings'.....	986
Dialog “Copy settings from trace instance”.....	988
Dialog “Trace wizard”.....	988
Command “Export trace configuration”.....	988
Element “ActiveX element”.....	989
“Busy symbol Flower” element.....	991
20.4.7 Visualization Properties.....	993
Property 'Diagram'.....	993
Property 'Element Name'.....	993
Property 'Number format'.....	993
Property 'Alignment'.....	993
Property 'Position'.....	993
Property 'Display frame'.....	994
Property 'Display tooltip'.....	994
Property 'Text properties'.....	994
Property 'Text properties'.....	995
Property 'Texts'.....	995
Property 'Labeling of the scale lines'.....	995
Property 'Times'.....	996
20.5 Visualization manager with clients.....	996
20.5.1 Visualization manager.....	996
“Visualization manager” - “Settings”.....	996
“Visualization manager” - “Default keyboard configuration”.....	998
“Visualization manager” - “Visualization”.....	998

Table of Contents

	Page
“Font settings” tab.....	999
20.5.2 Clients and client editors.....	999
Diagnostic visualization.....	999
20.6 Dialog “Properties” - “Visualization object”.....	1001
20.6.1 Use cases.....	1002
Input dialogs.....	1002
Global dialog.....	1007
Programmed access to input dialogs.....	1007
Noticing events and input actions.....	1011
20.7 Visualization libraries.....	1015
20.8 Options.....	1016
20.8.1 “Options” dialog - “Visualization”.....	1016
20.8.2 Dialog “Options” - “Visualization styles”.....	1016
20.9 Visualization profile.....	1016
20.10 Visualization style.....	1017
20.11 Keyboard operation in online mode.....	1018
 21 Service and support.....	 1019
 Index.....	 1021

1 About this documentation

Editions of this documentation

Edition	Release Date	Note
01	2020-11	First edition of 15VRS

Tab. 1-1: Change Record

1.1 Validity of the documentation

Target group This documentation is intended for users who program and commission the PLC component of a control of the systems MLC, MLD or MTX or intend to support it by an internally used visualization.

Required qualification: Individual who is able to assess the tasks assigned and to identify possible safety risks owing to qualification in the subject, knowledge and experience. The individual should also be familiar with the standards and regulations.

This documentation describes the PLC programming tool IndraLogic 2G and its usage.

The free IndraLogic is an IEC 61131-3 programming platform for automation devices with control tasks. It provides multiple and comfortable engineering functions for your engineering tasks:

	Refer to
Project configuration by wizards	chapter 2 "Creating and configuring projects" on page 7
Creation of professional control applications acc. to IEC 61131-3 with a high number of default functions	chapter 9 "Programming applications" on page 83
Comfortable programming with mouse and keyboard in all languages of the IEC 61131-3	chapter 17.1 "Programming languages and editors" on page 223
Corresponding editors for FBD, LD, IL, ST, SFC and the variants CFC and extended CFC	chapter 17 "Reference "Programming"" on page 223 chapter 18 "Reference user interface" on page 479
Input assistance for entering and configuring different data	chapter 9.5 "Using input assistants" on page 107
Support for object-oriented programming Real object-oriented programming acc. to 61131-3 3rd edition in all languages of the IEC 61131-3 is possible without additional tools Inheritance of program function blocks to similar application components to reduce development time and errors Object-oriented programming is not mandatory: Functional or object-oriented programming can be freely used and combined	chapter 9.18 "Object-oriented programming" on page 145
Comprehensive comparison of PLC objects and for graphic editors Comprehensive online comparison of PLC objects	chapter 4 "Comparing PLC objects" on page 17 chapter 5 "Comparing PLC objects online" on page 51
Debugging and online properties to optimize the application code and to accelerate test and commissioning	chapter 12 "Testing and debugging" on page 179 Command overview chapter 18.2.5 "Menu 'Debug'" on page 614

	Refer to
Integrated compilers for multiple CPU platforms to optimize the control performance	chapter ""Project settings" - "Compiler options" dialog" on page 715
Security properties to secure the source code and the operation of the control	chapter 6 "Protecting and saving projects" on page 57 chapter 18.1.28 ""Access control" object" on page 571
Field bus support and programming of devices	chapter 8 "Configuring an I/O link" on page 69
Library concept for a simple re-use of application code	chapter 16 "Creating and managing libraries" on page 219
Diagnostic visualization	chapter 20.1 "Visualization overview" on page 757

For information on the Bosch Rexroth customer service help desk, refer to [chapter 21 "Service and support" on page 1019](#).

This documentation supports the user during the following phases:

- Programming the PLC program
- Debugging and commissioning
- Visualization

1.2 Using safety instructions

1.2.1 Structure of the safety instructions

The safety instructions are structured as follows:

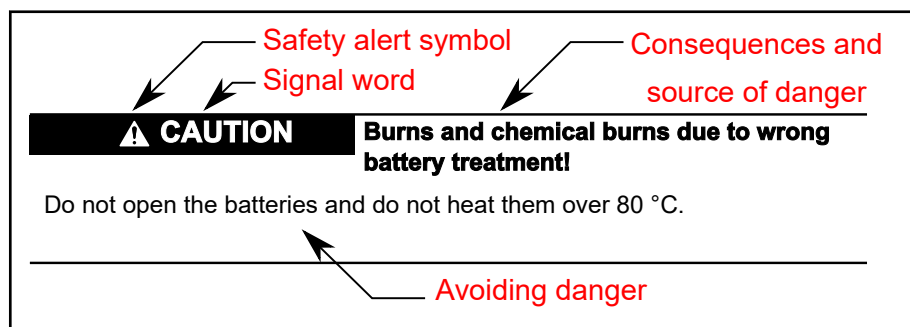


Fig. 1-1: Structure of the safety instructions

1.2.2 Explaining signal words and safety alert symbol

The safety instructions in this documentation contain specific signal words (danger, warning, caution, notice) and, if necessary, a safety alert symbol (according to ANSI Z535.6-2006).

The signal word draws attention to the safety instruction and indicates the risk potential.

The safety alert symbol (triangular safety reflector with exclamation marks), preceding the signal words Danger, Warning, Caution indicates hazards for persons.



In case of non-compliance with this safety instruction, death or serious injury **will** occur.

** WARNING**

In case of non-compliance with this safety instruction, death or serious injury **can** occur.

** CAUTION**

In case of non-compliance with this safety instruction, minor or moderate injury can occur.

NOTICE

In case of non-compliance with this safety instruction, material damage can occur.

1.2.3 Symbols used

Pointers are displayed as follows:



This is a note.

Tips are displayed as follows:



This is a tip.

1.2.4 Explaining the signal alert symbol on the device



If this symbol is on your device, you have to observe the documentation on the device. The respective documentation informs on the type of hazard as well as the steps required to avoid this hazard.

1.3 Required and supplementing documentation

Documentation titles with type codes and part numbers

IndraWorks 15VRS Software Installation DOK-IWORKS-SOFTINS*V15-CORS-EN-P, R911393450 This documentation describes the IndraWorks installation.
IndraWorks 15VRS Engineering DOK-IWORKS-ENGINEE*V15-APRS-EN-P, R911393303 This documentation describes the use of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.
IndraWorks 15VRS Field Buses DOK-IWORKS-FB*****V15-APRS-EN-P, R911393284 This documentation describes the field bus and local periphery connections supported by the MLC and MTX systems. The focus of this documentation lies in the configuration, parameterization, commissioning and diagnostics of different periphery connections. It is the basis for the online help.
IndraWorks 15VRS Basic Libraries IndraLogic 2G DOK-IL*2G*-BASLIB**V15-LIRS-EN-P, R911398633 This documentation describes the system-comprehensive PLC libraries.
IndraWorks 15VRS - Field Bus Libraries DOK-IWORKS-FB*LIB**V15-LIRS-EN-P, R911393275 This manual describes the field bus libraries: RIL_ProfibusDP_02, RIL_ProfibusDP Slave, RIL_ProfinetIO, RIL_ProfinetIO-Device, RIL_EtherNetIPAdapter, RIL_MappingList, RIL_SERCOSIII, RIL_Inline including their diagnostics and error reactions of the function blocks.
IndraMotion MLC 15VRS PLCopen Libraries DOK-MLC***-FUNLIB**V15-LIRS-DE-P, R911393307 This documentation describes the function blocks, functions and data types of the RIL_CommonTypes, ML_Base and ML_PLCOpen libraries for IndraMotion MLC. The documentation also includes the error reactions of the function blocks.

IndraMotion MLC 15VRS HMI Connection

DOK-MLC***-HMI*****V15-APRS-EN-P, R911399270

This documentation describes the visualization systems supported by the MLC and their connection.

Rexroth IndraMotion MLC 15VRS First Steps

DOK-MLC***-F*STEP**V15-CORS-EN-P, R911398631

This documentation describes the first steps of the IndraMotion MLC and the RobotControl. It includes the hardware and software prerequisites as well as the creation of a project.

Tab. 1-2: Documentation overview

1.4 Names and abbreviations

Term	Explanation
SFC	Structuring medium "Sequential Function Chart"
IL	PLC programming language "Instruction List"
CFC	PLC programming language "Continuous Function Chart"
DUT	Data type
EN/ENO	Boolean input (ENable) EN / Boolean output ENO (ENable Out)
FBD	PLC programming language "Function Block Diagram"
GNVL	Global Network Variable List
GVL	Global Variable List
IEC	International Electrotechnical Commission (Standardization platform for electrotechnical engineering)
IndraWorks Engineering Framework	Configuration and commissioning tool by Bosch Rexroth
LD	PLC programming language "Ladder Diagram"
POU	"Program Organization Unit"

Tab. 1-3: Names and abbreviations used

2 Creating and configuring projects

What is a project?

- A project contains objects required to create a control program ("application"):
 - Pure programming function blocks, such as programs, function blocks, functions and GVLs.
 - Additionally required objects to execute the application on a PLC, such as the task configuration, the library manager, the symbol configuration, the device configuration, visualizations and external files.
- Multiple control units can be programmed and linked to a project with one application each.
- IndraLogic manages application-specific function blocks in the Project Explorer below "Application".
IndraLogic manages project-globally available function blocks in the Project Explorer in the "General module" folder.
- To create projects, there are templates with specific objects.
- For basic the configuration and information on the project, go to **Project settings** and **Library info**. Example:
 - Compiler settings
 - User management
 - Author
 - Data on project file.

For settings on the version compatibility of the project, go to the configuration dialogs on the **project environment**.

- Save a project as file in the file system. Optionally, it can be stored together with project-relevant files and information in a project archive. Storage is also possible in a source code management system such as SVN.
- Each project includes information on the IndraLogic version used for creation. When opening it with another version, IndraLogic informs on possible or required updates with regard to memory format, library version, etc.
- Projects can be compared, exported, imported and a documentation can be created.
- A project can be write-protected and read-protected. Use a user management to specifically access the projects as well as individual objects in the project.

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 18.1.2 "'Application' object" on page 480](#)
- [chapter 18.1.17 "Object 'POU'" on page 520](#)
- [chapter 18.3.7 "'Project settings' dialog" on page 714](#)
- [chapter "'Library info' command" on page 676](#)
- [chapter 6 "Protecting and saving projects" on page 57](#)

2.1 Creating “empty project”

1. Select **File** ► **New project** and then the template **General templates** ► **Empty project**. Enter a name, e.g. `myProject` and a storage location in the file system.

The project `myProject` is displayed in the Project Explorer.

2. Select **Add** ► **IndraMotion MLC** ► **IndraControl L65** from the context menu of the project for example. Further information is prompted with the dialogs of the wizard "Insert <Control>".

The project name **MyProject** is displayed in the title bar of the main window of the programming system.

The following structure is created in the Project Explorer:

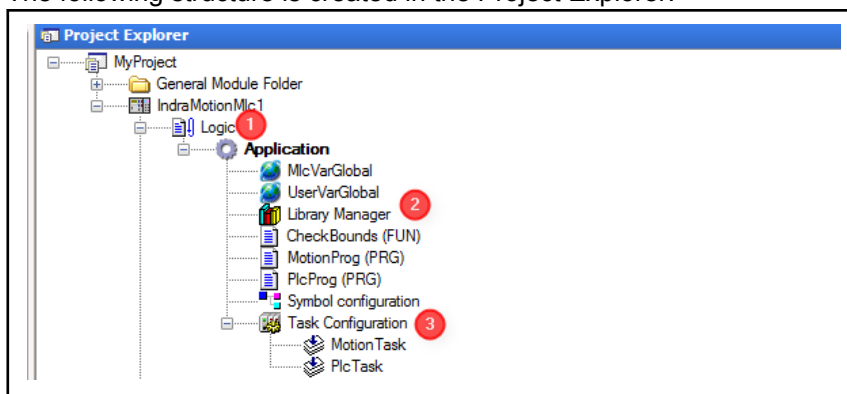


Fig. 2-1:

Depending on the selected control, additional device nodes can be displayed.

The PLC is always automatically displayed with a subnode (**Logic** (1)).

The basic objects are: an application with a **Library manager** (2) and a **Task configuration** (3).

The task configuration defines the main task (**PlcTask**) for processing the standard function block **PlcProg**.

The library manager already contains a set of libraries (depending on the selected control) that can be subsequently used for programming.

If correct code is entered into **PlcProg**, the code can be loaded to the control and executed without further programming objects.

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 2.2 "Adding objects" on page 8](#)
- [chapter 9 "Programming applications" on page 83](#)
- [chapter 8 "Configuring an I/O link" on page 69](#)
- [chapter 9.7 "Using library function blocks" on page 112](#)

2.2 Adding objects

Prerequisite: A project is open.

Refer to

- [chapter 2.1 "Creating “empty project”" on page 8](#)

The following instruction exemplarily shows how objects can be created in the project.

1. Select an entry in the Project Explorer or in the "General module folder", such as the **Application** object in the Project Explorer.
2. Select **Add** from the context menu.
Depending on the selected entry in the Project Explorer, IndraLogic provides the corresponding objects for selection.
3. For example, select the **POU** object and in the next dialog **Add object wizard** the type **Program** with the implementation language **Structured text (ST)** and the name "prog".
Click on **Finish** and IndraLogic inserts a program object **prog** into the Project Explorer below **Application**. The object is application-comprehensively known.
4. Select an object in the Project Explorer and then **Properties** from the context menu.
The **Properties** dialog is displayed with the object-relevant categories. When using a user management, access to the object can be restricted for example.
5. Select an entry in the Project Explorer to create a folder below this entry to collect certain objects.



Self-created folders cannot structure the arrangement of device nodes and device objects.

6. Select **Add folder** from the context menu of the **Application** object and define a name for the new folder in the dialog. Click on **OK**.
The folder is displayed in the Explorer.
7. Select an object in the tree and drag it to another position within the tree (e.g. to the folder) using the mouse.
8. Select the "General module folder" in the Project Explorer. Select **Add** and then for example the **Text list** object.
The **Text list** object is inserted into the "General module" folder! The object is project-comprehensively known.

Also refer to

- [chapter 2.1 "Creating "empty project"" on page 8](#)
- [chapter ""Add" command" on page 605](#)
- [chapter ""Add folder" command" on page 605](#)
- [chapter ""Properties" command" on page 596](#)

2.3 Modifying PLC compiler version

Working with IndraWorks,
compatibility mode

Project ► Change compatibility mode...

PLC compiler version

From IndraWorks 15VRS, no versions older than <15VRS are supported in the compatibility mode!

Projects from previous versions have to be converted to be used in 15VRS.

During the conversion, it might be required to change the control type and the configuration.

Please note the related documentations:

For the documentations, go to the Rexroth media directory:

- System MLC Project Conversion: DOK-MLC***-PRO-CONV*V15-APRS-EN-P
- System MTX Project Conversion: DOK-MTX***-PRO-CONV****-PRRS-EN-P



When opening a project and the latest compiler version is not set, a **Project environment** dialog is displayed with the note and option to directly update.

Prerequisite: A project is open.

To display the compiler version (cannot be changed), go to "General module folder", and then under **Project settings** ► **Compiler options**.

Also refer to

- [chapter 11.2 "Generating application code" on page 175](#)
- [chapter ""Project settings" - "Compiler options" dialog" on page 715](#)

2.4 Opening IndraLogic 2G project

Prerequisite: IndraLogic 2G is started. A project might already open.

1. Select **File** ► **Open project**.
2. In the **Open project** dialog, select the IndraLogic 2G project or project archive from the file system. There is a file filter in the right lower corner of the dialog.



If required, open the project "write-protected" to avoid that the existing project file is not subsequently overwritten with changes by default.

If no other project is already open, the following scenarios are possible:

- The project selected was saved with a later IndraLogic version:
Such a project can contain data that cannot be loaded. The project can still be opened. However, consider the following: The project might show an unexpected response, as it cannot be interpreted completely.

Objects, which could not be loaded completely or not loaded at all, are displayed in red in the Project Explorer and labeled with the text "[unknown]" or "[incomplete]".

IndraLogic cannot display objects unknown in the current version in the editor.

IndraLogic outputs a warning in case of incomplete objects. The warning shows that the displayed content might not correspond to the original.

IndraLogic cannot save incomplete projects under its original name. This is displayed by a write-protection note in the upper right corner. The file can be saved with a different name.

- IndraLogic did not properly close the project selected after its last change, but the project option **Save automatically** was enabled: To handle the backup copy, the **Auto save backup** dialog is displayed.
- The project selected is write-protected: The project can now be opened as write-protected or the write-protection can be disabled.
- The project selected is write-protected. You are now prompted to enter the password.

The **Project environment** dialog opens.

3. Check for possible updates in the **Project environment** dialog. For projects created with very old IndraLogic 2G versions, additional notes can be displayed on how to replace certain objects for further project use. You have to decide whether to update the project.

Upon each update, logging in without download/online change is not possible anymore.

The project is open.

Also refer to

- [chapter 6.1 "Setting up write protection" on page 59](#)

2.5 Configuring projects

The IndraLogic project can be configured in the following dialogs:

- **Project settings:** General settings for the response of the editors, the compiler, the user management, etc.
- **Library info:** Providing the project with individually labeled information

Also refer to

- [chapter 18.3.7 "'Project settings' dialog" on page 714](#)
- [chapter 18.1.20 "'Library info' object" on page 545](#)

2.5.1 Retrieving and editing library information

Use the **Library info** object to retrieve information on your project and the respective file and to edit certain information.

The object contains information on

- File attributes
- Meta-information, such as manufacturer, title, or author
- Properties with key
- Statistics

- Signed libraries

IndraLogic directly saves the library info as object to the project. When transferring a project to another system, the **Library info** object is also transferred without a project archive being required.

Use property keys to externally access library info via functions. For a library project, information on licensing can additionally be queried and the file can be signed.

Also refer to

- [chapter 18.1.20 "Library info" object](#) on page 545

Editing meta-information

1. For the steps described, it is required that the object "Library info" was added to the project before.

In the Project Explorer, open the "General module folder" and the "Library info object" in that folder.

The **Library info** dialog opens.

2. Select the **Summary** tab.
3. Enter your data in the input fields (e.g., 0.0.0.1 in the **Version** field).

IndraLogic creates a property with key for each entered value and manages them in the **Properties** tab. For a library project, IndraLogic continues using the properties and sorts the library repository accordingly.

If the option - that IndraLogic creates a function automatically for these properties - is selected, the properties can be accessed programmatically.

Creating functions for property access

- 1.

For the steps described, it is required that the object "Library info" was added to the project before.

In the Project Explorer, open the "General module folder" and the "Library info object" in that folder.

The **Library info** dialog opens.

2. Select the option **Automatically creating POU's for the property access**.

Prerequisite: The following property is defined.

Example

```
Schlüssel = nProp1
Typ= Zahl
Wert= 333
```

1. Select the option **Automatically creating POU's for the property access**.
2. Declare a variable of the type DINT, e.g. `showprop : DINT;`
3. Call the function `GetNumberProperty: showprop := GetNumberProperty("nProp1");`

You are granted access to the value in the application.

Signing library project

Prerequisite: The project is a library project. A private key file with an associated token is available.

1. For the steps described, it is required that the object "Library info" was added to the project before.

In the Project Explorer, open the "General module folder" and the "Library info object" in that folder.

The **Library info** dialog opens.

2. Select the **Signing** tab.
3. Select the option **Activate signing**.
4. Enter the storage location of your private key file.

IndraLogic enters the token automatically into **Public key token** (example: 427A5701DA3CF3CF).

5. Save the project as a **Compiled library**.

The token in the library repository or the library manager can be seen in the library details. It can be compared to the token published by the library manufacturer. This checks whether the library actually originates from the intended vendor. In addition, a library, which is manipulated at the binary level, does not match the signature anymore and is therefore not loaded. Thus, you can rely on the integrity of the library.

Creating private key file

1. For the steps described, it is required that the object "Library info" was added to the project before.

In the Project Explorer, open the "General module folder" and the "Library info object" in that folder.

The **Library info** dialog opens.

2. Select the **Signing** tab.
 3. Select the button **Create private key file**.
- The **Create private key file** dialog opens.

4. Select a safe storage location, e.g. D:\for lib developers only \mycomp_libkey.libpk and exit the dialog with **Save**.

Also refer to

- [chapter 18.1.20 "Library info" object on page 545](#)

2.5.2 Applying project settings

Make settings affecting the behavior of IndraLogic and that of certain editors in the **Project settings** object.

The settings are valid throughout the project and are applied immediately for active editors.

For the dialogs of the object, go to **General module folder ► Project settings**.

IndraLogic directly saves the project settings as object in the project. When transferring a project to another system, the **Project settings** object is also transferred without a project archive being required.

Also refer to

- [chapter "Project settings" command on page 596](#)
- [chapter "Project settings" - "SFC" dialog on page 714](#)
- [chapter "Project settings" - "Compiler options" dialog on page 715](#)
- [chapter "Project settings" - "Compiler warnings" dialog on page 716](#)
- [chapter "Project settings" dialog - "Static analysis light" on page 716](#)
- [chapter "Project settings" dialog - "Visualization" on page 719](#)

3 Exporting and transferring projects

There are import and export functions to exchange IndraLogic project data with other programs.

IndraLogic projects are exchanged between IndraLogic development systems using a copy of the export file (*.iwx) or a zipped export file (*.iwxz).

3.1 Transferring projects

When transferring a project to another computer and connecting to the same PLC from that computer without an online change or a download being required, observe the following:

- Ensure that the project requires only specified versions of the libraries (except for interface libraries), the visualization profile and the compiler
- Ensure that the boot application is the current one.

Create a project archive to unzip it on another computer.

Transferring a project to another system

Prerequisite: A project is open on the computer "PC1". Transfer this project to another computer "PC2" and connect it again to the same control.

1. Ensure that only libraries with specified versions are integrated into the project, except for pure interface libraries. Open the **Library manager** and check all entries with an "*" instead of a specified version information.
2. Ensure that specified compiler version is set in the project settings. For checking purposes, select **General module folder ► Project settings** and the category **Compiler options**.
3. Ensure that the currently open application is the same application as currently used on the PLC. Thus, the "boot application" has to be identical to the project in the programming system. Check the programming name in the title bar of the programming system window: If there is an asterisk behind the name, the project was changed, but not yet saved. This can show that application and boot application do not match!

In this case, generate (new) boot application first. It depends on the PLC and application properties, whether this is automatically executed when downloading the application. For explicit creation, select **Debug ► Generate boot application**. To download, select **Debug ► Login and Debug ► Load**.

Then start the application on the control with **Debug ► Start**.

The application - to connect again later on to "PC2" from the same project - is now running.

4. Create a project archive: Select **Project ► Archive**.
Also select the following information in the **Archive project** dialog:
 - **Add program code to archive**
 - **Add standard libraries to archive**
 Save the project archive at a location accessible by PC2.
5. Log out of the control: Select **Online ► Logout**. The PLC can be stopped and restarted without causing any problems before connecting again from "PC2".

6. Unzip the project archive to "PC2": Select **Project ► Restore** and open the archive saved above. Enable the same information and described before when creating the archive in the **Extract project archive** dialog.
7. Open the project and log on to the PLC "xy" again.
IndraLogic does not request an online change or download. The project is running.

Also refer to

- [chapter ""Project settings" - "Compiler options" dialog" on page 715](#)
- [chapter "Command 'Create boot application'" on page 617](#)

4 Comparing PLC objects

Comparing PLC objects is permitted:

1. to compare PLC objects of the **current** project with each other,
2. Comparing PLC objects of the **current** project to those of an export file (*.iwx, *.iwxz)
3. to compare PLC objects of two export files with each other.

In the first case, the desired objects are to be highlighted in the Project Explorer.

In the second case, the desired object is to be highlighted in the Project Explorer and "Compare..." is to be selected in the context menu of the object. In the appearing window, the desired export file has to be selected.

To compare two export files, select **Tools ► Compare...** from the context menu.

When comparing PLC projects as well as PLC objects, you can do the following:

- Ignore spaces. Only semantically relevant spaces are then individually considered
- Ignore comment. Only program-relevant text is then compared
- Ignore properties of PLC objects

Also refer to: [chapter 5.3 "Options" on page 54](#).

The description is made based on the combination of project/export file. This applies similarly to export file/export file.

If the comparison cannot be made, since the object types differ, an error message is displayed.

Basics of the comparison in IndraWorks Engineering:

- Rexroth IndraWorks 15VRS Engineering
DOK-IWORKS-ENGINEE*V15-APRS-EN-P, R911393303

Comparing PLC objects

- [General module folder, page 18](#)
- [Logic, page 19](#)
- [Application, page 20,](#)
- [Task configuration, page 23](#)
- [Task, page 18](#)
- [Folder \(Difference from IndraWorks folder\), page 25](#)
- [Data types, page 25](#)
- [Global variable list, persistent variables, network variables, page 27](#)
- [Network variable lists, receiver end, page 28](#)
- [Programs, function blocks, functions..., page 29](#)
- [Interfaces, interface methods, interface properties, page 36](#)
- [Library manager, page 37](#)
- [Library info, page 38](#)
- [Data server, page 39,](#)
- [Data source \(data server\), page 41,](#)
- [Symbol configuration, page 42,](#)

Comparing PLC objects

- Visualization manager, page 42,
- Visualization, page 43
- Text list, page 43,
- Image pool, page 44
- Recipe manager, page 44
- Recipe definition, page 45
- Devices (onboard I/O, Inline I/O, field buses, ...), page 46.

4.1 Comparison: General module folder

Here only the name is displayed in the window. "Merging" is not possible, since the element is fixed.

Differences can result based on the name or language.

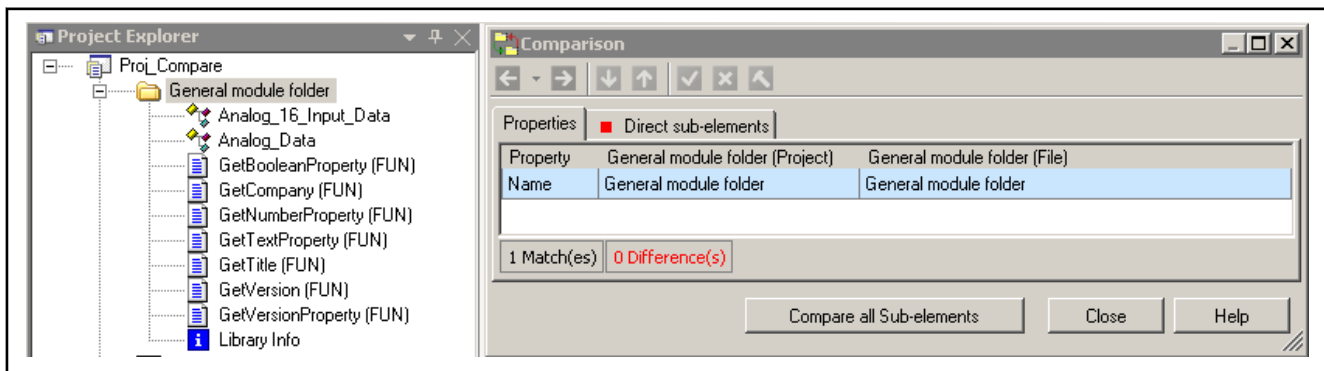


Fig. 4-1: Comparison: General module folder

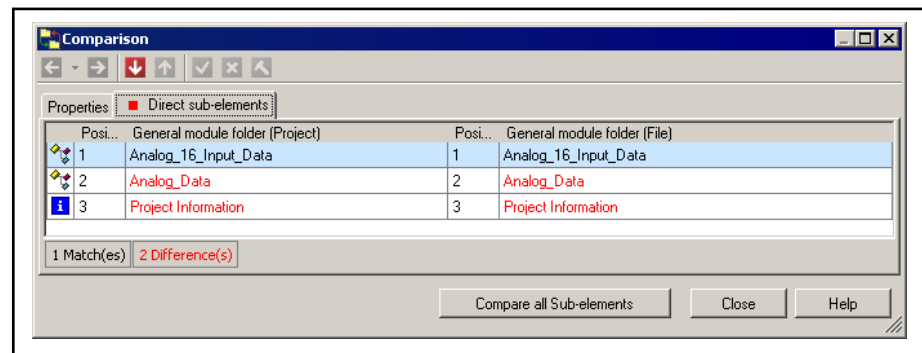


Fig. 4-2: Comparison: Listing of the "Direct subelements", two elements with differences (red)

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer for merging.

Select the cross to prevent merging (undo).



The terms "Project Information" and "Library Info" designate the same element.

Comparison: All subelements

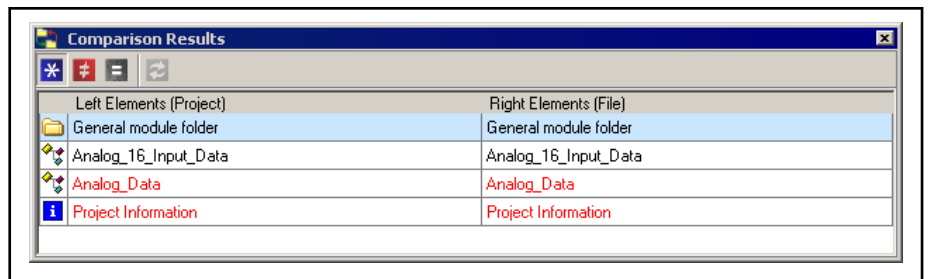


Fig. 4-3:

Comparison: "All subelements", two elements with differences (red) with filter option "All elements", "Different elements", "Identical elements"

4.2 Comparison: Logic

Here only the name is displayed in the window. "Merging" is not possible, since the element is fixed.

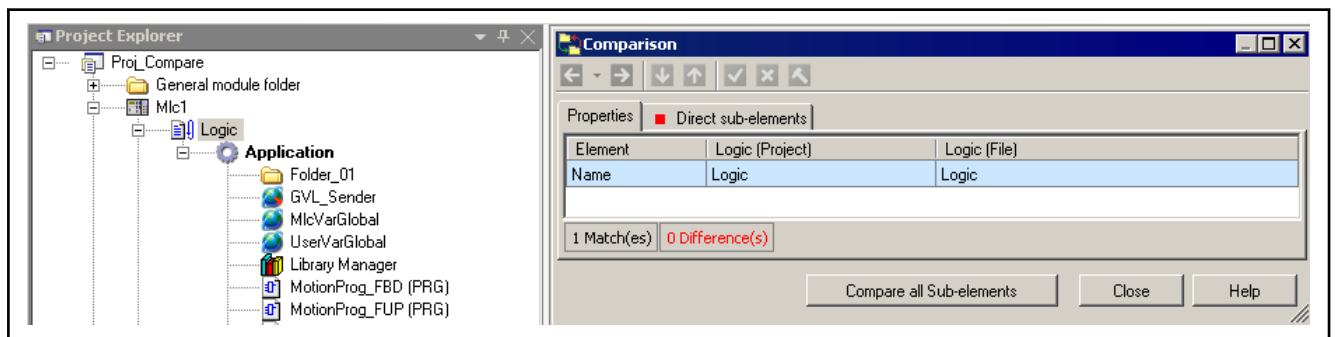


Fig. 4-4:

Comparison: Logic

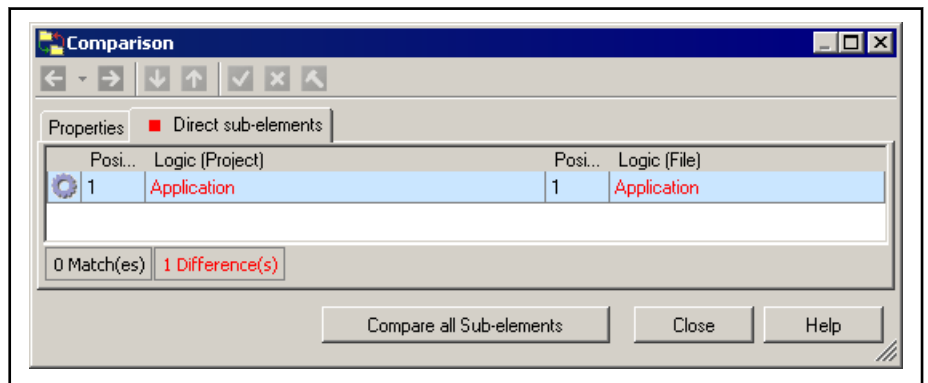


Fig. 4-5:

Comparison: Logic with one single subelement "Application"

Comparing PLC objects

Comparison: All subelements

Left Elements (Project)	Right Elements (File)
Logic	Logic
Application	Application
	DataServer_PRG
Folder_01	Folder_01
GVL_Sender	GVL_Sender
UserVarGlobal	UserVarGlobal
Library Manager	Library Manager
MotionProg_FBD	MotionProg_FBD
MotionProg_FUP	MotionProg_FUP
MotionPro	MotionPro
MotionProg	MotionProg
PlcProg	PlcProg
SymbolConfiguration	SymbolConfiguration
Task Configuration	Task Configuration
	DataServerTask
MotionTask	MotionTask
PlcTask	PlcTask
PersistentVars	PersistentVars

Fig. 4-6: Comparison: Logic with all subelements with filter option "All elements", "Different elements", "Identical elements"

4.3 Comparison: Application

Here, the name is displayed, the options for the boot project handling and the other application properties. These are located under Properties in the context menu of the "Application" node ([chapter 18.1.2 "Application" object](#) on [page 480](#)).

Element	Application (Project)	Application (File)
Boot application implied during download	False	True
Boot application implied during online change	False	False
Boot application prompt upon closing the project	True	True

2 Match(es) 1 Difference(s)

Compare all Sub-elements Close Help

Fig. 4-7: Comparison: Application

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

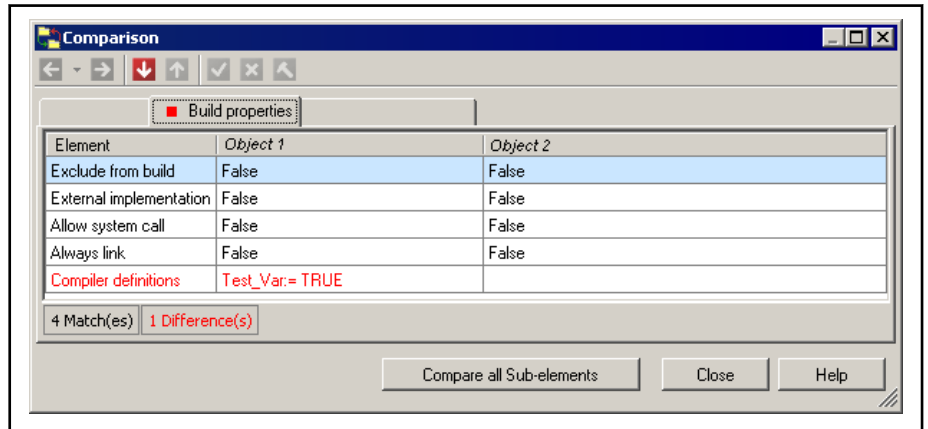


Fig. 4-8: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The "DataServer_PRG" program (right position 7) is missing in the project and is to be taken from the file.

"GVL_Sender" differs with regard to the content and the order of the objects is different from position 7.

Comparing PLC objects

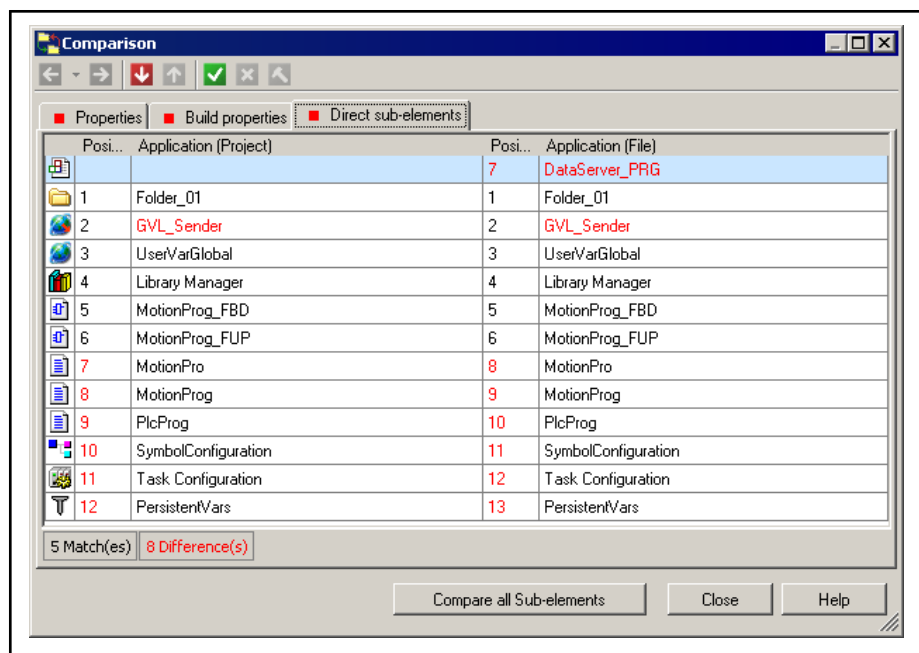


Fig. 4-9: Comparison: "Direct subelements"

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

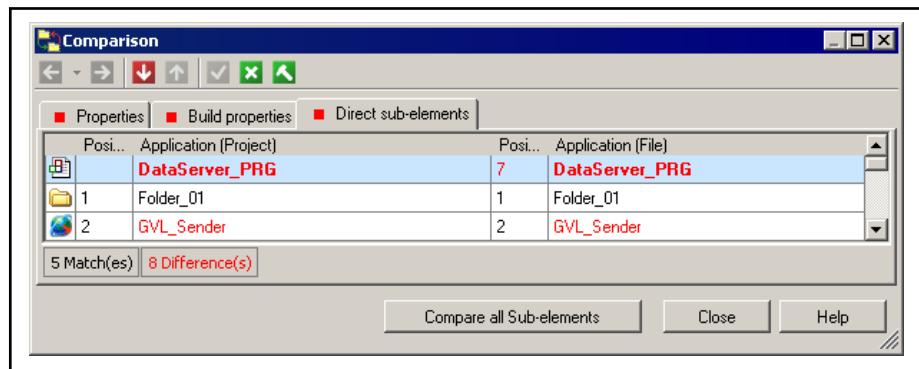


Fig. 4-10: Comparison: File is accepted into the project with "Start merge operation"

Comparison: All subelements

Left Elements (Project)	Right Elements (File)
Application	Application
	DataServer_PRG
Folder_01	Folder_01
GVL_Sender	GVL_Sender
UserVarGlobal	UserVarGlobal
Library Manager	Library Manager
MotionProg_FBD	MotionProg_FBD
MotionProg_FUP	MotionProg_FUP
MotionPro	MotionPro
MotionProg	MotionProg
PlcProg	PlcProg
SymbolConfiguration	SymbolConfiguration
Task Configuration	Task Configuration
	DataserverTask
MotionTask	MotionTask
PlcTask	PlcTask
PersistentVars	PersistentVars

Fig. 4-11: Comparison: Application with all subelements with filter option "All elements", "Different elements", "Identical elements"

In addition to the direct subelements, their subelements are displayed, as well. Here, the "sub-subelements" of the task configuration (DataServerTask, only right, MotionTask and PlcTask).

4.4 Comparison: Task configuration

Only the name "Task configuration" is displayed in the window ([chapter 18.1.25 "Task configuration" object](#) on page 562). "Merging" is not possible, since the element is fixed.

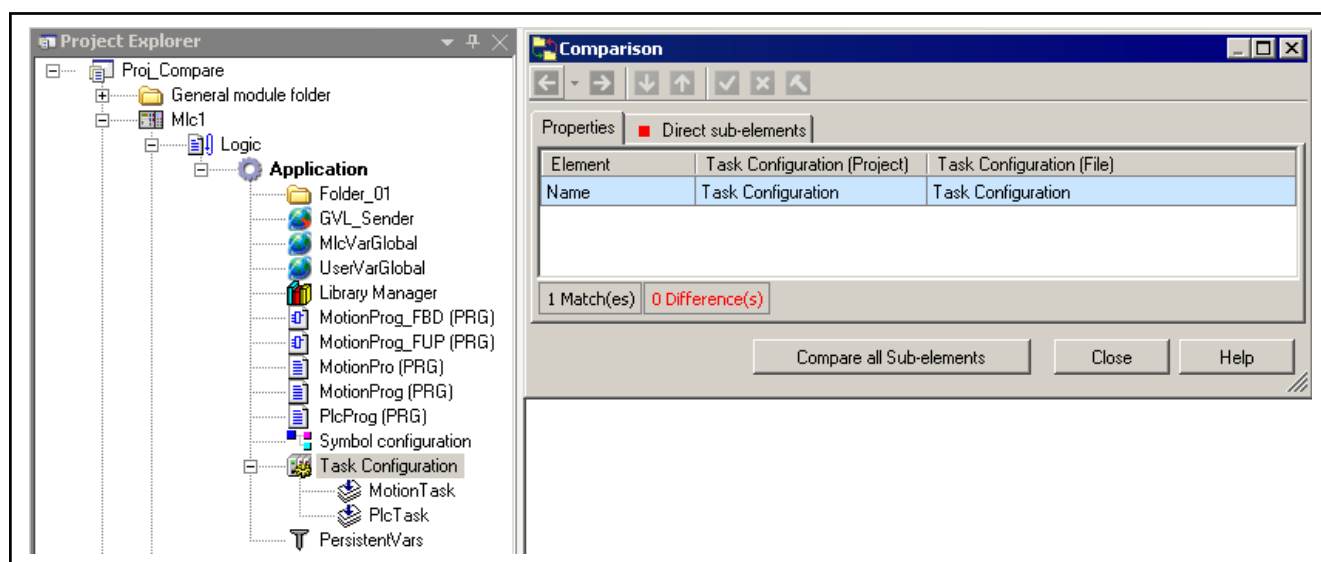


Fig. 4-12: Comparison: Task configuration

Comparing PLC objects

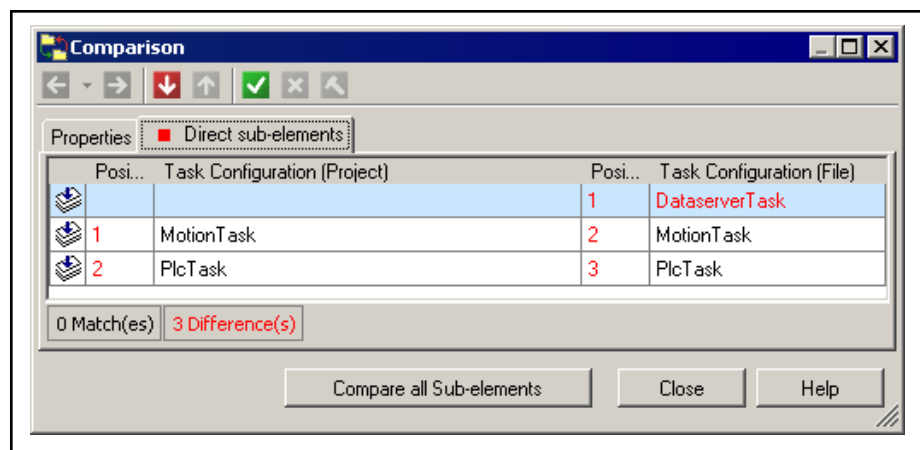


Fig. 4-13: Comparison: Task configuration, tasks as "Direct subelements"

The "DataserverTask" task is only Project Explorer the file storage and can be accepted into the project.

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer for merging.

Select the cross to prevent merging (undo).

Comparison: All subelements

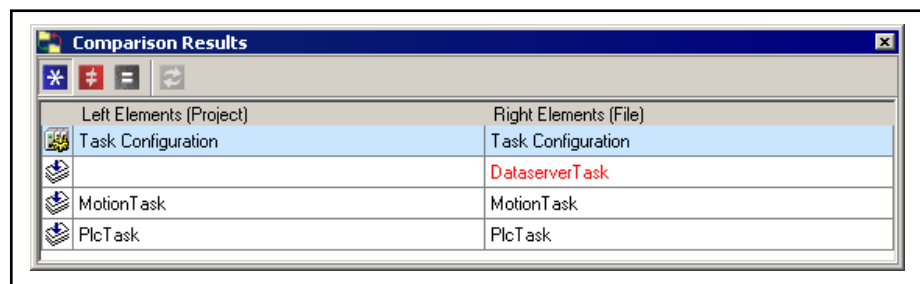


Fig. 4-14: Comparison: Application with all subelements with filter option "All elements", "Different elements", "Identical elements"

4.5 Comparison: Task

Task objects are compared in a table. Comparison between the "PlcTask" and the "MotionTask" ([chapter 18.1.26 "Task" object](#) on page 565).



Merging is not recommended...

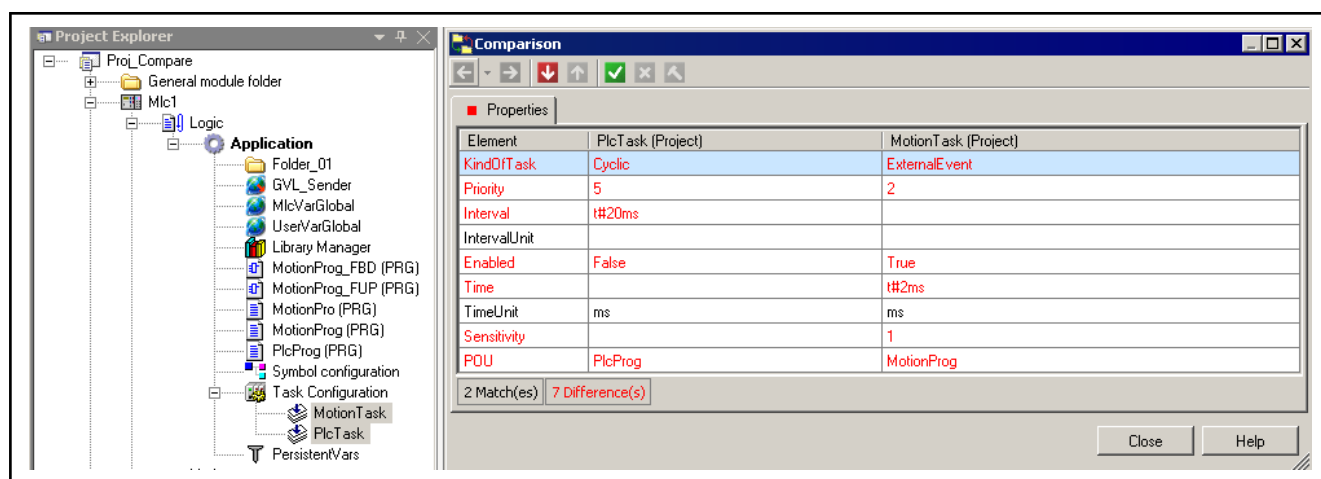


Fig. 4-15: Comparison: Task, here comparison between the "PlcTask" and "MotionTask" tasks

4.6 Comparison: Folder (Differences with Respect to the IndraWorks Folder)

Only the name is displayed. It can be renamed if the folder in the file (right) has another name.

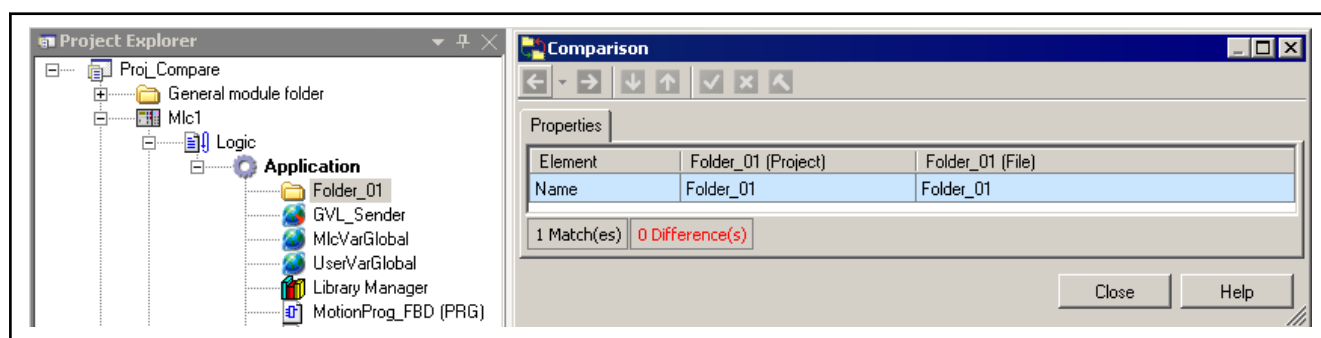


Fig. 4-16: Comparison: Folder

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.7 Comparison: Data Types

The comparison applies to all data types that can be created with the DUT editor as independent objects.

Among others, the following are supported: structures (STRUCT), arrays (ARRAY), unions (UNION), enumeration data types (ENUM) and partial range data types (SUBRANGE).

When data types are compared, the content is displayed as text and can be modified.

Comparing PLC objects

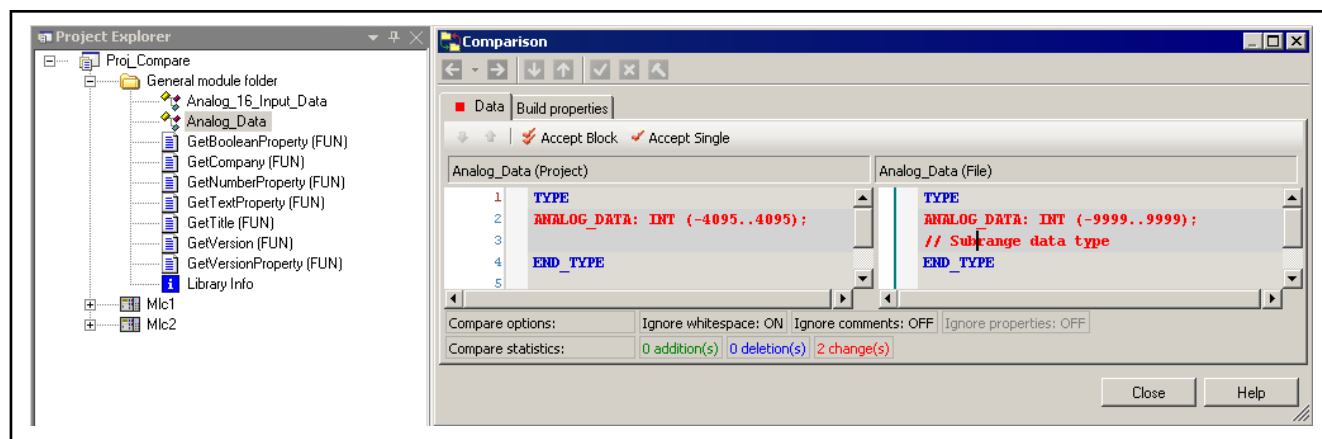


Fig. 4-17: Comparison: Data types

Also refer to

- [chapter 18.1.8 "Object "DUTon" on page 488](#)
- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)
- [chapter 17.5.14 "Data type 'ARRAY'" on page 363](#)
- [chapter 17.5.6 "'UNION' data type" on page 354](#)
- [chapter 17.5.17 "Enumerations" on page 374](#)
- [chapter 17.5.18 "Subrange types" on page 377](#)

The next/previous difference is selected with  or .



can be used to accept the complete highlighted text block.



can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties ([chapter "'Properties' - 'Build' dialog" on page 705](#)).

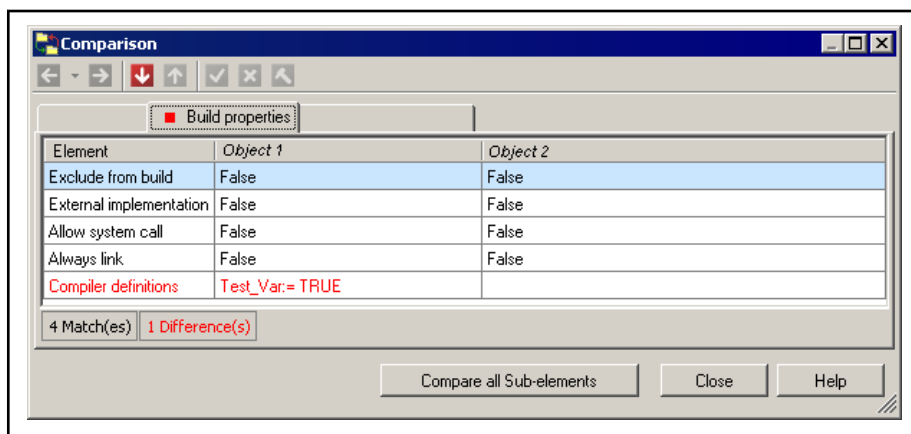


Fig. 4-18: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.8 Comparison: Global Variable List, Persistent Variables, Network Variables

The comparison applies to all Global Variable Lists ①, Global Variable Lists used as network variables on the sender end ② and persistent variable lists ③ that can be added as individual objects of an application (or in the general module folder).

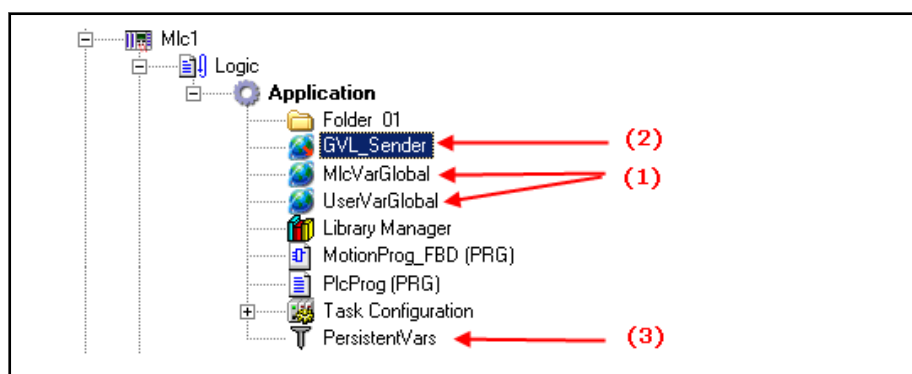


Fig. 4-19: Comparison: Possible global variable lists

Also refer to

- [chapter 18.1.11 "GVL" object - Global Variable List](#) on page 512
- [chapter 18.1.12 "Persistent variable list" object](#) on page 514

When data types are compared, the content is displayed as text and can be modified.

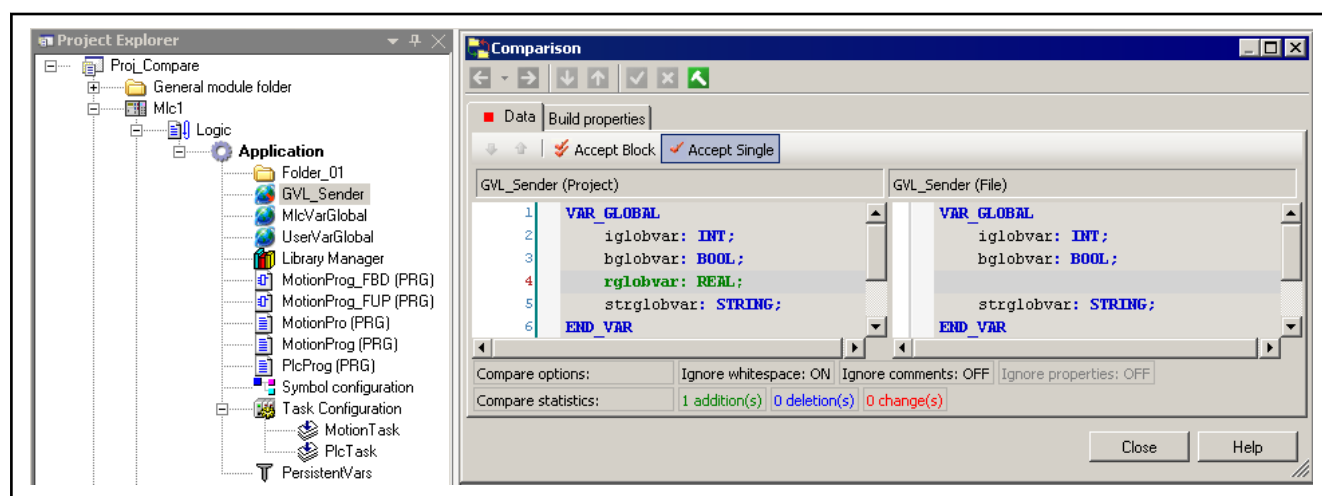
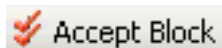


Fig. 4-20: Comparison: Global variable lists

The next/previous difference is selected with  or .



can be used to accept the complete highlighted text block.



can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

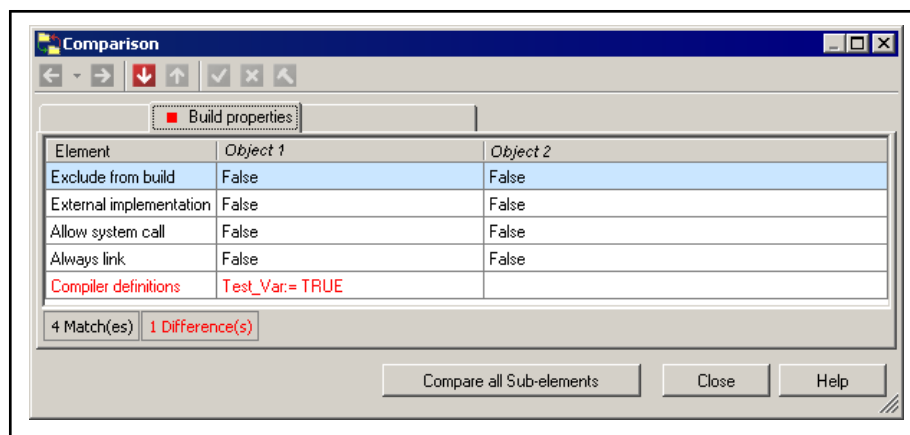


Fig. 4-21: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.9 Comparison: Network Variable Lists, Receiver End

The comparison applies to Global Variable Lists used as “network variables on the receiver end” created as individual objects of an application ([chapter 18.1.16 "Global Network Variable List \(receiver\) object" on page 519](#)).



These lists are created automatically and remain current within the project.

In this way, the comparison is only executed as a check for differences.

When comparing Network Variable Lists, the content display is XML-based. It cannot be modified (merged).

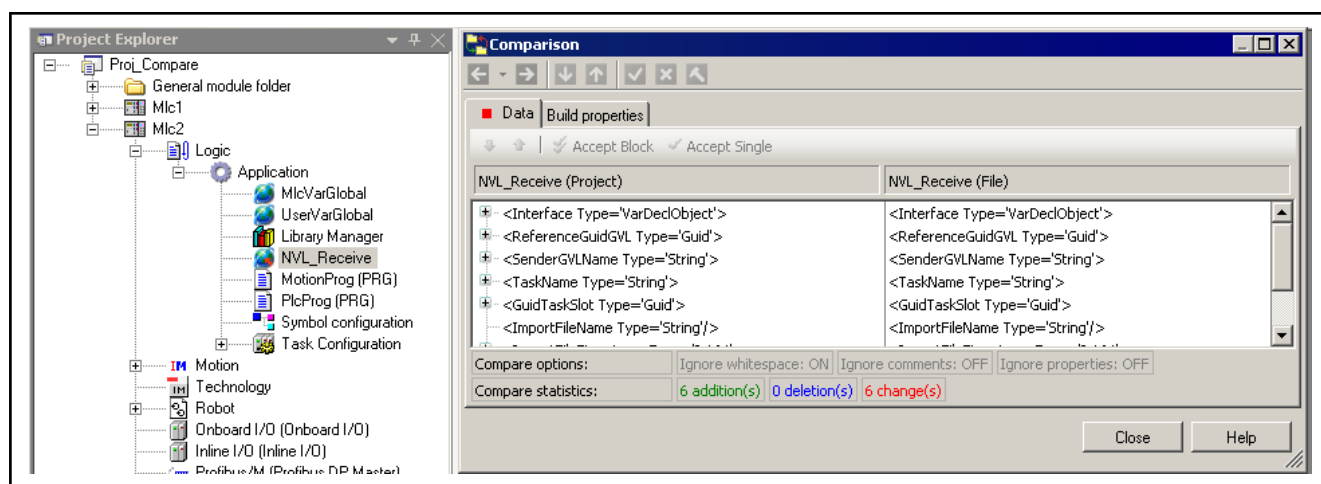


Fig. 4-22: Comparison: Network variable lists on the receiver end

The next/previous difference is selected with  or .

The next tab shows the build properties (chapter ""Properties" - "Build" dialog" on page 705).

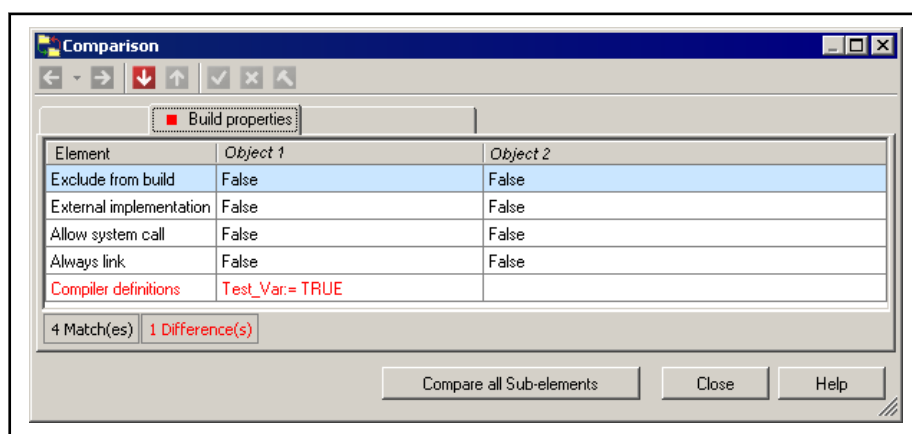


Fig. 4-23: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.10 Comparison: Programs, Function Blocks, Functions...

The comparison applies to the following objects:

- chapter ""Program" object (PROGRAM)" on page 522
- chapter ""Function block" object (FUNCTION_BLOCK)" on page 523
- chapter ""Function" object (FUNCTION)" on page 525
- chapter "Object "Actionon page 540

Comparing PLC objects

- chapter "SFC elements "steps" and "transition"" on page 245
- chapter ""Method" object " on page 532
- chapter "Object 'Property'" on page 537, PROPERTY with GET and SET

The objects named can be declared and implemented in the IEC programming language editors available in the system.

Comparison: ST

Comparisons of the declaration sections and objects implemented in Structured Text are text-based (chapter 17.1.3 "Structured Text and Extended Structured Text (ExST)" on page 225).

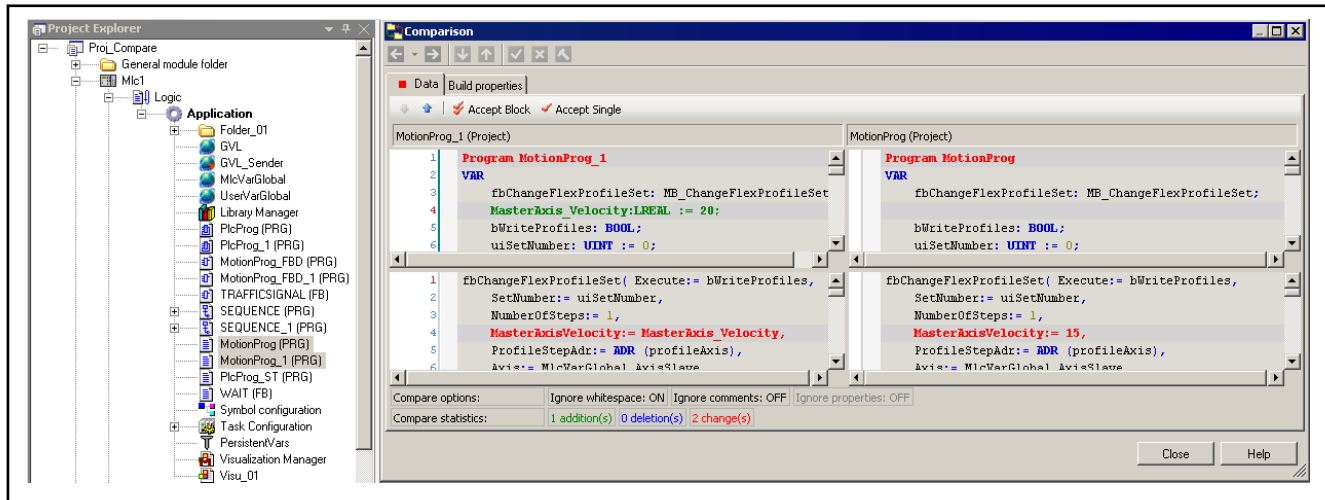


Fig. 4-24: Comparison: PROGRAM in ST, declaration and implementation

Merging is possible in the declaration and in the implementation.

The next/previous difference is selected with  or .



Accept Block

can be used to accept the complete highlighted text block.



Accept Single

can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties (chapter ""Properties" - "Build" dialog" on page 705).

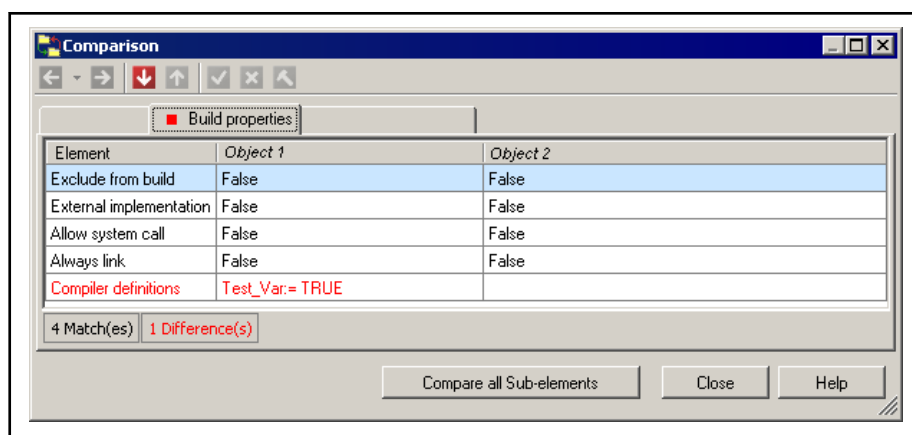


Fig. 4-25: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

Comparison: FBD

Objects that have been implemented as function block diagram are compared graphically.

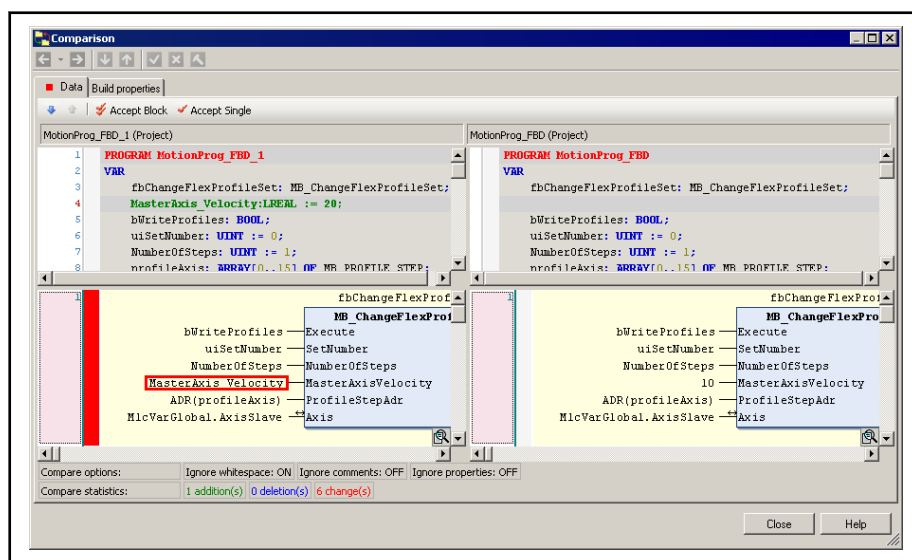


Fig. 4-26: Comparison: PROGRAM in FBD, declaration and implementation

Merging is possible in the declaration and in the implementation.

The next tab shows the build properties (chapter "Properties" - "Build" dialog on page 705).

Comparing PLC objects

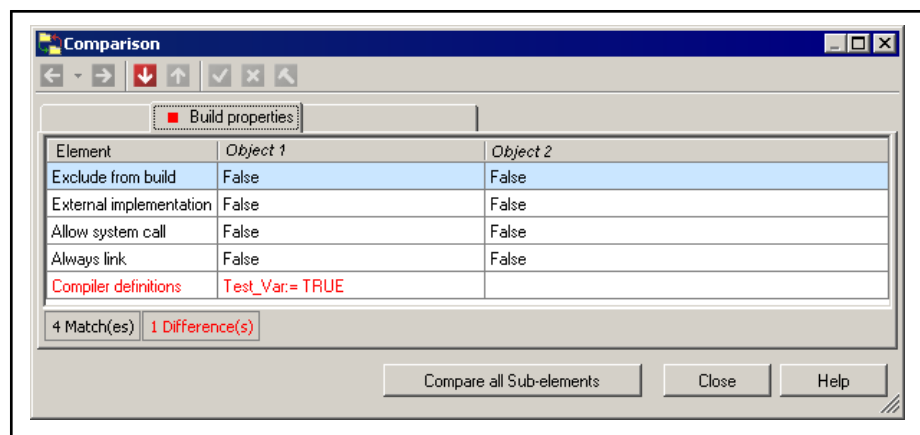


Fig. 4-27: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

Comparison: IL POU's implemented in the instruction list are compared as XML. Merging is not possible here.

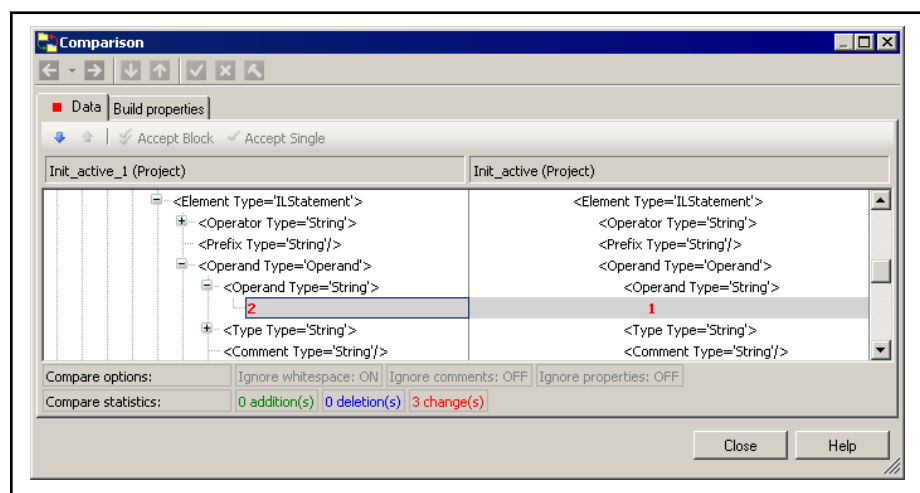


Fig. 4-28: Comparison: Action in instruction list, implementation

The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

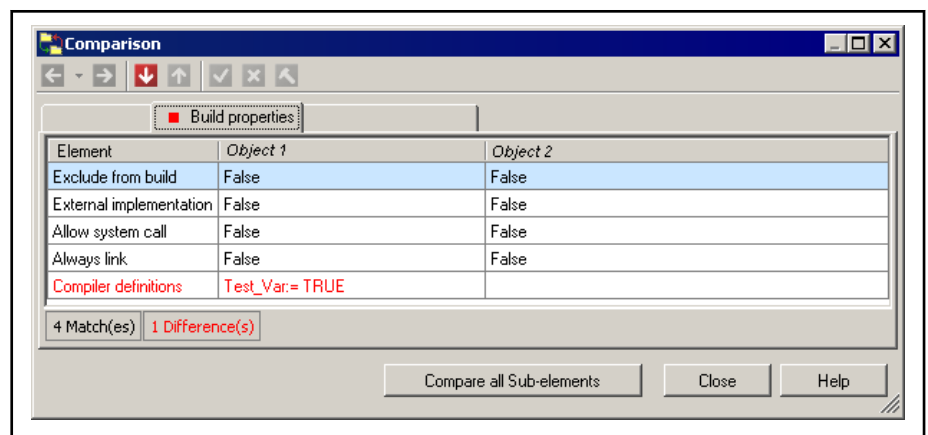


Fig. 4-29: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer for merging.

Select the cross to prevent merging (undo).

Comparison: SFC

Objects that have been structured by means of SFC (sequential function chart) are compared graphically. Merging is only possible in the declaration.

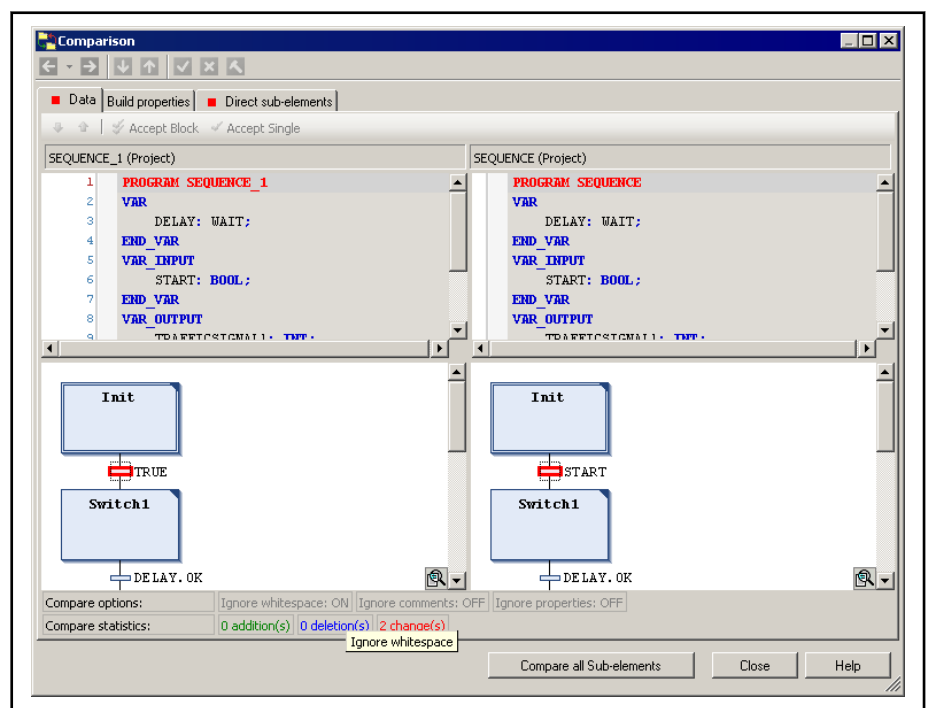


Fig. 4-30: Comparison: FUNCTION_BLOCK with SFC structure, declaration and implementation

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

Comparing PLC objects

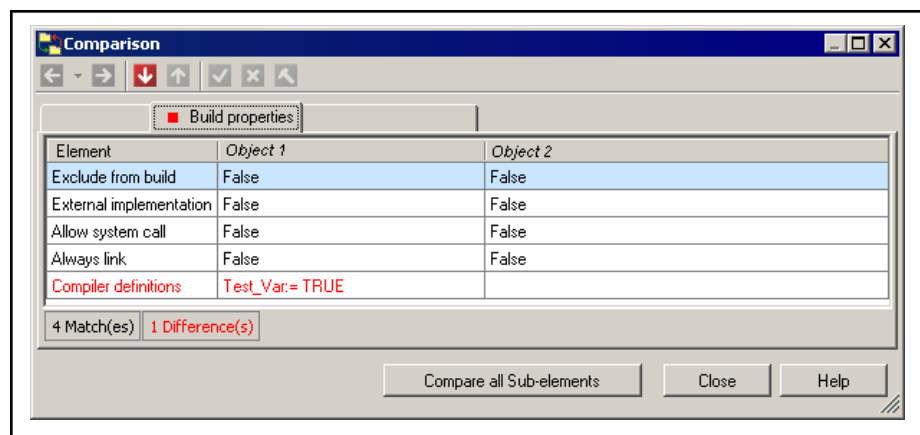


Fig. 4-31: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

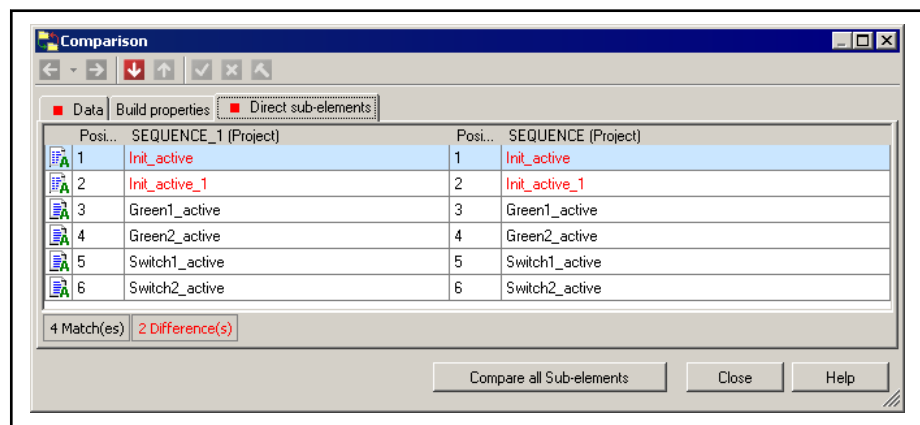


Fig. 4-32: Comparison: POU "Direct subelements"

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

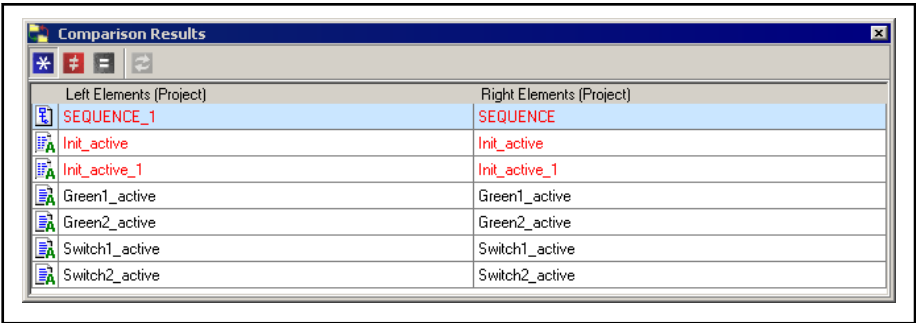


Fig. 4-33: Comparison: POU with all subelements with the filter option "All elements", "Different elements", "Identical elements"

Comparison: CFC POU implemented in the instruction list are compared as XML. Merging is not possible here.

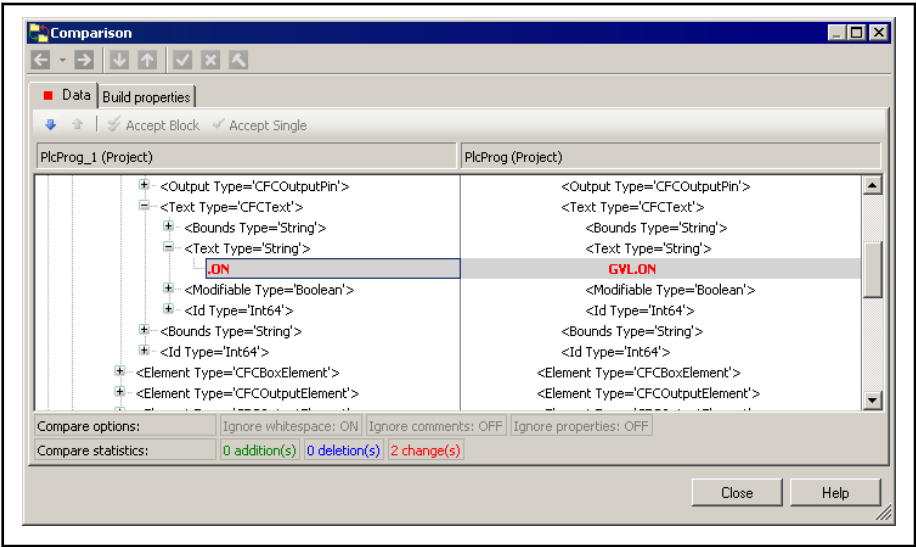


Fig. 4-34: Comparison: PROGRAM in CFC, declaration and implementation
The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

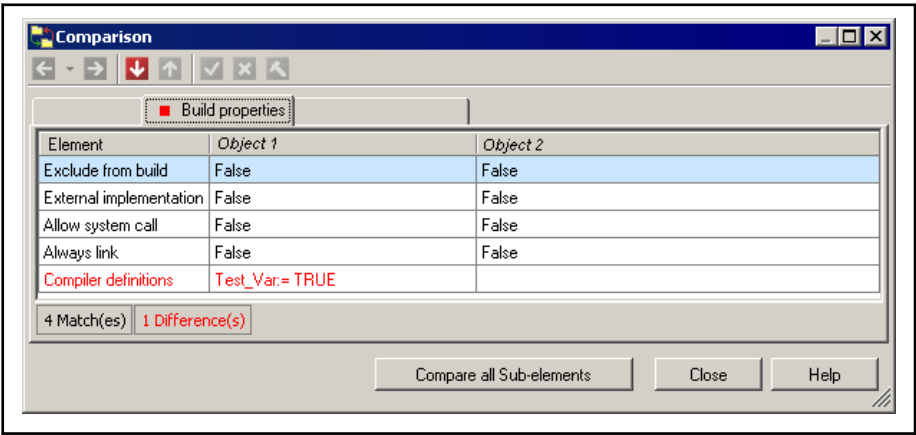


Fig. 4-35: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

Comparing PLC objects

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.11 Comparison: Interfaces, interface methods, interface properties

When comparing interfaces, interface methods or interface properties, the content display is XML-based ([chapter "Interface" object \(INTERFACE\)" on page 529](#)).

Modification (merging) is not possible.

The next/previous difference is selected with  or .

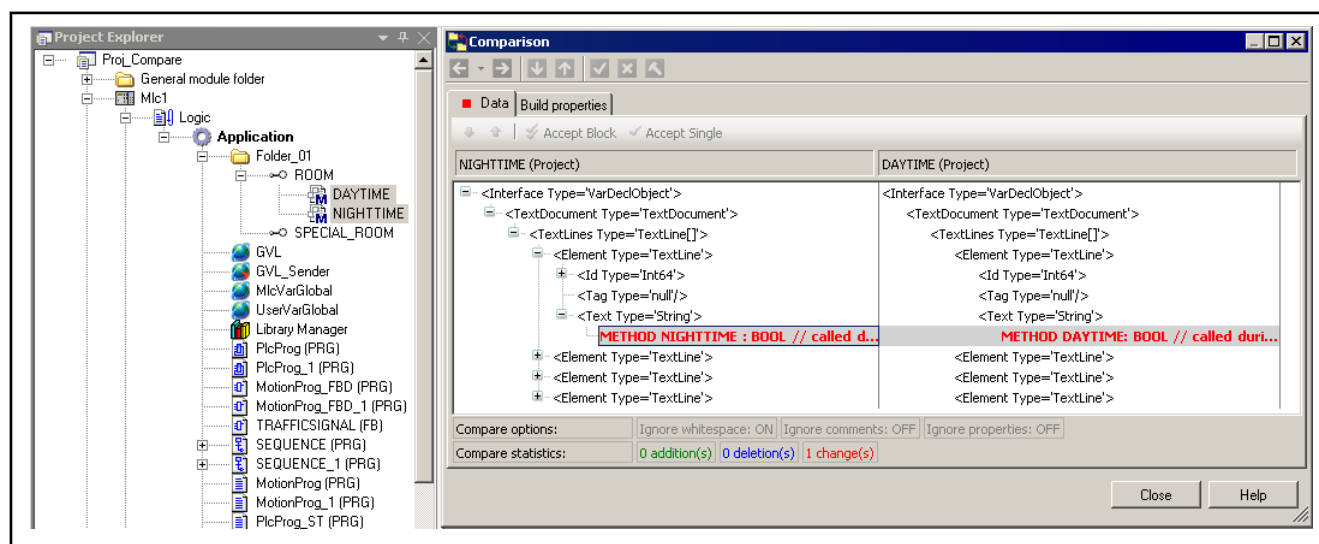


Fig. 4-36: Comparison: Interfaces, interface methods, interface properties

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

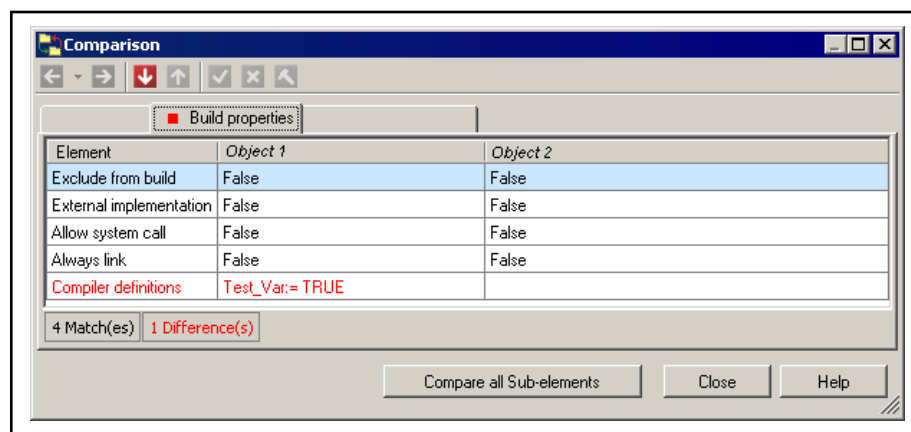


Fig. 4-37: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.12 Comparison: Library manager

When comparing library managers, the content is listed in tabular form. Acceptance is possible line-by-line.

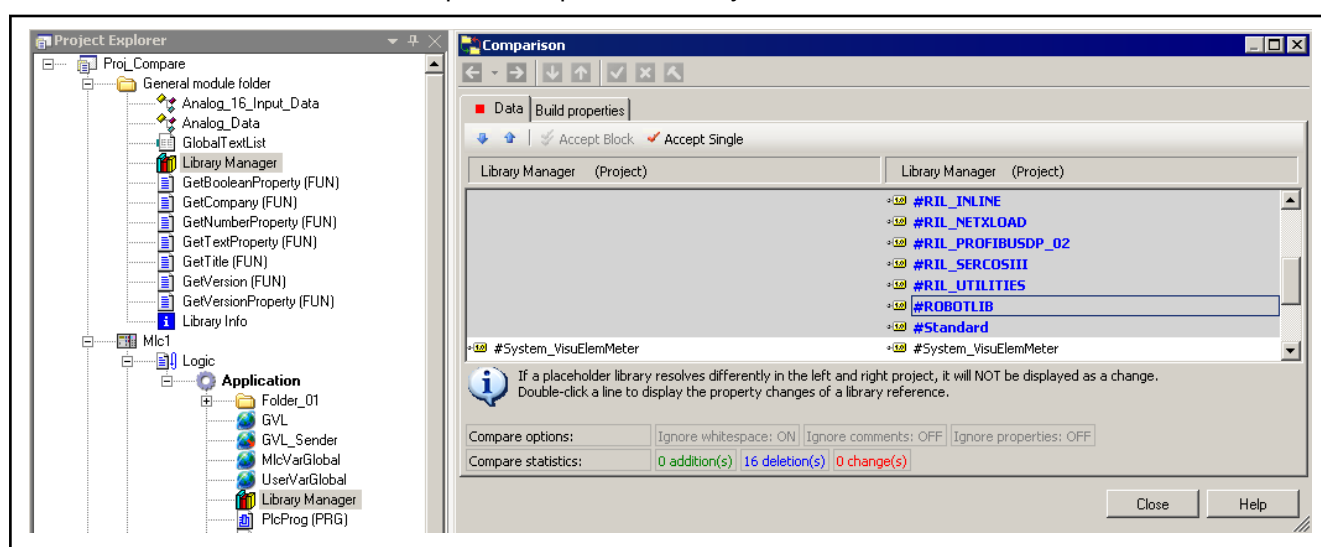
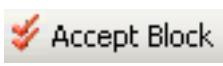


Fig. 4-38: Comparison: Library manager, here ML_Robot added into the project

The next/previous difference is selected with  or .

 Accept Block can be used to accept the complete highlighted text block.

 Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

Comparing PLC objects

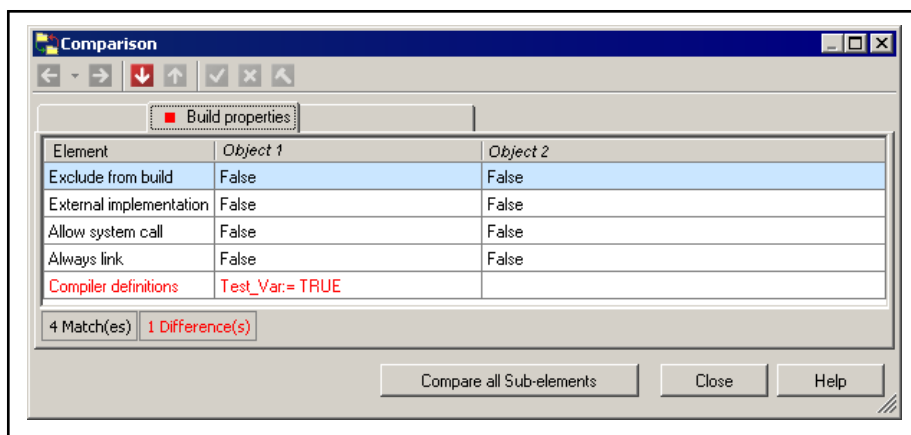


Fig. 4-39: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer for merging.

Select the cross to prevent merging (undo).

4.13 Comparison: Library Info

When comparing "library info", the content display is XML-based. It cannot be modified (merged).

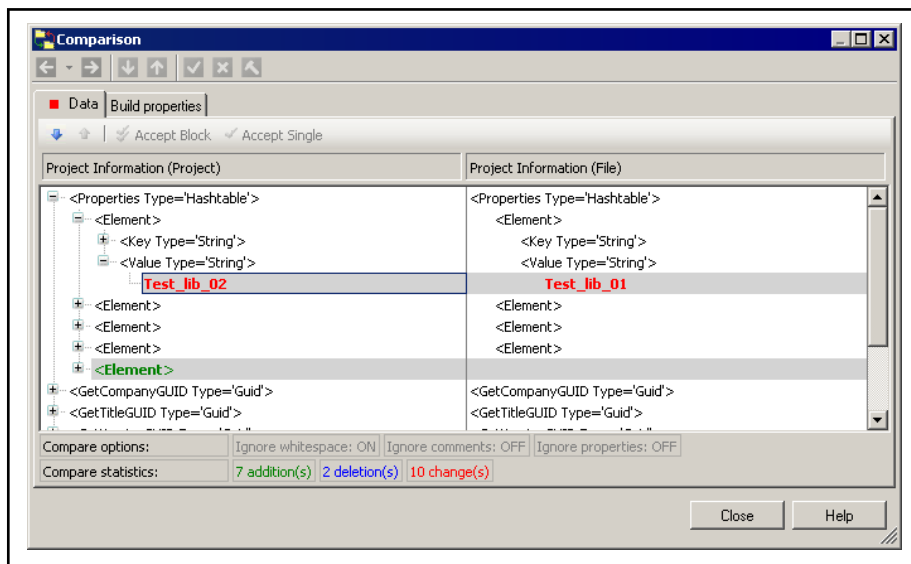


Fig. 4-40: Comparison: Library Info; library enabled with version number 10.06.0.1



The terms "Project Information" and "Library Info" designate the same element.

The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

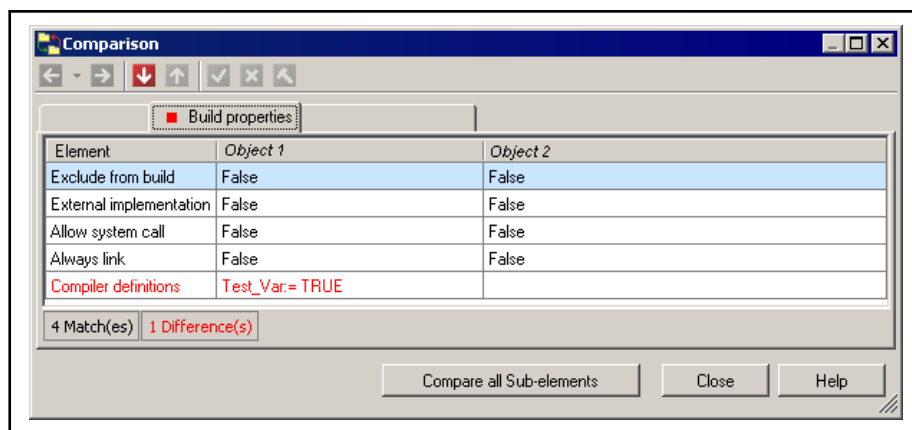


Fig. 4-41: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.14 Comparison: Data Server

When comparing data servers, the content display is XML-based. It cannot be modified (merged).

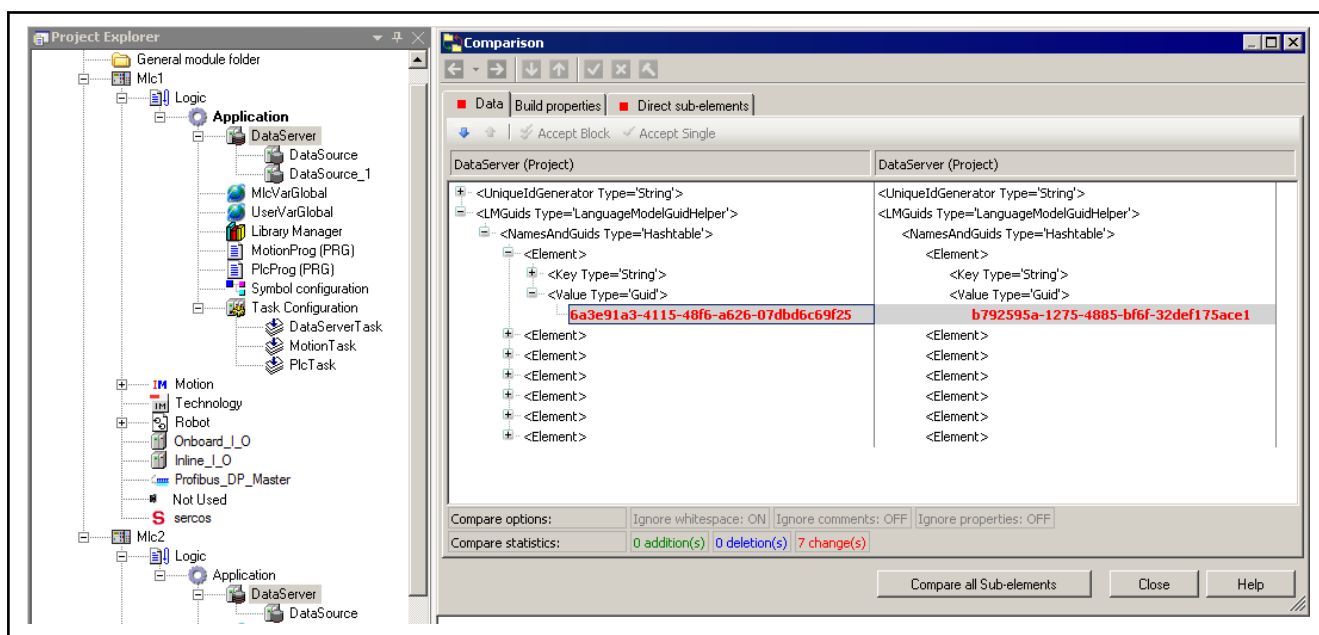


Fig. 4-42: Comparison: Data server

The next/previous difference is selected with  or .

Comparing PLC objects

The next tab shows the build properties (chapter ""Properties" - "Build" dialog" on page 705).

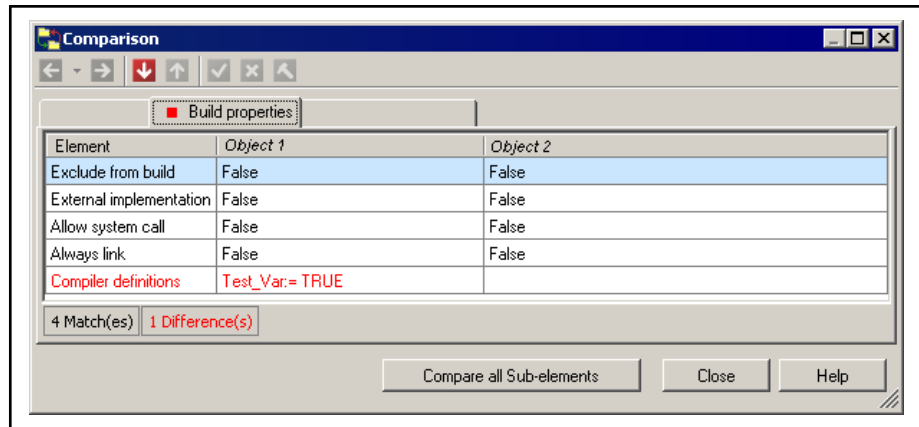


Fig. 4-43: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

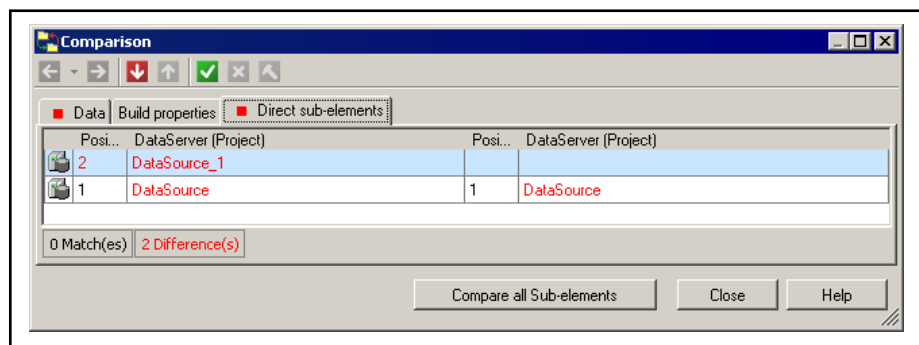


Fig. 4-44: Comparison: Data server, Direct subelements

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

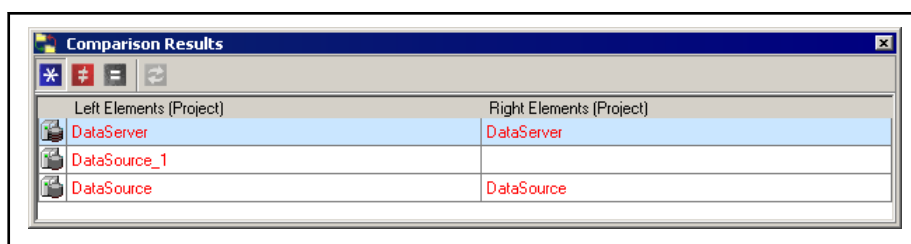


Fig. 4-45: Comparison results: Data server

4.15 Comparison: Data source (DataServer)

When comparing data sources, the content display is XML-based. It cannot be modified (merged).

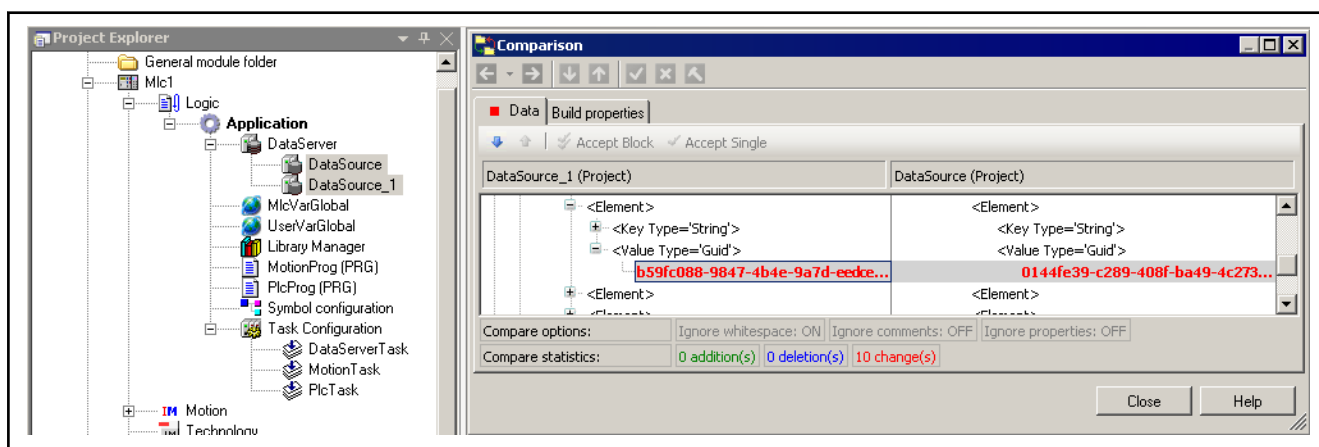


Fig. 4-46: Comparison: DataSources

The next/previous difference is selected with or .

The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

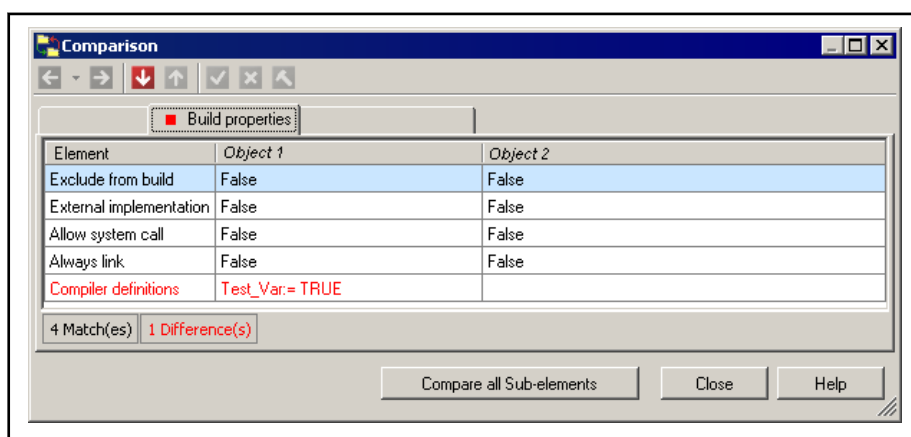


Fig. 4-47: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.16 Comparison: Symbol configuration

When comparing symbol configurations, the content display is XML-based. It cannot be modified (merged).

4.17 Comparison: Visualization manager

When comparing visualization managers, the content display is XML-based. It cannot be modified (merged).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

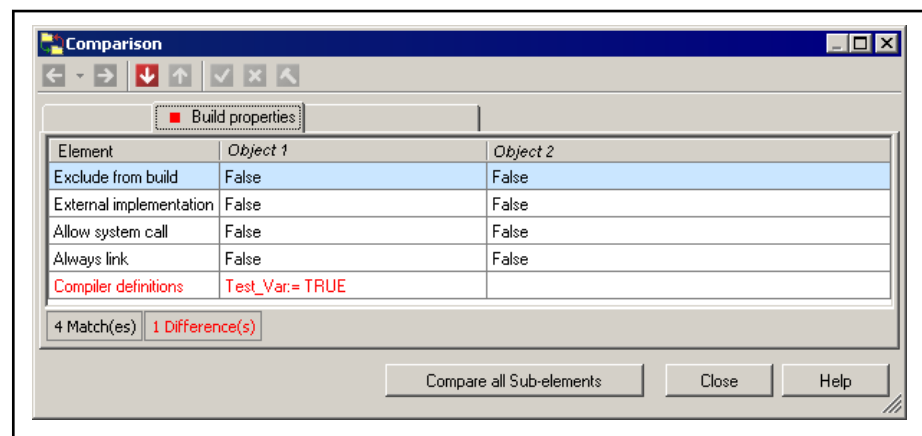


Fig. 4-48: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.18 Comparison: Visualization

When comparing visualizations, the content display is XML-based. It cannot be modified (merged).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

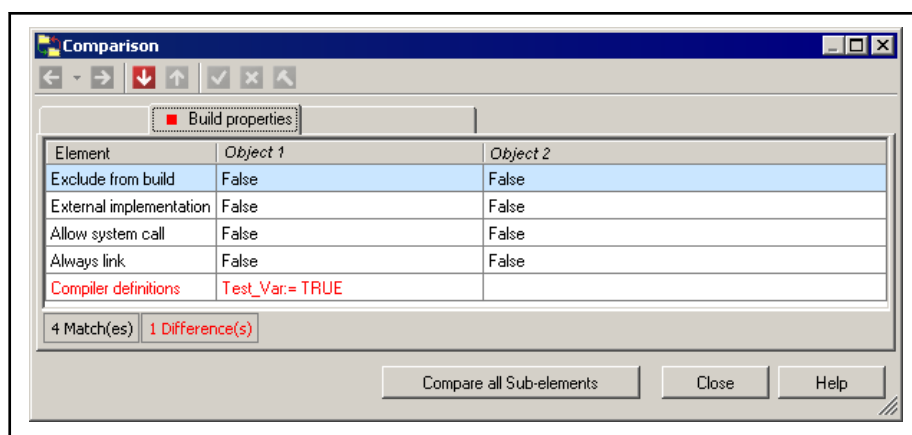


Fig. 4-49: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.19 Comparison: Text List

When comparing text lists, the content display is XML-based. It cannot be modified (merged).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

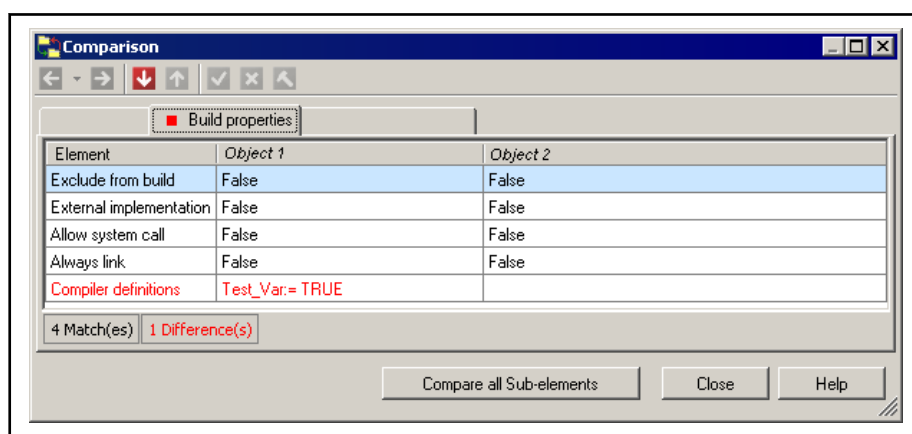


Fig. 4-50: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.20 Comparison: Image Pool

When comparing image pools, the content display is XML-based. It cannot be modified (merged).

4.21 Comparison: Recipe Manager

When comparing recipe managers, the content display is XML-based. It cannot be modified (merged).

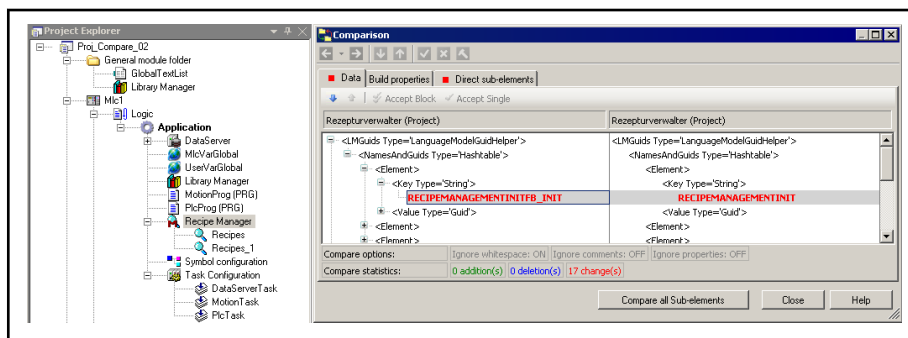


Fig. 4-51: Comparison: Recipe Manager

The next/previous difference is selected with  or .

The next tab shows the build properties (chapter "Properties" - "Build" dialog on page 705).

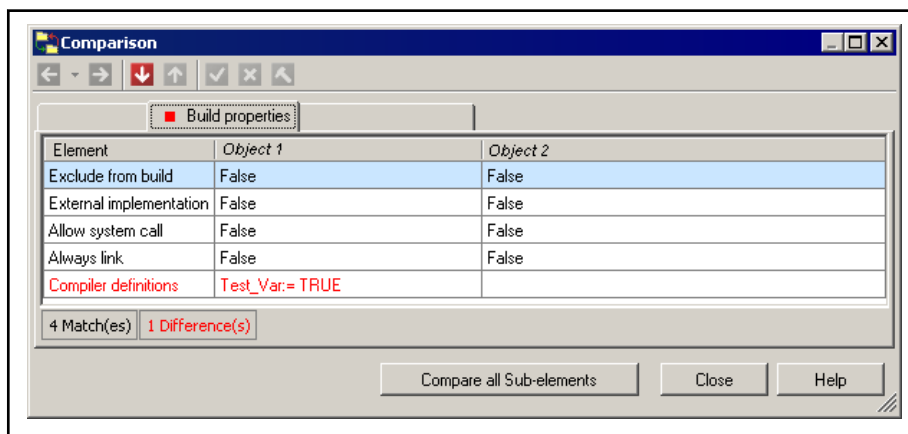


Fig. 4-52: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

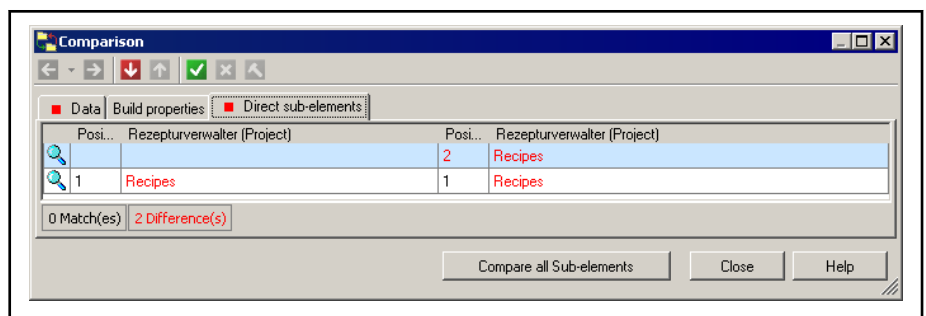


Fig. 4-53: Comparison: Recipe manager, Direct subelements

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

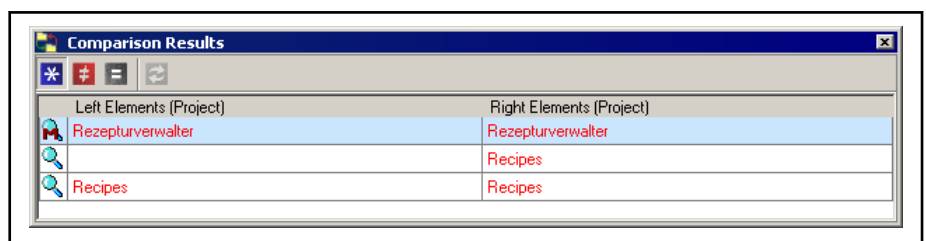


Fig. 4-54: Comparison results: Recipe Manager

4.22 Comparison: Recipe Definition

When comparing recipe managers, the content display is XML-based. It cannot be modified (merged).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog" on page 705](#)).

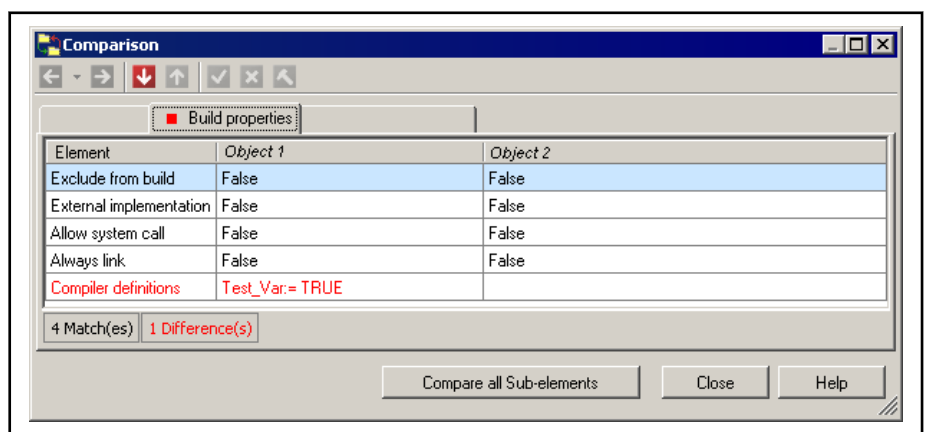


Fig. 4-55: Comparison: Build properties object 1/object 2. One difference identified and marked in red

Comparing PLC objects

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

4.23 Comparison: Devices (onboard I/O, Inline I/O, field buses, ...)

The comparison applies to the devices associated with the respective control (onboard I/O, Inline I/O, field buses). Since these differ from control type to control type, the following comparison results have to be seen as examples.

In principle, it is tried to provide a familiar environment at comparison with regard to the input and to enable merging based on the comparison.

Onboard I/O:

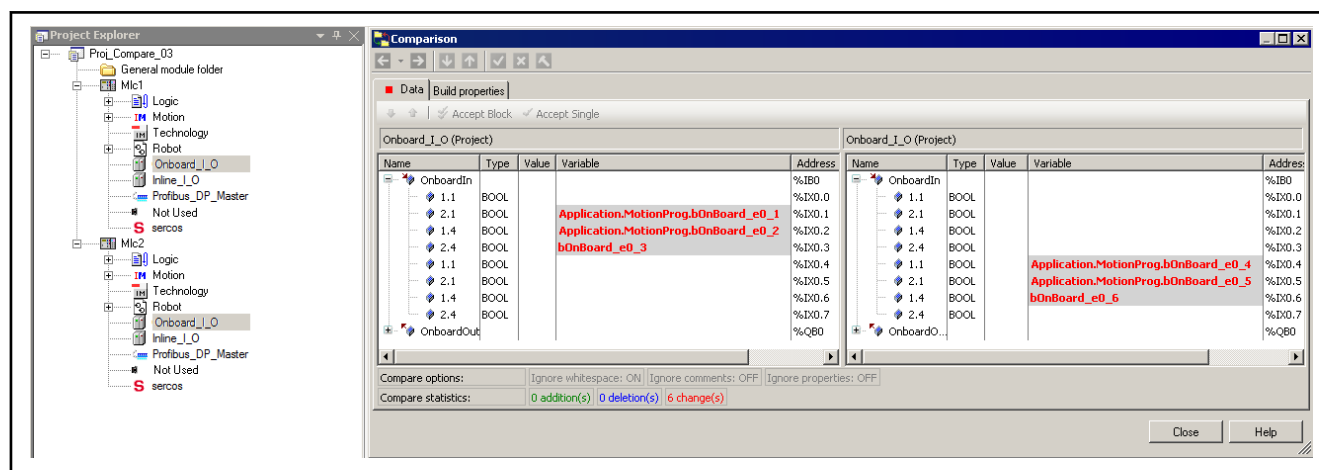


Fig. 4-56: Comparison: Onboard I/O:

The next/previous difference is selected with  or .

 **Accept Block** can be used to accept the complete highlighted text block.

 **Accept Single** can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

The next tab shows the build properties ([chapter "Properties" - "Build" dialog on page 705](#)).

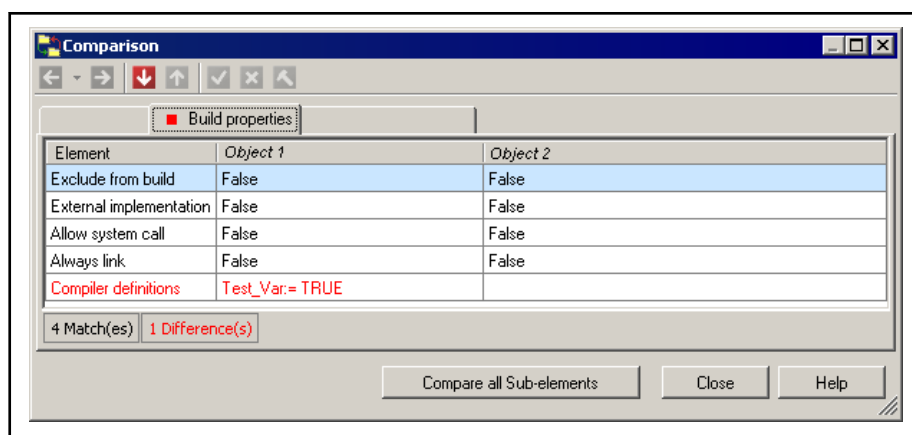


Fig. 4-57: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

If the checkmark is displayed, press it to prepare for merging:

Select the hammer for merging.

Select the cross to prevent merging (undo).

Inline I/O objects: The comparison shows the Inline cycle counters and diagnostic data tabs of the Inline I/O objects. Merging is not intended.

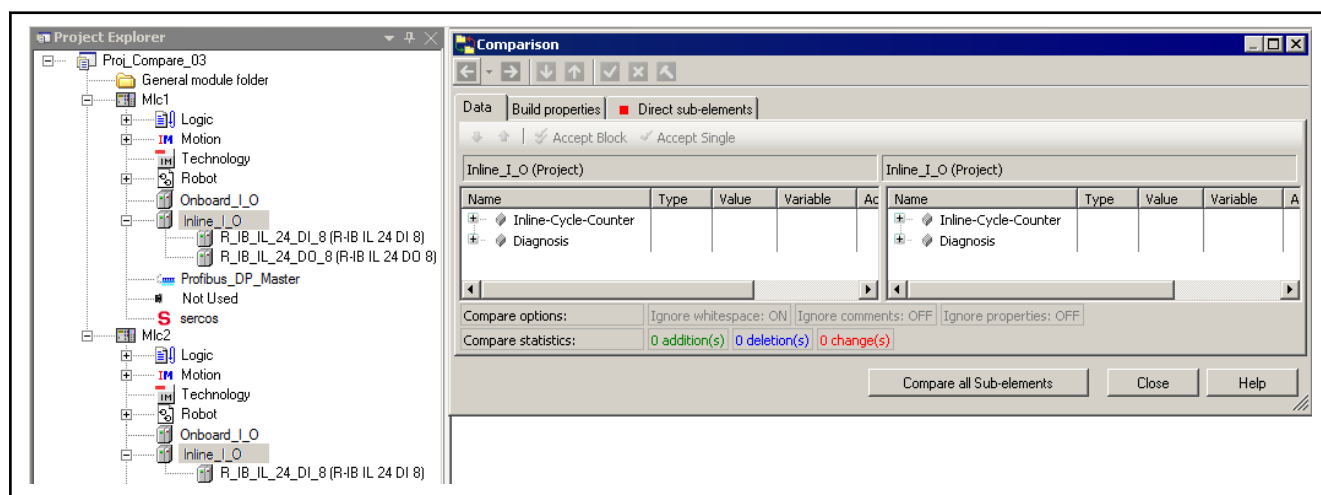


Fig. 4-58: Comparison: Inline I/O: Inline cycle counters and diagnostic data tabs

The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

Comparing PLC objects

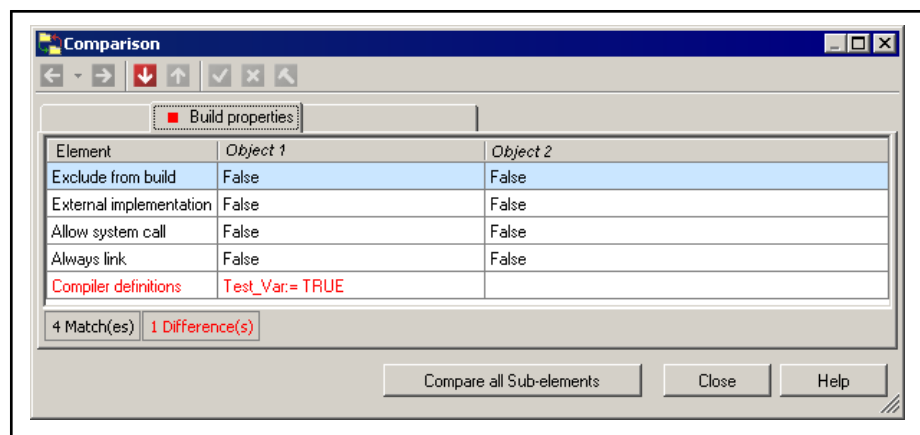


Fig. 4-59: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

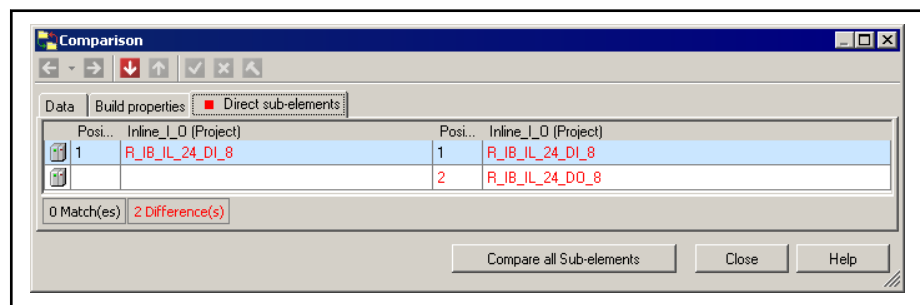


Fig. 4-60: Comparison: Inline I/O: Direct subelements

If the assignment in the columns of the two objects differs, use  to select the next difference ( for the previous difference).

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

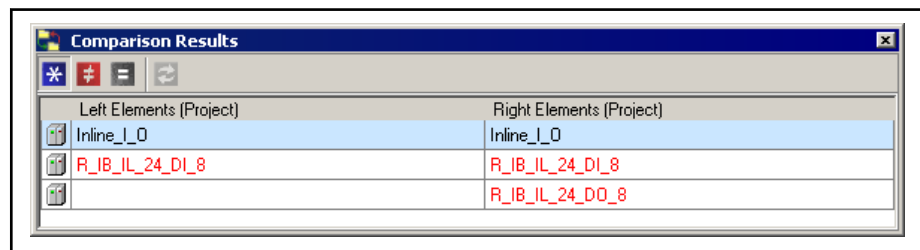


Fig. 4-61: Comparison results: Inline I/O: All subelements

Inline I/O modules

The same names but different assignments are indicated if the names of the Inline I/O modules under "Direct subelements" and "Comparison results" are highlighted in red:

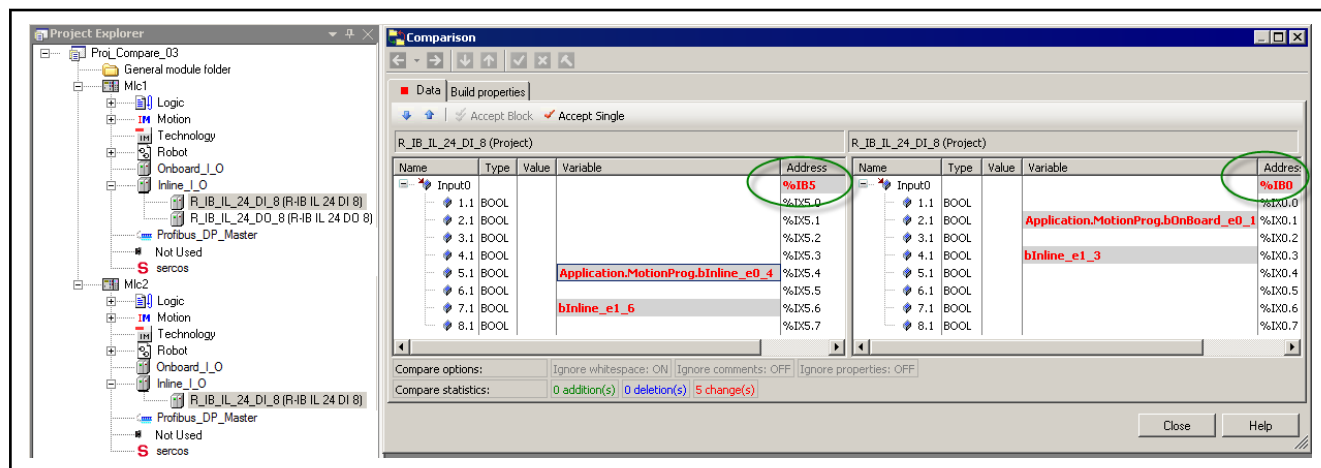
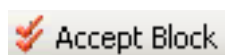


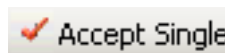
Fig. 4-62: Comparison: Inline I/O modules

The different addresses with identical position of the modules in the Project Explorer indicate an address shift.

The next/previous difference is selected with or .



can be used to accept the complete highlighted text block.



can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer for merging.

Select the cross to prevent merging (undo).

The next tab shows the build properties (chapter "Properties" - "Build" dialog" on page 705).

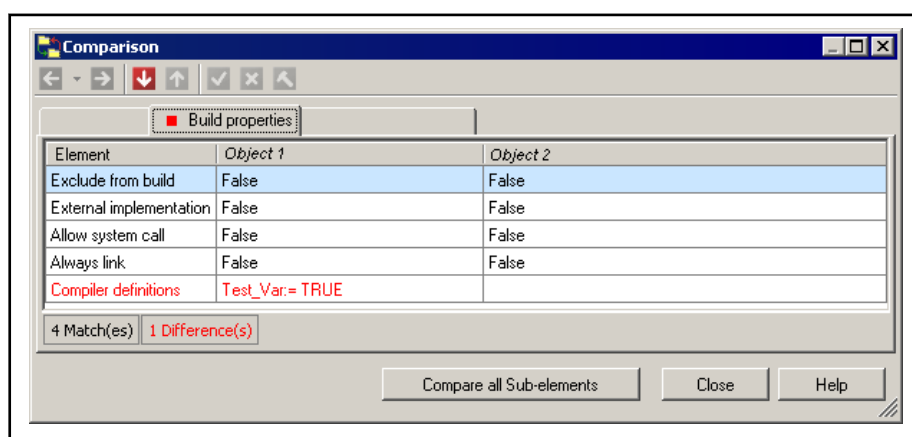


Fig. 4-63: Comparison: Build properties object 1/object 2. One difference identified and marked in red

If the assignment in the columns of the two objects differs, use to select the next difference (for the previous difference).

Comparing PLC objects

If the checkmark  is displayed, press it to prepare for merging:

Select the hammer  for merging.

Select the cross  to prevent merging (undo).

5 Comparing PLC objects online

To support the comparison of PLC objects between the Engineering PC and the control, the corresponding data has to be stored at a defined point in time. For the structural comparisons, checksums are transferred for the respective objects to quickly generate and transfer them. For the detailed comparison, zipped export files are transferred to the control. Creating and generating "big" projects might take some time. These actions are started after the actual download of the application and executed parallelly. Thus, there are only minimal effects on the operation of online change, download and monitoring.

To transfer data for the "overview on the comparison" of the PLC objects, the option "Enable online comparison" has to be selected ([Options on page 54](#)).

If the option is disabled, no data is transferred to the control. If the online comparison starts with this setting, the PLC objects are shown as "non-comparable". This option is enabled by default.

To additionally transfer the sources for the "Detailed comparison", select the "Transfer source files" option. This option is disabled by default.

If the options are set correspondingly, data is generated for the online comparison after transferring the changed code to the control (via **Debug ► Login**) and then transferred to the control. If the boot project is up-to-date (e.g. after an online change with boot project download), the sources are also transferred. If the boot project is not up-to-date (e.g. after an online change without boot project download), select **Debug ► Generate boot application** to start the transfer of the sources to the control.

WARNING

Data protection on the control

The transferred data is currently only protected by a password. With the respective IndraWorks version, everyone can access the source data using the online comparison functionality.

The users, who try to impede this, have to disable the source transfer individually for each control in the "Properties" dialog of the control in the "PLC" tab by selecting **Transferring PLC sources together with boot application**

If the online comparison is started or updated, data stored on the control is transferred to the engineering PC and analyzed for the "overview on the comparison" or for the "detailed comparison".

5.1 Overview on the comparison

The PLC objects are allocated by their names.

Status equal: Two objects obtain the status "Equal" if the checksum created on the control for this object matches the checksum determined in the project.

Status unequal: If the checksums of the objects are different, it is checked whether the change causes an online change or a download. In this case, "unequal" is displayed.

Status nearly equal: If only a comment was changed for example or a property was modified, the status "Nearly equal" is displayed.

For additional information, go to the tooltip:

Comparing PLC objects online

1. A warning is displayed at the "Logic" node if the data for the online comparison was not at all or not completely transferred to the control.
2. A warning is displayed at the "Application" node if the boot project is not up-to-date (no detailed comparison, as only the sources for the boot project are transferred to the control).
3. The comparison tooltip at the "Application" object displays whether the application can be logged in via online change.
4. If only a comment has changed in a POU for example, the tooltip displays that this change does not affect the compilation.
5. If both objects result from different projects (Project1 with POU "Name1" was loaded to the control and Project2 is open and also contains a POU with the name "Name1"), the objects are also displayed as different objects if their content is identical. This is the case, as the checksums of the objects differ due to their different guides. The tooltip displays that the objects have different IDs.
6. If an object is only available on one site, the status "unequal" is displayed.
7. Safety objects contain the status "cannot be compared"

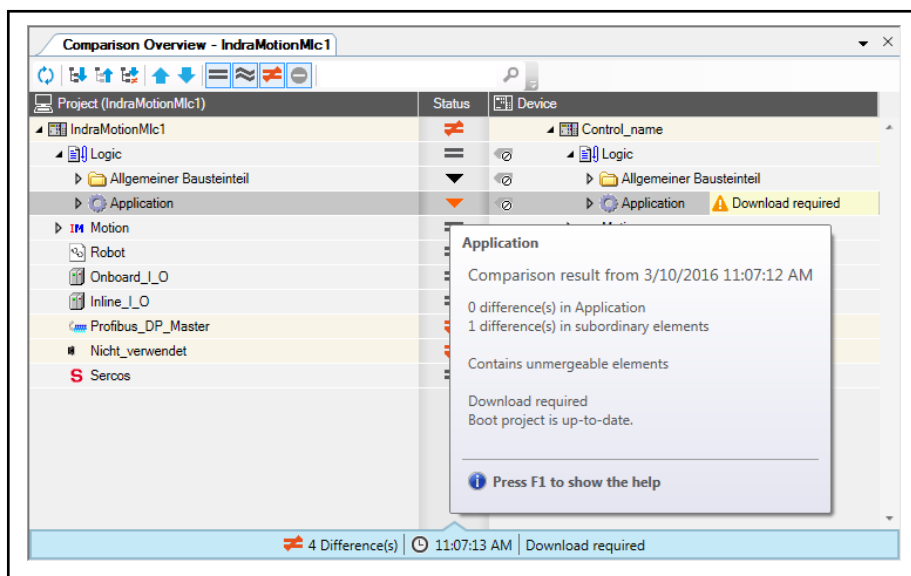


Fig. 5-1: Example on a "comparison overview"

5.2 Detailed comparison

Double-click on a line of the comparison overview to open the "Detailed comparison" or to update it if a detailed comparison is already open. The comparison overview is displayed in the lower section of the window.

If there are no valid sources on the control, this information is displayed in the open window:

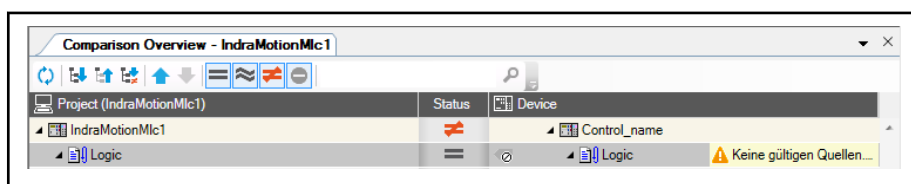


Fig. 5-2: File comparison without valid sources on the control

This can be the case if the corresponding option was not enabled, the sources were deleted on the control or the boot project is not up-to-date (visualized as warning at the "Application" node).

If the source data is on the control, a detailed comparison is executed (as for the export file comparison).

If the objects to be compared originate from different projects, i.e. the project on the control was loaded with a project currently closed in the engineering PC, the objects are shown as unequal in the structural comparison as described in [Overview on the comparison on page 51](#). In the detailed comparison, it is displayed above the actual comparison that it is true.

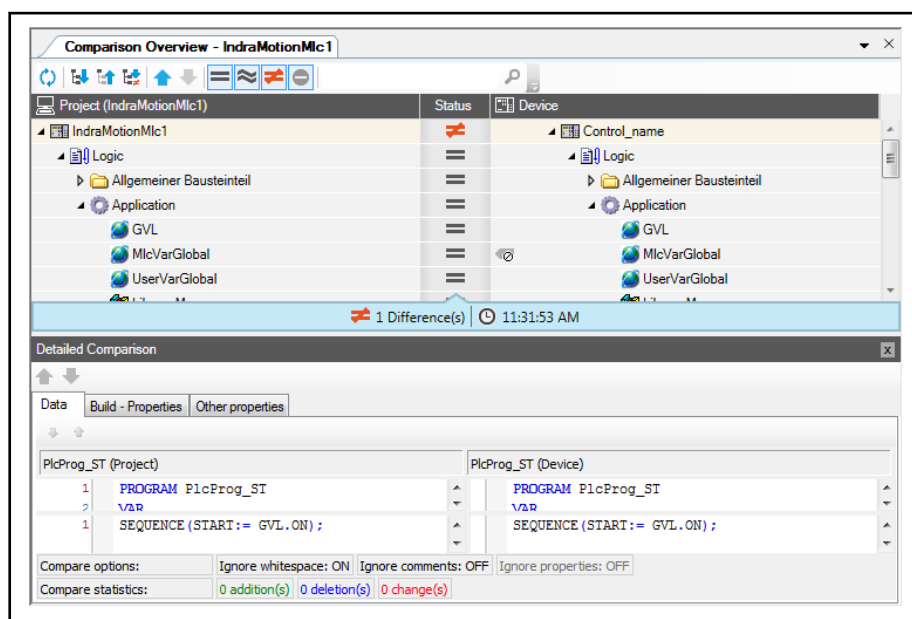


Fig. 5-3: Objects to be compared from different projects

If the object is only available on one site of the structural comparison, the source available is displayed in the detailed comparison. The note "No source data available" is displayed on the other site.

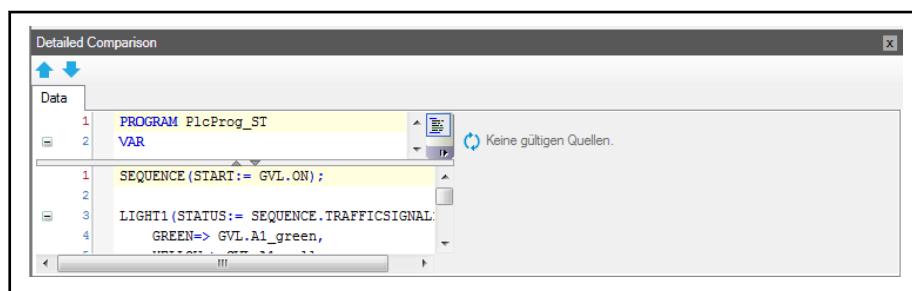


Fig. 5-4: Object is only available in the open project of the engineering PC:

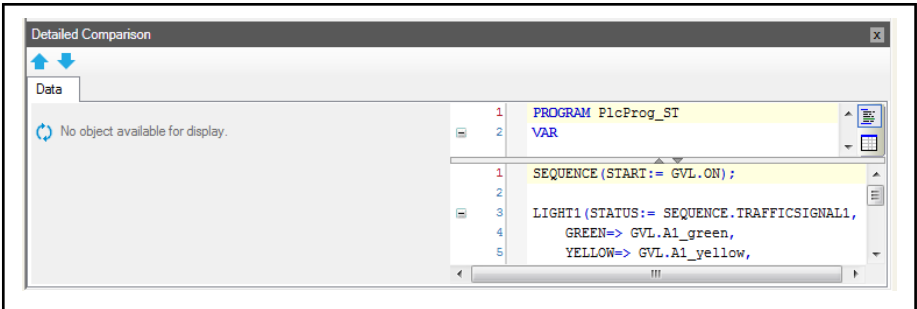


Fig. 5-5: Object only available on the control



Notes for comparison:

If the warning "Different application names" is displayed at the "Application" node, IndraWorks cannot detect whether the boot project on the control is up-to-date.

If this is the case, rename the application in the open project (as for the application name of the control). This detects whether the sources are present on the control and whether they match the running application. In this case, the user can make a comparison.

5.3 Options

There are setting options in the "Options" dialog to individually adjust the comparison options for the **Project comparison**. For the "Options" dialog, go to **Tools ► Options....**

Node comparison

Text settings	To adjust color and font for the comparison
External application	To specify an external comparison application to display the file differences. However, the comparison application has to be installed on the PC. Click on "..." and select the external application in the "Open" dialog.
PLC options	To specify the comparison for PLC elements. Enter whether to observe spaces, comments or properties.

For the **online comparison**, go to the context menu of the "Properties" dialog of the control to apply the PLC program sources to the control.

Control ► Properties ► PLC

Transferring PLC sources together with the boot application	To compare the program sources, this option has to be enabled additionally.

5.4 Error entries in the message window

Data for the online comparison is created and transferred to the control while the user can execute other actions in the project. Thus, it is possible that pro-

cesses required for the online comparison have not yet been completed and that new activities are pending.

Error messages in the category "Online comparison" inform in their message window. These error messages **exclusively** refer to the online comparison, i.e. a message is displayed for example:



Fig. 5-6: Example on a message in the "Online comparison" category

"The action LoginOnlineCompare cannot be executed for the device IndraMotionMlc1. An action is active for the online comparison of the device IndraMotionMlc1" if the "Login" was correctly executed for the application (i.e. the code was transferred correctly to the control!). However, the actions required (creation and transfer of the structure) for the online comparison could not be executed. A subsequent online comparison does not work properly, as the current data for the online comparison is not present on the control.

The following messages can be displayed:

- The action LoginOnlineCompare cannot...
- The action ClearOnlineCompare cannot...
- The action ClearAllOnlineCompare cannot...
- The action CreateBootApplicationOnlineCompare cannot...
- The action ResetOriginOnlineCompare cannot...

Ignore this message if the online comparison is not used. To use the online comparison, repeat the procedure. If the error still occurs, close the project and open it again before repeating the procedure.

6 Protecting and saving projects

General information on write and access protection

Protect a project against inadvertent changes via access protection and write protection. If required, provide it with read protection (know-how protection).

Write protection

The following options are available to easily set the entire project read-only:

- Set the status to **Released** in the **Library info** ("General module" folder).
- Enable the "Read-only" option in the properties of the project file in the local file system.

To protect only certain objects in a project against changes or to allow access only to certain users, use a user and access rights management (see below).

However, write protection and access protection do not serve as know-how protection of the function blocks!

IndraLogic as well as persons with knowledge of the project file format can view or modify function blocks created with IndraLogic.

Know-how protection

To ensure know-how protection of a project, encrypt the project file. Either use a project password or a certificate.

Protection is recommended via certificate, since no secret has to be shared between the authorized users in this case. The project encryption type is enabled in the project settings.

For the know-how protection of a library, provide it as target system-independent "protected library" (*.compiled-library). The library file does no longer contain source code in this format, but only encrypted precompile context. The compiler can still interpret this data. Whether access by other IndraLogic components is still possible depends on their functionality and is to be observed individually.

For the know-how protection and the copy protection of a boot application, use a certificate. For the activation, go to the object properties of the application.

Also refer to

- [chapter 6 "Protecting and saving projects" on page 57](#)
- [chapter 6.3 "Protecting objects in the project via access rights" on page 61](#)
- [chapter 16.1 "Information for library developers" on page 219](#)

User management and password manager

User accounts with different rights can be managed in IndraLogic. The actions used by the user to access a project object can be defined for each account.

The user management is configured in the **Project settings** in the category **Users and groups**.

Note the following before creating users and groups:

- Rights can only be assigned to user groups. Thus, assign each user to a group.
- There is always a group "Everyone" and every user and every other group is initially a member of this group by default. Thus, each user account is automatically equipped with at least the defined default rights.

The group "Everyone" cannot be deleted, but renamed. No members can be removed from this group.

Caution: By default, "Everyone" does not have the right to change the current user, group and rights configuration!

- There is always automatically a group "Owner" containing a user "Owner".

From V3.5, only the "Owner" has the initial right to change the current user, group and rights configuration in a new project!

Hence, only "Owner" can assign this right to another group.

In the first instance, the "Owner" can log in with user name "Owner" and an empty password. Further users can be added to the group "Owner" or users can be removed, but at least one member has to be retained. Like "Everyone", the group "Owner" cannot be deleted and it is always provided with all access rights. This prevents, that a project becomes unusable by denying all access rights for all groups.

Renaming the group "Owner" and the user "Owner" is possible.

- If the programming system or a project is restarted, no user is initially logged in to the project. However, the user can then log in via a certain user account with user name and password to obtain the access rights defined for the account.
- Each project has its own user management! To obtain certain access rights to a library integrated into the project for example, log into the library project explicitly.
Users and groups defined in different projects are not the same, even if their names are identical.
- A user management in a project is only reasonable if it is connected to the corresponding rights assignment for the access to project and objects. The project rights are generally managed in the **Rights** dialog of the **User management**. The access rights to an individual project object can also be changed on the **Access control** tab in the **Properties** of the object.
- There are default menu commands to log in and log out of a project as defined under **Project ► User management**. A password manager manages the login data on your computer.



From V3.5, only the "Owner" has the initial right to change the current user, group and rights configuration in a new project! Hence, only "Owner" can assign this right to another group.



IndraLogic stores the user passwords inaccessibly. The user account cannot be used without password anymore. Without the "Owner" password, the entire project can become unusable!

Password manager

The password manager allows to save login data records entered for project while logging in. It can be accessed via a button in the login dialog and provides fast access to the login data currently required. This can be useful for example when working simultaneously with multiple library projects protected by different passwords.

The password manager is protected by an individual master password. For the initial use of the password manager, IndraLogic prompts you to define this password in the configuration dialog of the password manager. IndraLogic remembers the master password until your current IndraLogic session is completed. The password has to be entered upon each initial login to the password manager during a new session or after changing the password.

Also refer to

- [chapter 6.3 "Protecting objects in the project via access rights" on page 61](#)
- [chapter 6.4 "Logging in via user account and password manager" on page 62](#)

Rights management

Rights management to access a project and objects in a project is required for a reasonable user management.

Configure the general rights management for a project in the **Rights** editor of the **User management**. The access rights to an individual project object can also be changed on the **Access control** tab of the **Properties** dialog of the object.

Before assigning rights, please observe the following:

- In a new project, IndraLogic always sets all rights for the execution of actions on objects with the **default value** "granted" (default right).
The only exception is the right to change the current configurations for users, groups and rights. Initially, only the "Owner" group has this right.
- If you are member of a group that is permitted to change rights, every right can be changed at any time when working further on a project.
Change a right by switching between "granted" and "denied" or by resetting to the default.

Also refer to

- [chapter 6.2 "Setting up user management" on page 60](#)
- [chapter 6.3 "Protecting objects in the project via access rights" on page 61](#)

Storing, saving

Provide the project file with the protection before saving it in the file system (see above). For a read-only project file, there are various options to still save the file. The options depend on the type of write protection.

If the project is to be opened later in an older IndraLogic version, save the project for precisely this version (file type), as IndraLogic informs immediately on possible data loss while saving.

To save library projects, observe the rules for library creation. Also consider the option of installing a library directly in a library repository.

To use a project on another computer, it is reasonable not only to save the project file, but also to create a project archive from all relevant additional files.

The creation of a backup copy of this project each time the project is saved, can be set. In addition, it can be configured that projects are automatically saved at certain time intervals.

Also refer to

- [chapter 3 "Exporting and transferring projects" on page 15](#)
- [chapter 6.5 "Saving the project" on page 64](#)
- [chapter 16.1 "Information for library developers" on page 219](#)

6.1 Setting up write protection

Protect a project against inadvertent changes using access protection and write protection. In addition, however, it can also be provided with read protection (know-how protection). There are the following options:

Providing a project with the attribute "Released"

Prerequisite: Project is open.

1. Select **<ProjectName> ► General module folder ► Library info** and then the **Summary** tab.
2. Select the option **Released** and press **OK**.
3. Save the project, e.g., press **<Ctrl>+<S>**.



The meta data **Released** is applied to library projects as well as standard projects!

If a standard project is released for example, it is not permitted to store a new list with **persistent variables**. This is only possible after deleting the metadata.

Also refer to: [chapter 9.16.5 "Saving the values of a persistent variable list in a recipe" on page 144.](#)

Also refer to

- [chapter 18.1.20 "Library info" object" on page 545](#)

Providing a project in the file system with the property "Read-only"

1. Provide the project file with the property attribute "Read-only" in its local file system.

If the project is already open and it is tried to save the project under the same name, a dialog is displayed that there is a write protection. This dialog provides the following options:

To save the project under another name or in another path, press **Save as....**

Cancel the saving to remove a possible write protection on the disk.

When opening the project again, the line `The project cannot not be saved...` is displayed in the upper right of the main window.

6.2 Setting up user management



It is about the user management for an IndraLogic project file.

The following instruction describes how to customize the user management for the first time in a project. It deals with the definition of a user and a group to which the user belongs.

Prerequisite: The project to set up the user management for, is open. There is no customized user configuration yet.

1. Select **Project ► User management ► Users and groups** and then the **Users** tab. The user `Owner` is already created by default.

2. Click on **Add**.

The dialog **Add user** dialog opens.

3. Enter a login name, for example "Dev1" and a password.

Do not disable the option **Enabled**. Click on **OK**.

When creating a group for the first time, IndraLogic prompts you to authenticate yourself to perform this action.

In this case, enter "Owner" as the **User name**. Do not enter a **password** and click on **OK**.

The user `Dev1` is displayed in the list and is automatically a "member of the group "Everyone"".

4. Change to the **Groups** tabs to add the user to a new group.

The groups `Everyone` and `Owner` have already been created.

5. Select **Add** to open the **Add group** dialog.
6. Enter at least one name for the new group, e.g. "Developers". Select the entry **User 'Dev1'** in the **Members** field. Click on **OK**.

The group **Developers** is now displayed with "has user member `Dev1`".

7. Change to the **Users** tab.

The user **Dev1** is now displayed as a member of the groups "Everyone" and "Developers".



The user management from a different project can be applied using the **Export/Import** functionality in the **Project settings** dialog in the **Users and groups** category.

Also refer to

- [chapter 6 "Protecting and saving projects" on page 57](#)
- [chapter ""Rights" command" on page 601](#)

6.3 Protecting objects in the project via access rights

Protecting individual objects by setting access rights in the "Rights" editor

1. Select **Project ► User management ► Rights**
The window of the **Rights** editor opens. The action categories are shown on the left. The current user groups are located on the right.
2. Expand the relevant action category and below it the action for which the right should be changed.
3. Select the goal of the action in the **Actions** window. In the **Rights** window, select the group for which the right should be changed. Several items can be selected at once.
The toolbar buttons are enabled.
4. Click on the appropriate button to change the right of the group for the action on the target object.
IndraLogic updates the symbol in front of the group according to the new right. The right is applied immediately.

Also refer to

- [chapter ""Rights" command" on page 601](#)

Protecting individual objects by setting access rights in the object properties

It can be configured whether the members of a group have the right to view, edit or remove the object and to add/remove child objects to/from the object.

1. Select the object in the Project Explorer.
2. Select **Properties** from the context menu and the category **Access control** in the dialog.
3. In the table under **Groups, actions and rights**, double-click on the symbol of the right to be changed.
A selection list of the possible rights is displayed: **Grant**, **Deny**, **Reset**.
4. Select the right and click on **Accept** or **OK**

The right is immediately applied for the action and the group. The symbol changes accordingly.

Also refer to

- [chapter ""Properties" - "Access control" dialog" on page 706](#)

6.4 Logging in via user account and password manager

Logging in to a project without using the password manager functions

Prerequisite: A project is open.

To log in as a defined user for this project or for a library integrated in it to edit it with certain rights. You have the required login data for the respective project or the library.

1. Select **Project ► User management ► Log in user....**

The **Login** dialog opens.

2. Select the project file from **Project/library** and enter the required login data **User name** and **Password**.
3. Log in with **OK**.

If another user is already logged in, this user will automatically be logged out by the new login.

Setting a master password for the password manager

*Prerequisite: A project is open. The **Login** dialog is open to log in as a defined user for a project or for a library integrated in the project. You want to use the password manager to save login data.*

1. Select **Project ► User management ► Log in user....**

2. Click on  in the **Login** dialog.

If you are working with the password manager for the first time, the dialog **Password manager configuration** opens.

3. Enter a character string as the future master password. Confirm it in the second line and click on **OK**.

IndraLogic remembers the master password until the current IndraLogic session is completed. The password has to be entered upon each initial login to the password manager during a new session or after changing the password.



If you have forgotten your master password, you can no longer access the login data already saved! In this case, the only option is to reset the password manager. Passwords have to be saved again to the manager!

Saving login data in the password manager

Prerequisite: A project is open. To log in as a defined user for this project or for a library integrated in it to edit it with certain rights. You have the required login data for the respective project or the library. These login data has not yet been saved in the password manager.

1. Select **Project ► User management ► Log in user...** to open the **Login** dialog.
2. Select the project file from **Project/library**.
3. Enter the user name and password for the project or the library.
4. Click on .

When working with the password manager for the first time, you are requested to define a master password. Refer to the instruction "Setting a master password for the password manager" shown above.
When calling the password manager for the first time in this IndraLogic session, you are requested to enter the master password.

5. Enter the master password upon request.
The password manager menu is displayed.
6. Select the option **Save login data locally on this computer**.
It is logged in. Data is saved in the password manager.

Getting login data from the password manager

Prerequisite: A project is open. To log in as a defined user for this project or for a library integrated in it to edit it with certain rights. The required login data is already saved in the password manager.

1. Select **Project ► User management ► Log in user...** to open the **Login** dialog.
2. Click on .

When working with the password manager for the first time, you are requested to define a master password. Refer to the instruction "Setting a master password for the password manager" shown above.
When calling the password manager for the first time in this IndraLogic session, you are requested to enter the master password.

3. Enter the master password upon request.
The password manager menu is displayed.
4. Select the appropriate entry **Use saved login data for <UserName>**.
5. It is logged in automatically with the data retrieved from the password manager.

Opening password manager, changing master password

Prerequisite: A project is open. You want to open the password manager to view or edit the entries or to change the master password. You have already logged in once with the master password.

1. Select **Project ► User management ► Log in user...** to open the **Login** dialog.
2. Click on .

Select **Open password manager**.

The password manager window opens.

3. Select **Change master password** and make the change.

Logging out from project

*Prerequisite: A project is open. A user is logged in. This can be seen due to a name entry in the field **Current user** in the status bar.*

1. Select **Project ► User management ► Logout user....** Alternatively, double-click on the field **Current user** in the status bar.

If the user is logged in to only one project, he is now be logged out without further interaction. - Is displayed again in the field **Current user** in the status bar

If the user is logged in to several projects, the dialog **Logout** opens. Select the specific project or library project from which the user is to be logged out.

Also refer to

- [chapter 6 "Protecting and saving projects" on page 57](#)
- [chapter ""Log in user" command" on page 597](#)
- [chapter 6.2 "Setting up user management" on page 60](#)

6.5 Saving the project

Saving a project under the same name

Prerequisite: The project is open. The project file is not write-protected.

1. Select **Project ► Save**.

IndraLogic saves the project file with the current project name displayed in the title bar of the main window. If the project has been changed since it was saved last, the project name is provided with an asterisk.

Saving a project under a different name or format

Prerequisite: The project is open.

1. Select **Project ► Save as....**

The **Save project** dialog opens.

2. Select a storage location in the file system as well as the **file type** (project file (*.xiwp) or library file (*.xiwp)) and the storage version. If the project is to be opened later in an older IndraLogic version, save the project for precisely this version, as the message window immediately outputs possible data loss.

If the project file is not write-protected, IndraLogic saves it in the selected path. Otherwise, you are informed on how to proceed.



A write protection specified in IndraLogic does not change the response (**General module folder ► Library info ► Summary "Released"**).

Saving project after reminder

Prerequisite: The project is open.

1. Select **Tools ► Options ► General ► Projects**.

Save project is displayed.

2. Select the option **Create backup copy**.
3. Enable the option **Remind saving for changes in this minute interval** and select a time interval.
4. Click on **OK** to close the **Options** dialog.

Also refer to

- [chapter 6 "Protecting and saving projects" on page 57](#)

7 Multilingual PLC comments

PLC comments can be displayed in different languages. PLC localization data is used for this functionality.



PLC comments can be displayed in different national languages. However, editing is (currently) only possible in the interface language set under options when the project was created.

After adding an additional language via the menu **Project ► Language ► Manage project languages**, the node “PLC localization data” is displayed under “General module folder”. The PLC localization table can be opened via this node. There is a separate column for each national language.

To add a new national language, proceed as follows:

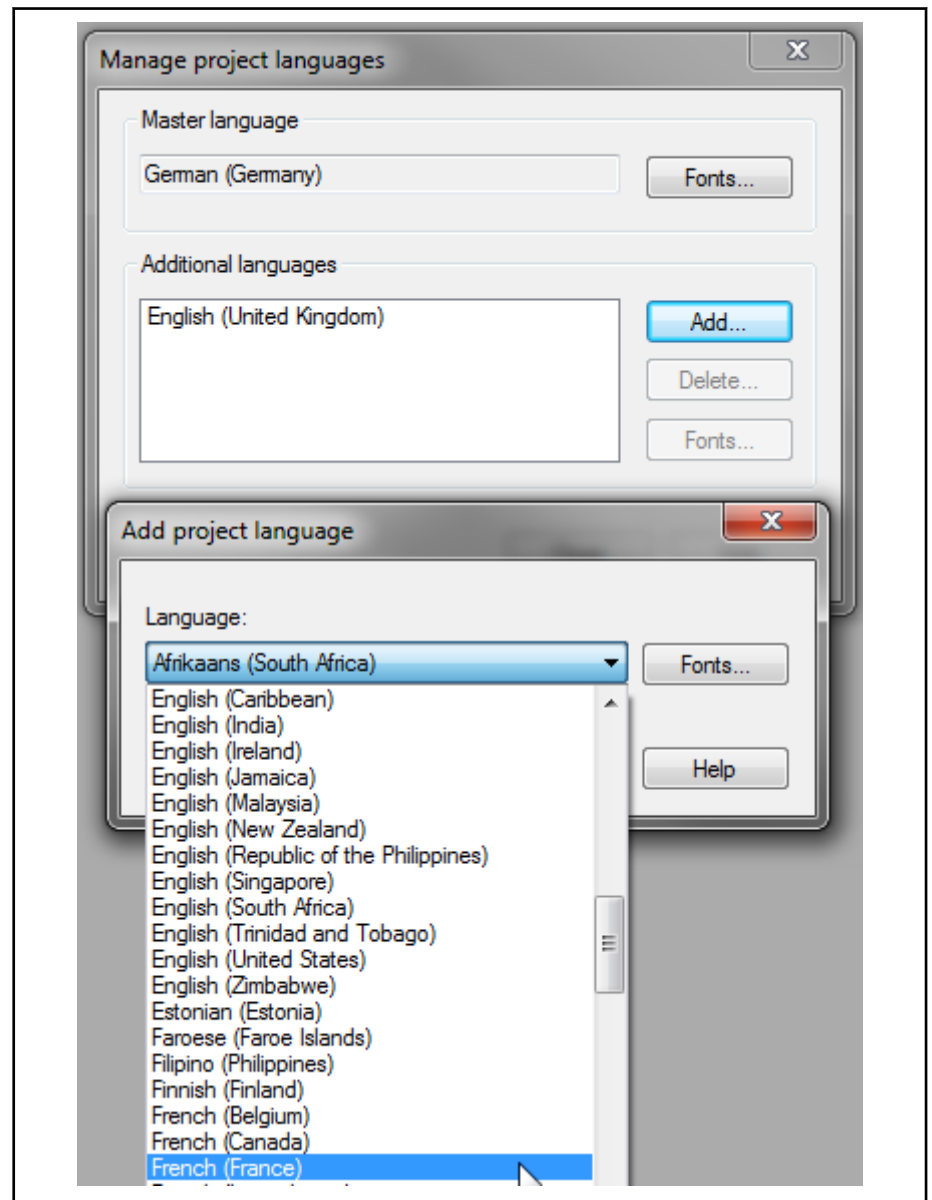


Fig. 7-1: Adding project language

Multilingual PLC comments

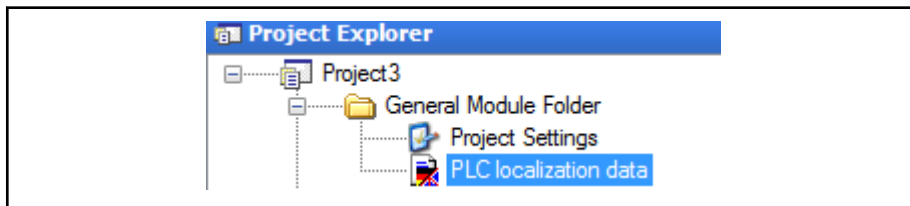


Fig. 7-2: "PLC localization data" node enabled

PLC comments are translated into another national language via the export file (Excel file). An own translation file can be generated for each national language via the menu item **Project ► Language ► Export translation file**. It includes the freely selectable source language required for translation as well as the national language mentioned above as target language. Use a suitable editor (e.g. Excel) to edit and subsequently import the csv file. It is imported via **Project ► Language ► Import translation file**.

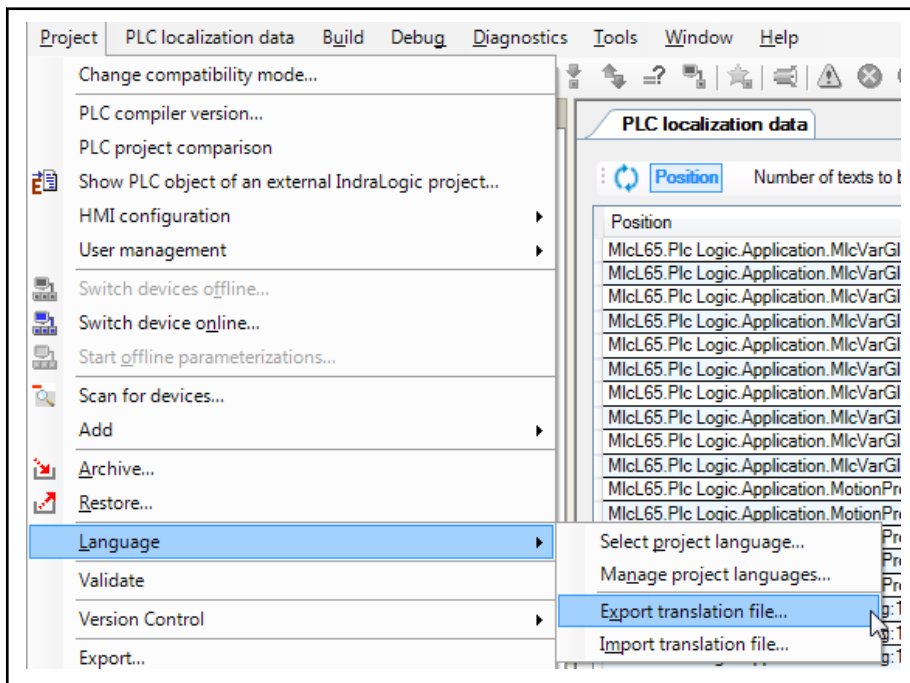


Fig. 7-3: Exporting translation file

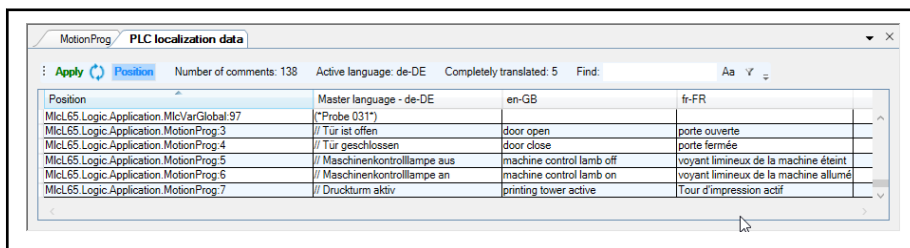


Fig. 7-4: PLC localization data table in different languages

Double-click on a line to directly enter the translation of the comments. The menu item **Apply** refreshes the translated comments also in opened editors.

The program paths on the comments can be shown and hidden using the menu item **Position** in the PLC localization data table. If the program paths are hidden, the respective program path is displayed in the tooltip.

Double-click on the original comment to open the position and to highlight the source in the POU.

Comments can be sorted by the selection in the top line.

Entire comments or individual character strings can be searched or using the menu item **Search**. The sources are highlighted. The remaining lines can be shown or hidden using the filter symbol. The menu item **Aa** facilitates a case sensitive search.

To select the language of the comments, go to **Project ► Language ► Select project language**. The status line additionally displays the selected language.

If another language is selected as master language, it is automatically returned to the master language when comments are edited.

8 Configuring an I/O link

The hardware to be controlled can be mapped in your IndraLogic project using device objects in a tree structure (Project Explorer). Thus, linking hardware and application is easy to handle:

Settings for the communication between IndraLogic and the control and especially for the I/O assignment can be made in the configuration editors of the device objects. The I/O assignment, also called I/O mapping, is the linking of the inputs and outputs of the control to the variables of your application.

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 8.2 "Devices in the Project Explorer" on page 70](#)
- [chapter 8.3 "Configuring devices and I/O mapping" on page 73](#)

8.1 Project Explorer and device editor

Project Explorer In the **Devices** view (also called the **Project Explorer**), organize applications with regard to the target device. In this view, the hardware of one or multiple control devices or field bus systems can be shown, the communication can be configured and multiple applications can be assigned.

The root node of the Project Explorer is a symbolic node entry **<Project-Name>**.

Insert device objects below this node for one or more controlling devices, also called target systems. Each device object represents a specific hardware, e.g. control unit, field bus node, bus coupler, drive, I/O module, monitor.

When adding objects, a wizard provides the matching devices from your local device repository.

If you are already connected to a control network, scan the hardware for available devices and save them to the current configuration in the Project Explorer.

Specific rules apply (see below) when creating device objects in the Project Explorer (mapping to the hardware environment to be controlled). The hierarchical layout of application objects and device objects defines the scope of other objects, such as libraries and GVLs.

There are programmable devices and parameterizable devices. The device type defines the possible insertion position in the Project Explorer and the selection of objects that can be inserted below the device. Programmable devices automatically receive an additional node **PLC Logic** below the device object (only for organizational purposes). Insert the objects required for programming the device (this means the application(s) and GVLs or text lists, for example) below this node.

Each device is defined by its device description and has to be installed on the local system to be insertable into the Project Explorer. The device description file defines the device properties for configurability, programmability and possible connections to other devices.



The "General module folder" includes objects that can be used across the project.

Programming objects intended for a specific application have to be inserted below the application object in the "Project Explorer" view.

- Observe that the active application can run on a "simulation device" provided by default within the programming system. In simulation mode, the online functionality of the application can also be tested without hardware. To enable the simulation, go to the context menu of the control.
- Device editor** Configure the settings for the communication between IndraLogic and the target device in the tabs of the device editor. To open the editor, double-click on the device object in the Project Explorer. The editor includes generic tabs as well as device-specific tabs. Its title contains the device name.
- Also refer to*
- [chapter 18.1.9 "'Device' object and generic device editor" on page 490](#)
 - [chapter 8.3 "Configuring devices and I/O mapping" on page 73](#)

8.2 Devices in the Project Explorer

All objects required for executing an application (a control program) on a device (control, PLC) are managed in the Project Explorer in a tree structure ([chapter 18.1.2 "'Application' object" on page 480](#)).

These objects are also referred to as "Resource" objects. Device objects, application objects, task configuration and task are "Resource" objects.

Programming objects such as individual POU's, global variable lists and library managers can be managed below a control in the Project Explorer and can then be used only for your application.



Globally applicable programming objects are managed in the "General module" folder in the Project Explorer.

To convert device references when opening projects created in another format, see

IndraMotion MLC - Project Conversion.

Control configuration

The "control configuration" illustrates the target hardware in the programming system in order to make the inputs and outputs and the control parameters and the applications field bus devices accessible. In addition, it enables the available device parameters to be displayed.

The "Control configuration" tree is integrated below the device node in the Project Explorer. Other objects required to run an application on a target system are also arranged.

The map of the current hardware configuration below the "Devices" node is simplified in the default device editors by a scan functionality.

The control inputs and outputs for project variables are assigned either using the AT declaration in the declaration editor or in the "I/O mapping" dialog of the respective field bus which provides dialogs to configure a device.

If a new control is added to the Project Explorer, a preset enters the application automatically added as "mapping application".

Also refer to

- [chapter 9.10.2 "AT declaration" on page 125](#)

For details, refer to

- Mapping dialog of the individual field buses (Rexroth IndraWorks 12VRS/13VRS/14VRS, Field Buses, DOK-IWORKS-FB*****V1x-AP02-EN-P)

General information on the Project Explorer

- The root node in the Project Explorer is the project name specified when a new project is created.  <project name>.
- The configuration trees for the "control configuration " and "task configuration" that were handled in separate windows in IndraLogic 1.x are integrated into the device object tree in IndraLogic 2G. The configuration of the devices and task parameters is carried out in separate editor dialogs.

Also refer to

- [chapter 9.15 "Task configuration" on page 136](#)
- ["Control configuration" on page 70](#)

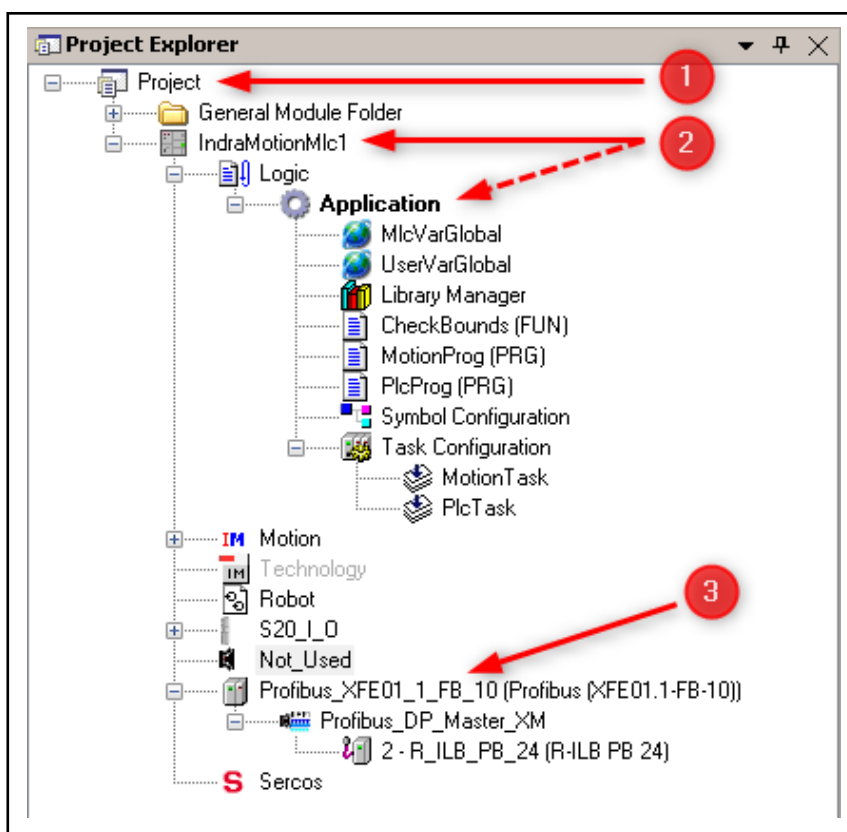
This way, the structure of the hardware environment to be controlled is illustrated in the device object tree with the corresponding arrangement of objects and it is possible to superimpose a heterogeneous system of controls with multiple networking and underlying field buses.

See the rule for arranging objects below the device node in the following.

- A **Devices** object represents a certain hardware (target system).
Examples: control device, drive, I/O module, monitor.
- An entry in the Project Explorer shows the symbol, the symbolic device name (can be edited in the Explorer) and the device type behind it (= device name as defined in the device description).
- There are **programmable** devices and devices that can be **parameterized**. The device type determines the possible insertion position below the device node and the selection of objects that can be added below the device.

Programmable devices automatically obtain an additional  **Logic** node below the device object. The objects required for device programming (visualizations, GVLs, text lists, etc.) can then be added below this node:

Configuring an I/O link



- (1) Project name
 (2) Device, programmable, with application
 (3) Device, parameterizable

Fig. 8-1: Devices in the Project Explorer

- In a project, one or multiple programmable devices can be configured irrespective of the manufacturer or type ("[Arranging and configuring objects in the Project Explorer](#)" on page 72).
- Communication, parameters or I/O mapping of a device are configured in the "Device editor" dialog. Double-click on the device entry to open the dialog ([chapter "Generic device editor" on page 490](#)).
- In **online mode**, symbols at the device entry contain additional information (refer to the tooltips).

For additional diagnostic information on the PLC component of the device, refer to the logbook (double-click on the device object) ([chapter ""Log" tab" on page 495](#)).

Arranging and configuring objects in the Project Explorer

- The object types that can be added depend on the currently selected position in the tree.

Example:

Modules for a DP Profibus slave cannot be inserted without inserting the respective slave object before.

Moreover, the only devices that can be selected for insertion are those that have been correctly installed in the local system and that are suitable in the currently selected position in the Explorer.

To add an object, highlight the position in the tree where the object is to be inserted and click on **Add ► <ObjectType>** in the context menu. Alternatively, insert objects using the mouse to drag them from the library to the corresponding position.



Programmable devices can only be added from the "Drive and Control" library. Highlight the device (e.g. IndraLogic MLC L65) and drag it to the root node.

- The arrangement of objects below an application is sorted alphabetically and by object type. It is not possible to change to any position. On the other hand, the objects below (e.g. actions, transitions) can be positioned as desired by using the mouse to drag them to the corresponding position.
- A device already added to the Project Explorer can be replaced by another version of the same device or by a device of another type. The configuration tree indented below the device can be retained as much as possible. To do this, highlight the device node and select **Update Device** in the context menu.
- Devices can be installed and uninstalled in the Device Database... dialog. The installation is based on "device description files" in XML format. The default extension for a valid description file is *.devdesc.xml. However, bus-specific description files, e.g. *.gsd files (Profibus) can also be installed using the "Device repository" dialog.
- A device is added as node in the Project Explorer. If these are defined in the device description, the subnodes are automatically added as well. In turn, a subnode can also be a programmable device.
- Further devices that are installed in the local system and are provided in the library can be added below a "Device" object. For each level, the programmable devices are arranged first (PLC logic), then the other types.
- An application is automatically added below the "Logic" node (symbolic node for programmable devices). Only one application is permitted for each device. Subsequently, the other objects required for programming, e.g. data types (DUT), Global Variable Lists (GVL) and visualizations can be added below an application. A task configuration is automatically added below every application. Define the corresponding program calls.

8.3 Configuring devices and I/O mapping

Configuring devices

Configure the device objects inserted into the Project Explorer in the associated device editor. The options depend on the device description. The "generic device editor" provides tabs that can be supplemented by device-specific tabs.

Prerequisite: A default project is open. A standard PLC and a field bus device object below it are added to its Project Explorer.

1. Double-click on the device object of the standard PLC in the Project Explorer of your project.

The **<DeviceName>** editor is displayed in the IndraLogic main window. The **Applications** tab is in the foreground. Change to the other tabs in order to make configuration settings for the control.

2. Double-click on the field bus device object in the Project Explorer of your project.

The **<FieldBusDeviceName>** editor is displayed in the IndraLogic main window. Specific tabs are available for specific devices. For the configuration options, please refer to the help pages of the respective device

editor. If the option **Display generic configuration editors** is selected in **Tools ► Options**, category **Device editor**, also refer to the tabs provided by the generic device editor.

Also refer to

- [chapter "Generic device editor" on page 490](#)

General information on I/O mapping ("Mapping")

Whether you can configure an I/O mapping to project variables or even to the entire function blocks depends on the type of device.

Configuring an I/O map means linking input and output channels of the device with the project variables. The term **"mapping"** is also used.

Observe the following for the mapping of inputs and outputs of a device to variables in IndraLogic:

- Variables that are mapped to an input cannot be accessed for writing.
- An existing variable can only be mapped to one input.
- New global implicit variables can be directly generated in the I/O image and the variables can be mapped to a device channel.
- The device specifies the memory layout of structures.
- Addresses can be changed and values can be specified in the I/O mapping.
- For each variable assigned to an I/O channel in the dialog **I/O mapping**, "force variables" can be generated while compiling the application (see further below). Use these variables to force a value to an input or an output when commissioning a system via a visualization/HMI for example.
- Transfer changes in the I/O mapping to the control via online change.
- If a pointer to a device input is used, the access is considered to be a write access, for example `pTest := ADR(input);`.

This outputs a compiler warning when code is generated : "...invalid assignment target".

If such a construct is required, copy the input value `input` to a variable with write access.

- An I/O address can also be linked to a variable via the "AT declaration" in the IEC code.

Since a device configuration changes often, we recommend to make assignments only in the device editor.

When using the AT declaration, note the following:

- An AT declaration is only allowed for local or global variables and not for input or output variables of function blocks.
- Implicit "force variables" for I/Os (see below) cannot be generated for AT declarations.
- If an AT declaration is used for structural variables or function block variables, all instances access the same memory location. This corresponds to the use of "static variables" in classic programming languages such as "C".



If a pointer to a device input is used, the access (e.g. `pTest := ADR(input);`) is considered to be a write access. This outputs a compiler warning when code is generated : "...invalid assignment target".

If this is required, copy the input value (`input`) to a variable with write access first.



Alternatively, assign a variable to an address in the programming code using the AT declaration.

Due to possible changes in the device configuration, we recommend to make assignments only in the device editor.



You can export the I/O mapping configuration of a device to a "csv" file or import it from such a file.

Also refer to

- [chapter "Tab '<Device name> I/O mapping'" on page 504](#)
- [chapter "'Export I/O mapping to CSV' command" on page 610](#)

Linking a device input to an existing project variable ("mapping")

Prerequisite: A device, which supports an I/O mapping configuration in IndraLogic, is inserted into the Project Explorer of your project. The input and output channels of the device with specification of the addresses and data types are displayed as table on the **I/O mapping** tab in the device editor.



Mapping data types that are "too large"

If a variable of a data type, which is larger than one byte, is mapped to a byte address, the value of the variable is cut to the byte size!

For the monitoring of the variable value in the dialog **I/O mapping**, this means:

The current value in the project is shown in the "root" element of the address. The individual present bit values of the byte are displayed in succession in the bit elements below that, but this may not be sufficient for the entire variable value.



If a UNION is represented by I/O channels in the "Mapping" dialog, it depends on the device whether mapping to the "root" element is also possible.

1. In a block, declare, for example, a variable `xBool14` of the type `BOOL` with which you want to access an input of the target device from the application.
2. Double-click on the device object in the Project Explorer to open the device editor and go to the **I/O mapping** tab.
3. Observe the **Variable** column with the display of the device input channels  and device output channels , which can still be sorted by organizational nodes  depending on the device. It is assumed that there is a device input of the type `BYTE`. It is displayed with its individual bit addresses (bit channels) below the `BYTE` node.

4.



When mapping structured variables, the editor does not allow you to enter the structural variable (e.g. to %QB0) as well as the individual structural elements (e.g. to %QB0.1 and QB0.2).

That means: if there is a main entry with a subtree of bit channel entries in the mapping table, enter a variable either in the line of the main entry or in the lines of the subelements (bit channels), but not in both.

Either assign the entire channel with a variable of a suitable type **OR** assign its individual bit-channel addresses with suitable variables of the type **BOOL** or **BIT**. Double-click on a bit input channel in the **Variables** column.

An input field opens.

5. To place an existing variable on the channel, enter the project variable with the complete path. Open the input assistant via . Select for example the variable `Application.PlcProg.xBool4` declared in `PlcProg`.

The variable is inserted. The  symbol is displayed in the **Mapping** column.

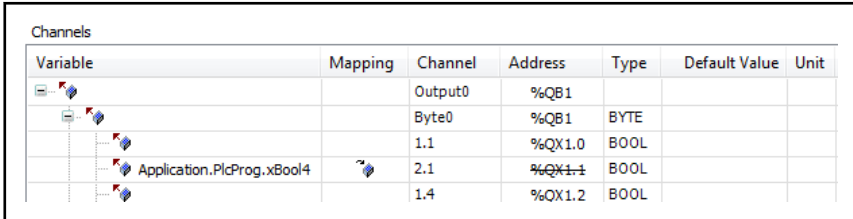
The address is now struck through. That does not mean that the address is no longer available. Values of existing variables are managed at another storage location.

But:

To avoid ambiguities when writing the values, do not assign another variable to the address, especially for outputs.



From IndraWorks 14V22, an initialization value of the variable is used as default value for a mapping on an existing value. In the "Default value", editing is only possible a newly generated variable is mapped or if no mapping is entered. In previous versions, the user had to ensure that default value and initialization value are identical.

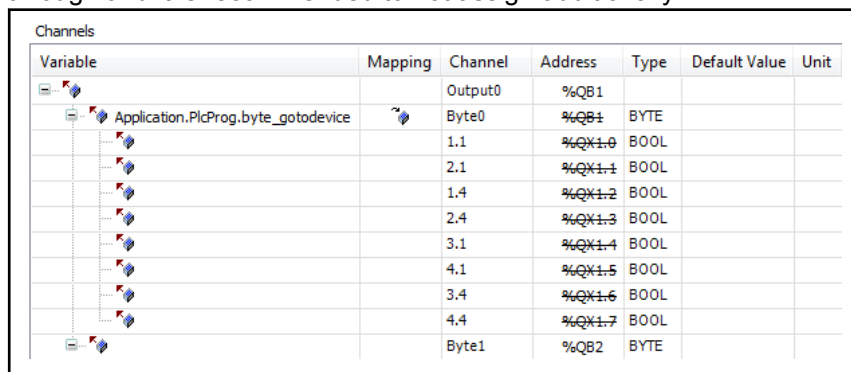


Variable	Mapping	Channel	Address	Type	Default Value	Unit
		Output0	%QB1			
		Byte0	%QB1	BYTE		
		1.1	%QX1.0	BOOL		
Application.PlcProg.xBool4		2.1	~~%QX1.1~~	BOOL		
		1.4	%QX1.2	BOOL		

Fig. 8-2:

6. Delete the variable assignment and the declaration of `xBool4` again.
7. In "PlcProg", declare a variable `byte_gotodevice` of the type **BYTE** for example which is used to access an input of the target device from the application.
8. Select the root of the channel, the **BYTE** node. Use the input assistant again to select the variable `Application.PlcProg.byte_gotodevice`.

The variable is inserted, all bit addresses of the main channel are struck through and it is recommended to not assign additionally.



Variable	Mapping	Channel	Address	Type	Default Value	Unit
Application.PlcProg.byte_gotodevice		Output0	%QB1	BYTE		
		Byte0	%QB2	BYTE		
		1.1	%QX1.0	BOOL		
		2.1	%QX1.1	BOOL		
		1.4	%QX1.2	BOOL		
		2.4	%QX1.3	BOOL		
		3.1	%QX1.4	BOOL		
		4.1	%QX1.5	BOOL		
		3.4	%QX1.6	BOOL		
		4.4	%QX1.7	BOOL		
		Byte1	%QB2	BYTE		

Fig. 8-3:

9. Delete the variable assignment and the declaration of `byte_gotodevice` again.

Also refer to

- [chapter "Tab '<Device name> I/O mapping'" on page 504](#)

Mapping a device input to a newly created project variable

Now map a device output to a global implicit variable. Create it directly in the I/O mapping dialog for this purpose.



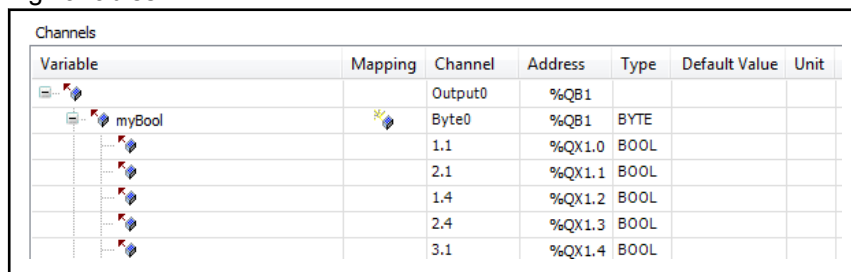
The dialog **I/O mapping** is thus another place for declaring a global variable.

Prerequisite: A device, which supports an I/O mapping configuration in IndraLogic, is inserted into the Project Explorer of your project. The input and output channels of the device with specification of the addresses and data types are displayed as table on the **I/O mapping** tab in the device editor.

1. Double-click on the device object in the Project Explorer to open the device editor and go to the **I/O mapping** tab.
2. In the mapping table, click on a channel entry in the **Variable** column to open an input field.
3. Enter a simple name (without ".") for a new variable, e.g. `myBool`.

IndraLogic creates the variable as an implicit global variable in the project and assigns it directly to the channel address.

Thus, the address is not struck through as in case of mappings to existing variables. .



Variable	Mapping	Channel	Address	Type	Default Value	Unit
myBool		Output0	%QB1	BYTE		
		Byte0	%QB2	BYTE		
		1.1	%QX1.0	BOOL		
		2.1	%QX1.1	BOOL		
		1.4	%QX1.2	BOOL		
		2.4	%QX1.3	BOOL		
		3.1	%QX1.4	BOOL		

Fig. 8-4:

4. Delete the variable assignment of `myBool` again.

Configuring an I/O link

Linking a device with a function block instance

If supported by the device, you can map entire function blocks to an input or output channel. This allows you to count the frequency of signal changes or scale a channel value for maintenance purposes, for example.

In the following example, map a device output channel to a function block. The function block scales the channel output value.

Prerequisites:

- A device with a digital output that supports function block mapping is linked in the project.
- There is a function block `Scale_Output_Int` with the following implementation

The attributes of the function block itself and before the output parameter with which the channel output is processed are important.

```
{attribute 'io_function_block'}
FUNCTION_BLOCK Scale_Output_Int
VAR_INPUT
    iInput : INT;
    iNumerator : INT;
    iDenominator : INT :=1;
    iOffset : INT := 0;
END_VAR
VAR_OUTPUT
    {attribute 'io_function_block_mapping'}
    iOutput : INT;
END_VAR
VAR
END_VAR
END_VAR
IF iDenominator <> 0 THEN
    iOutput := TO_INT(TO_DINT(iInput) * TO_DINT(iNumerator) / TO_DINT(iDenominator)) + iOffset;
```

Procedure to link a device channel

1. Double-click in the Project Explorer and open the generic device editor of the affected device module
2. Open the I/O mapping tab in the device editor
3. Double-click on the output that should be connected to the function block
4. Click the button  "Add FB for I/O channel"

The "Select Function Block" dialog opens.

On the left side, the function block `Scale_Output_int` can be found below the "Application" node.

Libraries linked in the project that contain corresponding function blocks are also displayed for selection.

5. Select the `myScaleOutputInt` function block and confirm the dialog with "OK"

After confirming with OK, the path of the function block parameter `iOutput` is entered in the "Variable" column in the mapping dialog. The path comprises the application name, the device channel name, and the selected FB output.

Example: `App1.Out_4_Int_myScale_Output_Int_1.iOutput`

6. Select the channel and click on  "Go to instance"

The focus switches to the "IEC objects" tab and the created entry for the new IEC object `Out_4_Int_myScale_Output_Int_1`.

In this view in online mode, you see the current value of the parameter `iOutput` for the channel `Out_4_Int` scaled by the FB.
You can also write and force the value as in other monitoring views.

Also refer to

- [chapter 18.3.1 "Dialog "Select function block"" on page 700](#)
- [chapter "Tab '<Device name> I/O mapping'" on page 504](#)
- [chapter "Tab '<Device name> IEC objects'" on page 506](#)
- [chapter "Attribute 'io_function_block', 'io_function_block_mapping'" on page 393](#)

Changing and specifying an address value in the I/O mapping

Prerequisite: Your project has I/O mapping. Refer to the previously mentioned sections in this chapter.

The address value of an entire channel (but not that of an individual subelement of the channel!) can be changed and fixedly set in the mapping table.

Double-click in the Project Explorer and go to the **I/O mapping** tab to open the generic device editor of the I/O object..

By a fixed setting, the addressing can be adjusted to a specified machine configuration and thus retain the address value even if the module layout changes.

Without a fixed setting, the changing of the module layout automatically adapts the address values.

1. Double-click in the Project Explorer and go to the **I/O mapping** tab to open the generic device editor of the I/O object..
2. In a "PlcProg", declare a variable `xBool4` of the type `BOOL` for example used to access an input of the target device from the application. Assign the variable again as described.

Variable	Mapping	Channel	Address	Type	Default Value	Unit
		Output0	%QB1			
		Byte0	%QB1	BYTE		
		1.1	%QX1.0	BOOL		
		2.1	%QX1.1	BOOL		
		1.4	%QX1.2	BOOL		

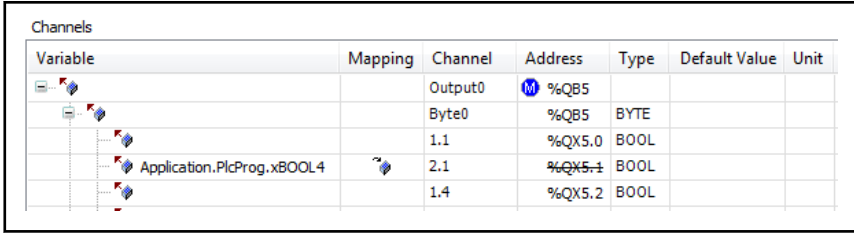
Fig. 8-5:

3. In the mapping table, click on a channel entry in the **Address** column to open an input field. This is only possible for the "root" address of a channel, not for individual addresses of its subelements!

Therefore, change the top address entry of a channel in the table, for example from `QB1` to `QB5`. Exit the input field.

The address value is changed. The  symbol precedes the address. It indicates that the address is specified. The addresses of the subelements of the channel are also changed accordingly. If the position of the device object within other device objects is changed in the Project Explorer using input/output channels, IndraLogic does not adapt these addresses to the new sequence, as in case of no specification.

Configuring an I/O link



Variable	Mapping	Channel	Address	Type	Default Value	Unit
		Output0	%QB5	BYTE		
		Byte0	%QB5	BYTE		
		1.1	%QX5.0	BOOL		
Application.PlcProg.xBOOL4		2.1	%QX5.1	BOOL		
		1.4	%QX5.2	BOOL		

Fig. 8-6:

- To undo the manual change or fixing, open the input field of the address value again, delete the address entry and press Enter.

IndraLogic resets the address and the subsequent addresses to the values before the change and removes the symbol .

Also refer to

- chapter "Tab '<Device name> I/O mapping'" on page 504
- chapter 17.4.13 "Addresses" on page 348

Configuring I/O variable update

Depending on the device linked to the project, the variables present to its inputs and outputs are differently updated in IndraLogic. The respective settings can be explicitly made in the **I/O mapping** dialog.

Also refer to

- chapter "Tab '<Device name> I/O mapping'" on page 504

Monitoring variables in the I/O mapping in online mode

Prerequisite: An application with a device configuration containing I/O maps was successfully compiled. The corresponding hardware and the bus system are running. You connected to the control via **Debug ► Login**, loaded and started the application.

- Open the **I/O mapping** tab of the control device in the device editor. To open the editor, double-click on the device object in the Project Explorer.

Now, The mapping table additionally contains the columns **Current value** and **New value**.

If a structural variable is mapped to the "root" element of an address¹, IndraLogic does not display any value in this line in online mode. If, for example, a **DWORD** variable is mapped to the address, however, the respective values are monitored in the "root" line as well as in the indented bit-channel lines below.

The field in the "root" line always remains empty if the value is composed of several subelements.

¹ "Root" = Top element of this address in the "Mapping" dialog

- Enter a certain variable value for an entry in the column **New value** and press <F7> to force or <Ctrl>+<F7> to write the value.

As for the monitoring in the declaration editor or in watch lists, the forced variable value is displayed in the column **Current value** with a preceding red F-symbol or the written value.



Inputs and outputs that the PLC code does **NOT** use are not read by the PLC in online mode. As a result, the displayed value can be incorrect.

The **current value** of the respective variables is displayed with a gray background.

Variable	Mapping	Channel	Address	Type	Default Value	Current Value	Prepared Value
Application.PlcProg.invar1		OnboardIn	%IB0	BYTE		3	
		1.1	%IX0-0	BOOL		TRUE	
		2.1	%IX0-1	BOOL		TRUE	
		1.4	%IX0-2	BOOL		FALSE	
		2.4	%IX0-3	BOOL		FALSE	
		1.1	%IX0-4	BOOL		FALSE	
		2.1	%IX0-5	BOOL		FALSE	
		1.4	%IX0-6	BOOL		FALSE	
		2.4	%IX0-7	BOOL		FALSE	
		OnboardOut	%QB0	BYTE		0	
		1.1	%QX0.0	BOOL		FALSE	
		2.1	%QX0.1	BOOL		FALSE	
		1.4	%QX0.2	BOOL		FALSE	

(1) Forced values on the control

(2) Values not used on the control, value shown in gray

Fig. 8-7:

I/O mapping in one dialog for multiple devices

There is a table that displays the I/O mapping of a device including the I/O mappings of all subelements inserted below it in the Project Explorer. The user can edit the I/O maps in exactly the same way as in the individual mapping tables of the respective device editors.

Prerequisite: In the Project Explorer of your project, there are several control devices inserted. Each allows one I/O mapping configuration.

1. Select the control in the Project Explorer and select the **Edit I/O mapping** command in the context menu.

The **Edit I/O mapping** dialog is displayed showing the I/O mapping configurations of all devices inserted into the project in a table. Edit the entries as in the **I/O mapping** dialog of the respective device editor.

2. Now select one of the control objects in the Project Explorer and select **Edit I/O mapping** again from the context menu.

The dialog **Edit I/O mapping** now shows only the I/O table for the I/O mapping configurations in and below the selected object.

3. Set a desired **filter** in the bar above the table or enter a variable name into the **Browse variable** field to see the use of this variable in the mapping.

Also refer to

- [chapter "Edit I/O mapping" command" on page 609](#)

9 Programming applications

To create an executable application program on the control, add declarations and implementation code (source code) to the POU and link the I/Os of the control to the application variables and configure a task assignment.

After checking and recovering the errors, the IndraLogic compiler creates the application code that can be loaded to the control.

Programming the application function blocks (POUs) is supported by the programming language editors and some other functionalities such as text lists, image pools, pragmas and the use of ready function blocks from the PLC programming system or from the libraries.

There are options to verify the syntax and to analyze code, to generate data persistence and to encrypt application code that is uploaded to the control.

9.1 Specifying identifiers

Identifiers are names of variables and programming objects (for example programs, function blocks, and methods) and names of other objects of the application and the project.

There are rules to be observed when assigning identifiers. Furthermore, there are also recommendations on how to find uniform and expressive identifiers.

Specify the variable IDs when declaring variables. These identifiers can be changed in the declaration section of the programming object.

Specify identifiers for programming objects and other objects in the dialog when adding the object. Change the identifier of an existing object of the application or of the project in the "Properties" dialog of the object.

However, the identifiers of objects that can be available only once per application or project cannot be changed (e.g. **library manager** and **ImagePool** identifiers).

Also refer to

- [chapter 17.7 "Identifier" on page 419](#)

9.2 Declaring variables

Variable declaration: Where and how

In IndraLogic projects, variables can be declared at the following locations:

- Declaration section of a POU
- Dialog **Declare variable** if the focus is on the POU
- DUT editor, GVL editor or NVL editor
- I/O mapping configuration of an I/O device object

The variable declarations in Global Variable Lists (GVL, NVL) and in the declaration section of POUs are made in the declaration editor.

Special editors or dialogs are available for all other variants.



Add pragmas to affect the behavior and the properties of one or more variables ([chapter 17.6 "Pragmas" on page 378](#)).

Syntax for the variable declaration in POUs or Global Variable Lists if textual display is selected for the declaration editor:

```
<Identifier> {AT <Address>}:<DataType> {:=<Initialization>;}
```

- **<Identifier>**: Variable name. The rules listed have to be followed without exception when assigning an identifier ([chapter 17.7 "Identifier" on page 419](#)). In addition, it contains recommendations for standardizing the names.
- **{AT Address}** (optional): A variable can be directly linked to a certain address using the keyword AT.
- **<DataType>**
- **{:=<Initialization>}** (optional)



In function blocks, variables with incomplete address information can also be declared.

To use such a variable in a local instance, there has to be a corresponding entry for this variable in the "Variable configuration".

If a variable is declared in the tabular declaration editor, the syntax is automatically correct.

Also refer to

- [chapter 9.2.1 "Using the declaration editor" on page 85](#)
- [chapter 17.5 "Data types" on page 351](#)
- [chapter 17.2 "Variables" on page 280](#)
- [chapter "Command 'Declare variable'" on page 588](#)
- [chapter 17.7 "Identifier" on page 419](#)
- [chapter 18.1.11 "'GVL" object - Global Variable List" on page 512](#)
- [chapter 9.6 "Using pragmas" on page 110](#)
- [chapter 9.2.2 "Using the dialog "Declare variable"" on page 86](#)

Variable initialization

The default initial value for all declarations is 0.

In the declaration section, user-defined initialization values can also be specified for each variable and each data type.

The user-defined initialization starts with the assignment operator := and consists of any valid expression of the programming language ST (Structured Text).

Thus, define the initial value using constants, other variables or functions.

When using a variable, initialize it as well (before)!

Examples

```
VAR
var1:INT := 12;           // initialization value 12

x: INT := 13 + 8;         // initialization value defined by an expression of constants*)

y: INT := x + fun(4);      (* initialization value defined by an expression,
                           that contains a function call;
                           notice the order! *)

z: POINTER TO INT := ADR(y); (* not described in the standard IEC61131-3:
                           initialization value defined by an address function;
                           Notice: the pointer will not be initialized
                           during an Online Change *)

END_VAR
```

Notes on the initialization sequence



From the compiler version 3.5.3.40, variables in a function block are initialized in the following order:

First, all constants acc. to the order of their declarations, then all other variables acc. to the order of their declarations.



From the compiler version 3.3.2.0, variables from Global Variable Lists are always initialized before the local variables of a POU.

Also refer to

- [chapter 17.5.14 "Data type 'ARRAY'" on page 363](#)
- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)
- [chapter 17.5 "Data types" on page 351](#)

9.2.1 Using the declaration editor

The declaration editor is used to declare variables in variable lists and POUs.

The declaration editor provides two possible views: textual  and tabular



In the dialog under **Tools ► Options ► IndraLogic 2G ► Declaration editor**, define whether only the textual view or only the tabular view is available or whether it can be toggled between both views using the buttons on the right side of the editor view.

If the declaration editor is used with a programming language editor, it is displayed as declaration section in the upper part of the window of a POU.

Also refer to

- [chapter "Command 'Declare variable'" on page 588](#)
- [chapter 9.2.2 "Using the dialog "Declare variable"" on page 86](#)
- [chapter ""Edit declaration header" command" on page 677](#)
- [""Input assistant" dialog, tab "Categories"" on page 587](#)

Declaring in the textual declaration editor

The response and design of the textual editor are configured using the settings in the dialog **Tools ► Options ► Text editor**. The settings concern colors, line numbers, tab widths, indentations etc. The usual Windows functions are available and the IntelliMouse functions as well if required.

Prerequisite: A programming object (POU, GVL or NVL) of a project is open. The focus is on the textual declaration editor.

1. Enter the variable declarations with a correct syntax. Press <F2> to open the **Input assistant** dialog to select the data type or a keyword.

Declaring in the tabular declaration editor

In the tabular declaration editor, insert variable declarations in a table with the following columns: **Scope**, **Name**, **Address**, **Data type**, **Initialization**, **Comment** and **Attributes** (pragmas).

Prerequisite: Open a programming object (POU or GVL) of a project. The focus is set on the tabular declaration editor.

1. Click on the  button in the declaration header or select the command **Insert** from the context menu.

IndraLogic inserts a new line for a variable declaration and the input field for the variable name opens.

2. Enter a valid variable identifier.
3. Double-click to open the other fields of the declaration line as required and select the specifications from the selection lists or use the dialogs.

Also refer to

- [chapter 18.3.3 ""Prepare value"" on page 701](#)
- ["Declaration editor in online mode" on page 223](#)
- ["Forcing in the declaration section" on page 183](#)
- [chapter "Command 'Declare variable'" on page 588](#)
- [chapter ""Edit declaration header" command" on page 677](#)
- [""Input assistant" dialog, tab "Categories"" on page 587](#)

9.2.2 Using the dialog "Declare variable"

Prerequisite: A programming object (POU or GVL) of a project is open.

1. Select **Edit ► Declare variable**.
The **Declare variable** dialog opens.
2. Select the scope for the variable from the selection list **Scope**.
3. Enter a variable name into the input field **Name**.
4. Select the data type from the selection list **Data type**.
5. If the initialization value deviates from the standard initialization value, enter an initialization value for the variable.
6. Complete your entries with **OK**.
IndraLogic lists the newly declared variable in the declaration section of your programming object.



Use pragmas in the declaration section to affect the processing of the declaration by the compiler.

Also refer to

- [chapter "Command 'Declare variable'" on page 588](#)
- [chapter 17.5 "Data types" on page 351](#)
- [chapter 9.6 "Using pragmas" on page 110](#)

9.2.3 Declaring arrays

Prerequisite: A programming object (POU or GVL) of a project is open.

1. Select **Edit ► Declare variable**.
The **Declare variable** dialog opens.
2. Select the scope for the array from the selection list **Scope**.
3. Enter an identifier for the array in the **Name** input field.
4. Click on  next to the **Data type** input field and select the **Array assistant** entry from the selection menu.
5. In the input fields **Dimension 1**, type in the lower and upper limit of the first dimension of the array, example: 1 and 3.

The field **Result** displays the first dimension of the array, e.g.: ARRAY [1..3] OF ?.

6. In the input field **Basic type:**, enter the data type of the array or use the



Input assistant or the **Array assistant**, e.g. DINT.

The field **Result** now also displays the data type of the array, e.g.: ARRAY [1..3] OF DINT.

7. Define the dimensions 2 and 3 of the array according to the steps 5 and 6, e.g.: Dimension 2: 1 and 4, dimension 3: 1 and 2.

The field **Result** displays the array with the defined dimensions: Array [1..3, 1..4, 1..2] OF DINT. The array consists of $3 * 4 * 2 = 24$ elements.



In an array of variable length, declare the dimension limits with an asterisk placeholder (*). Arrays of variable length are permitted to be used only in VAR_IN_OUT declarations of function blocks, methods, or functions.

Example of a two-dimensional array of variable length: aiUnknownLengthData : ARRAY [*,*] OF INT

8. Click on **OK**.

In the **Declare Variable** dialog, the **Data Type** field shows the array.

- 9.

To modify the initialization values of the array, click on  next to the **Initialization value** input field.

The **Initialization value** dialog opens.

10. Select the line of the array element whose initialization value is to be modified.

Example:

Select array component [1, 1, 1].

11. Enter the initialization value in the input field below the list and click on **Use value on selected lines**, e.g.: Value 4.

IndraLogic displays the changed initialization value of the selected line.

12. Click on **OK**.

In the **Initialization value** field of the **Declare variable** dialog, IndraLogic displays the initialization values of the array, e.g.:

[4, 23 (0)].

13. A **comment** can be entered optionally into the input field.

14. Click on **OK** to complete the declaration of the array.

IndraLogic adds the declaration of the array to the declaration section of the programming object.

Also refer to

- [chapter "Command 'Declare variable'" on page 588](#)
- [chapter 17.5.14 "Data type 'ARRAY'" on page 363](#)

9.2.4 Declaring global variables

Declaring global variables available within the application.

Prerequisite: A project is open.

1. In the Project Explorer of your project, select the application, in which the global variables should be valid.
2. Select **Add ► Global Variable List**.
IndraLogic inserts the **GVL** into the Project Explorer below the application and opens it in the editor.
3. Select **Edit ► Declare variable**.
The **Declare variable** dialog opens.
4. Select the entry **VAR_GLOBAL** in the selection list **Scope**.
5. In the field **Name**, enter a name for the global variable.
6. Select a data type from the selection list **Data type**.
7. If the initialization value of your variable should differ from the standard initialization value, click on  next to the field **Initialization value**.
The **Initialization value** dialog opens.
8. Double-click on the cell **Initialization value** of your variable and enter the desired valid value.
9. Click on **OK**.
The initialization value is displayed in the dialog **Declare variable**.
10. Enable one of the **flags** if required.
11. Press **OK** to confirm your entries.
IndraLogic inserts the declared variable into the GVL.
The global variable is available in the complete application of your project.

Declaring global variables available in the entire project.

1. Go to the "General module" folder.
2. Select the folder and the context menu command **Add ► Global Variable List**.
IndraLogic adds **GVL** to the folder.
3. Double-click on the object to open the editor.
4. Select **Edit ► Declare variable**.
The **Declare variable** dialog opens.
5. Select the entry **VAR_GLOBAL** in the selection list **Scope**.
6. In the field **Name**, enter a name for the global variable.
7. Select a data type from the selection list **Data type**.
8. If the initialization value of your variable should differ from the standard initialization value, enter it into the column **Initialization**.
9. Enable one of the **flags** if required.
10. Press **OK** to confirm your entries.
IndraLogic inserts the declared variable into the GVL.
The global variable is now available in the entire project.

Also refer to

- [chapter "Command 'Declare variable'" on page 588](#)

9.3 Generating source code in IEC

Source code:

The "source code" is the implementation code inserted into the programming function blocks using the corresponding programming language editors.

The following POU function block types are available:

- Program
- Function block
- Function
- Method
- Property
- Action
- Interface

Programming language:

Decide for an implementation language for programming when creating a programming function block. Apart from IEC languages, CFC is also available.

Programming language editors:

To edit a programming function block in the respective programming language editor, double-click on the object in the Project Explorer.

The function block is thus displayed either in a textual ST editor or in a graphical editor for FBD/LD/IL, SFC or CFC.

Each editor consists of two windows:

Make declarations in the upper window. Depending on the setting, declarations can either be made as text or table.

Insert the implementation code into the lower window. Representation and response of each editor can be configured project-comprehensively in the respective tab of the IndraLogic options.

Note that a programming function block can also be opened while the application is in online mode and the editor is offline.

Also refer to

- [chapter 17.1 "Programming languages and editors" on page 223](#)
- [chapter "'Edit object \(offline\)' command" on page 608](#)

9.3.1 FBD/LD/IL

A combined editor enables programming in the languages FBD (function block diagram), LD (ladder diagram) and IL (instruction list).

The basic unit of the FBD and LD programming is a network.

Each network contains a structure that can represent the following: A logical or arithmetic expression, the call of a programming function block (function, function block, program etc.), a jump or a return instruction.

IL actually does not require any networks. In IndraLogic, however, an IL program also consists of at least one network to support conversion into FBD or LD.

Thus, also divide an IL program into useful networks.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

Function block diagram (FBD)

The function block diagram is a graphically-oriented IEC 61131 programming language. It works with a list of networks. Each network contains a structure that can contain logical and arithmetic expressions, calls of function blocks, a jump or a return instruction.

Function blocks known from Boolean algebra are used. Function blocks and variables are connected by connecting lines. The signal flow in the network runs from left to right. The signal flow in the editor runs from top to bottom, starting with network 1.

Example:

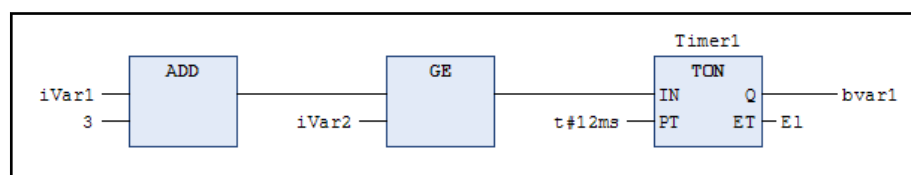


Fig. 9-1:



CFC is also a programming language based on the same principle as FBD, but with the following differences:

- The CFC editor is not network-oriented.
- The elements can be arbitrarily placed in the CFC editor.
- Direct insertion of feedbacks is possible.
- The order of execution is determined by a list of currently inserted elements, which can be changes.

Also refer to

- [chapter "Programming in Function Block Diagram \(FBD\)" on page 92](#)
- [chapter 18.2.9 "Menu 'FBD/LD/IL'" on page 661 \(Commands\)](#)
- [chapter 9.3.2 "Continuous Function Chart \(CFC\)" on page 96](#)

Ladder diagram (LD)

The ladder diagram (LD) is a graphically-oriented programming language that is close to an electrical circuit diagram. On one hand, the ladder diagram is suitable for constructing logical switches, but on the other hand, networks as in FBD can also be created.

Therefore, LD can be used very well for control calls of other program function blocks.

The ladder diagram consists of a series of networks.

A network is limited on the left by a vertical line (bus bar).

A network contains a circuit diagram of contacts, coils, optional function blocks (POUs) and connecting lines.

On the left side of a network, there is a contact or a series of contacts that transmit the ON or OFF state from left to right. This corresponds to the Boolean values `TRUE` and `FALSE`. Each contact has a Boolean variable. If this variable is `TRUE`, the status is transferred from left to right via the connection line. Otherwise, `OFF` is transferred. Thus the coil(s) in the right section of the network receive(s) the value `ON` and `OFF` from the left and the value `TRUE` or `FALSE` is written into the Boolean variable assigned to them.

If the elements are connected in series, this means an AND operation. If they are connected in parallel, this means an OR operation.

A line through an element means a negation of the element. The negation of an input or an output is indicated by a circle symbol.

Example:

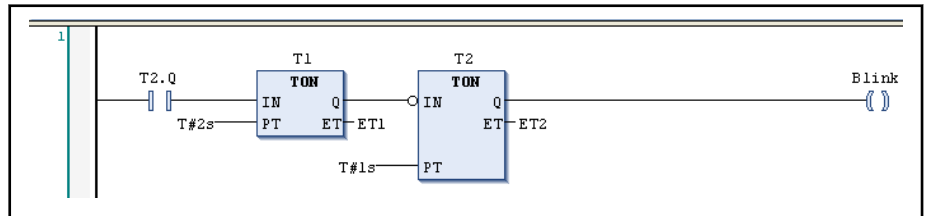


Fig. 9-2:

IEC 61131-3 defines a complete LD command set consisting of different types of contacts and coils.

Contacts conduct the current (according to their type) from left to right.

Coils store the incoming value.

Contacts and coils are assigned to Boolean variables.

Supplement an LD network by jumps, returns, labels and comments.

Also refer to

- [chapter "Programming in Ladder Diagram \(LD\)" on page 93](#)
- [chapter 18.2 "Menu commands" on page 572](#)

Instruction list (IL)

The instruction list is an assembler-like IEC 61131-compliant programming language. It supports accumulator-based programming.

An instruction list (IL) consists of a series of instructions.

Each instruction begins on a new line and includes an operator and, depending on the type of operation, one or multiple operands separated by commas. A mark, followed by a colon, can be placed in front of an instruction. It is used to identify the instruction. The label can be used as jump target. The last element in a line has to be a comment. Empty lines can be inserted between instructions.

All IEC 61131-3 operators are supported, as are multiple inputs, multiple outputs, negations, comments, set/reset of outputs and conditional/unconditional jumps.

Each instruction is based primarily on the loading of values to the accumulator (LD instruction). Then, the corresponding operation is executed with the parameter from the accumulator. The result of the operation is written again to the accumulator. Save it from there using an ST instruction.

The instruction list supports comparison operators (EQ, GT, LT, GE, LE, NE) and jumps for programming conditional executions or loops.

Jumps can be unconditional (JMP) or conditional (JMPC / JMPCN). In case of conditional jumps, it is checked whether the value is TRUE or FALSE in the accumulator.

Example:

1	LD	BVar1		
	ST	inst.IN	starts timer with rising e...	
	JMPC	mark1		
	CAL	inst(		
		PT:=t1,		is True
		ET:=>t2)		
	LD	inst.Q	gets TRUE delay time after...	
	ST	inst2.IN	starts timer with rising e...	
2	mark1:			
	LD	iVar2		
	ADD	230		

Fig. 9-3:

Also refer to

- [chapter "Programming in Instruction List \(IL\)" on page 95](#)
- [chapter "Modifiers and operators in IL" on page 259](#)
- [chapter 18.2.9 "Menu 'FBD/LD/IL'" on page 661](#)

Programming in Function Block Diagram (FBD)

Creating a POU in the Function Block Diagram (FBD) implementation language

1. Select an application in the Project Explorer.
2. Select **Add ► POU**.
The **Add POU** dialog opens.
3. Enter a name and select the **Function Block Diagram (FBD)** implementation language.
Click on **Finish**.
The POU is added to the Project Explorer and opened in the editor.
The editor contains the declaration editor in the upper part and the implementation section with an empty network in the lower part. The view **Tools** is also automatically opened, in which the suitable elements, operators and function blocks for FBD programming are available.

Also refer to

- [chapter 18.1.17 "Object "POU"" on page 520](#)
- [chapter 9.3.1 "FBD/LD/IL" on page 89](#)

Programming a network

1. Click into the automatically inserted empty network in the implementation section.
The network is highlighted in yellow and the area with the network number at the left is highlighted in red.
2. Right-click to open the context menu.
Among other things, the insert commands for the elements that can be inserted at this positions, is provided.

3. Insert the elements required for programming using the menu commands or by dragging in the elements from the toolbox.
4. Select the command **Insert assignment** for example.
An assignment line is inserted. In each case, three question marks stand for assignment source and assignment target.
5. Select the question marks and replace them with the variable. The input assistant is available.
6. Move the cursor over the assignment line.
The possible insertion positions for further elements are displayed as grey diamonds. A click on a diamond selects that position and the suitable insertion commands are available.
7. Alternatively, drag an element from the toolbox and drop it onto the network. For example, click in the toolbox on the function block element, keep the mouse button pressed and move the cursor over the network.
Each possible insertion position lights up green.
8. Release the mouse button to insert the function block.
The function block box is displayed in the network. The type of function block in the inside and the instance name above the function block required for a function block, remain free with three question marks.
9. Select the character string ??? inside the function block and replace it with the function block name. The input assistant is available.
The inputs and outputs of the selected function block are displayed. They are still saved with question marks (the instance name for function blocks as well).

Also refer to

- [chapter 18.2.9 "Menu 'FBD/LD/IL'" on page 661](#)
- [chapter 9.3.1 "FBD/LD/IL" on page 89](#)

Programming line branchings (subnetworks)

1. In the implementation section of your POU, insert a new network using the command **FBD/LD/IL ► Insert network** or drag it in from the toolbox.
2. For example, drag an **ADD** operator into the empty network and replace the characters ??? with two variables of the data type **INT**.
3. Drag the element **Line branching** from the toolbox into your implementation and release the mouse button at the green insertion position directly at the output of the operator.
The line branching splits the processing line at the output of the operator function block into 2 subnetworks.
4. Further FBD elements and also further line branchings can now be added to each of the two subnetworks.

Also refer to

- [chapter "FBD/LD/IL element "Line branching"" on page 265](#)
- [chapter ""Insert line branching" command" on page 668](#)

Programming in Ladder Diagram (LD)

Creating a POU in the Ladder Diagram (LD) implementation language

1. Select the application in the Project Explorer.

2. Select **Add ► POU**.
The **Add POU** dialog opens.
3. Enter a name and select the **Ladder Diagram (LD)** implementation language.
Click on **Finish**.
IndraLogic adds the POU to the Project Explorer and opens it in the editor. An empty network is inserted into the implementation section. The empty network is bound on the left by a vertical line, which represents a bus bar. The view **Tools** is also automatically opened, in which the suitable elements, operators and function blocks for LD programming are available.

Adding a contact and a function block (TON)

Prerequisite: A POU with the implementation language LD is open in the editor and an empty network is inserted.

1. Click on the category **Ladder diagram elements** in the view **Tools**
2. Click on the **Contact** element, drag it into your network and release the mouse button at the insertion position **Start here**.
The contact is added on the left in the network directly against the vertical line.
3. Click on **???** and enter the identifier of a Boolean variable. The input assistant is also available.
4. Click on the category **Function blocks** in the **Toolbox** view and drag the function block **TON** onto an insertion position on the connecting line to the right of the inserted contact.
IndraLogic inserts the box **TON** to the right of the contact. The contact is connected to the input **IN** of the TON function block.
5. Enter a time constant at the input **PT**, e.g. **T#3s**.
If the variable of your contact becomes **TRUE**, the input **IN** of the TON function block becomes **TRUE** as well. The TON function block forwards **TRUE** to the output **Q** with a switch-on delay of **T#3s** for example.

Inserting a closed line branching

Prerequisite: A POU with the implementation language LD is open in the editor and an empty network is inserted.

1. Click into the empty network and select the command **FBD/LD/IL ► Insert contact**.
2. Select the connecting line to the left of the contact and select the command **FBD/LD/IL ► Set branch start point**.
The starting point on the connecting line is marked by a red rectangle. IndraLogic marks all possible end points of the branching with a blue rectangle.
3. Click on a blue rectangle to set the end point of your closed line branching.
IndraLogic inserts the line branching between the starting and end points. The program flow runs through both branches up to the end. When inserting the line branch at a function block instead of at a contact, the function block is only called if none of the other branchings is **TRUE**.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "Closed line branching" on page 268](#)
- [chapter "LD element "Contact"" on page 266](#)

Programming in Instruction List (IL)



Enable the IL in the IndraWorks options if required.

Tools ► Options ► IndraLogic 2G ► FBD, LD and IL ► IL ► Enable IL

Creating a POU in the Instruction List (IL) implementation language

1. Select the application in the Project Explorer.
2. Select **Add ► POU**.
The **Add POU** dialog opens.
3. Enter a name and select the **Instruction list (IL)** implementation language.
Click on **Finish**.
IndraLogic adds the POU to the Project Explorer and opens it in the editor. A network is already inserted into the implementation section.

Programming a network (for example an ADD operation)

Prerequisite: A POU (IL) is open in the editor and has an empty network.

1. Click on the line highlighted in color in the first column and enter the operator **LD**.
2. Press <Tab>
The cursor jumps to the second column
3. Enter the first summand of your ADD operation, e.g. 6
4. Press <Ctrl>+<Enter> or select **FBD/LD/IL ► Insert IL line after**
IndraLogic inserts a new instruction line below. The focus is in the first column of this line.
5. Enter **ADD** and press <Tab>.
6. Enter the second summand of your ADD operation, e.g. 12.
7. Press <Ctrl>+<Enter>
8. Enter the operator **ST** and press <Tab>.
9. Specify a variable of the data type **INT**, e.g. **iVar**.
The result - 16 in the example - is stored in **iVar**.

Calling a function block

Prerequisite: A POU (IL) is open in the editor and has an empty network. A variable of the data type <FunctionBlock> is declared in the declaration section, e.g. C1:CTU;

1. Click on the line highlighted in color in the first column and select the command **FBD/LD/IL ► Insert function block**.
The input assistant opens.
2. Select the function block in the category **Function blocks** or the category **Function block calls**, e.g. the **CTU** counter from the **Standard** library and click on **OK**.

→ IndraLogic inserts the selected function block **CTU** as follows:

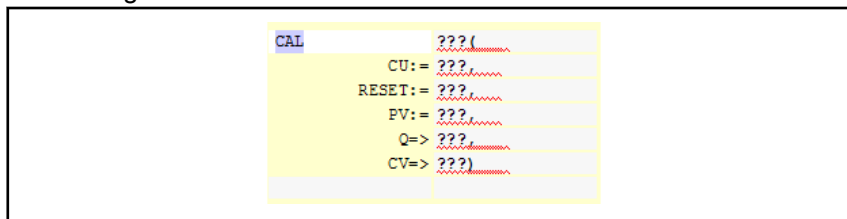


Fig. 9-4:

3. Replace the strings ??? with the variable name and the values or variables for the inputs/outputs of the function block.
4. As an alternative to inserting the function block via the input assistant, enter the call directly into the editor.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "Modifiers and operators in IL" on page 259](#)

9.3.2 Continuous Function Chart (CFC)

The "Continuous Function Chart (CFC)" implementation language is a graphical programming language which extends the standard languages of IEC 61131-3.

The graphic programming in "CFC" is similar to the programming in "FBD", but provides important additional functions:

- Inserted elements can be positioned freely (e.g. function blocks)
- Elements can be interconnected to a network using graphical connections
- Feedback paths can be inserted between the elements
- Option for site arrangement. Complex programs can be distributed on multiple sites.

One CFC advantage is that the function plan (program) can be read like a circuit diagram or a block diagram.

The execution sequence of a CFC function plan is based on data flow. One programming function block can process multiple data flows. In this case, data flows do not have any common data. In the editor, there can be multiple networks without being interconnected.

On the other hand, POU's in FBD, LD, or IL have a network-based execution order.

The "Continuous Function Chart (CFC) - page-oriented" implementation language is also a graphical programming language which extends the standard languages of IEC 61131-3. "CFC - site-oriented" provides advantages in the graphic programming of complex and large function plans by arranging the networks on any number of sites.

On the respective sites, the same elements and commands are available as for Continuous Function Chart (CFC). Each site has two additional border areas (left/right):

- Inputs and target connection marks can be arranged in the left border area
- Inputs and source connection marks can be arranged in the right border area

The border areas support when inserting connecting lines and provide for clearer arrangement.



It is not possible to switch a programming function block between the Continuous Function Chart (CFC) - page-oriented and Continuous Function Chart (CFC) implementation languages.

Also refer to

- [chapter "CFC editor" on page 270](#)
- [chapter "CFC editor in online mode" on page 273](#)
- [chapter "Page-oriented CFC editor" on page 272](#)

Programming in the CFC editor

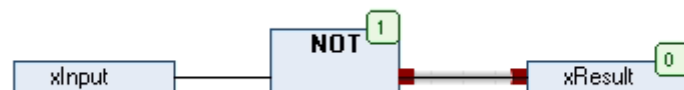
In the CFC editor, you can wire programming function blocks to each other and create well-structured block diagrams.

The editor supports the following functions:

- Programming with elements and connecting lines
- Dragging instances and variables to the editing area
- Auto-routing the connecting lines
- Automatic linking
- Fixing of connecting lines by control points
- Collision detection
- Input assistance for connection marks
- Forcing and writing of values in online mode
- Movement of selection using arrow keys
- Reduced display of a block without disconnected pins

Inserting elements and wiring with connecting lines

1. Place a **Function block** element and an **Output** element in the editor.
2. Select the output of the **function block** element.
The output is labeled with a red square.
3. Drag a connecting line from the box output of the **Box** element to the box input of the **Output** element.
The cursor symbol changes when it reaches the box input.
4. Release the left mouse button.
The output pin of the box is wired to the input pin of the output.



You can also hold down the <Ctrl> key, select each pin, and then right-click **Connected Selected Pins**.

Creating connection marks

Prerequisite: A CFC function block with connected elements.

1. Select a connecting line between two elements.

→ The connecting line is displayed as selected. The ends of the connecting line are marked with red boxes (■).

2. Click on **CFC ► Connection mark**.

The connection is separated and replaced by a "source connection mark" and a "target connection mark".
The name of the two marks is generated automatically, but can subsequently be changed.

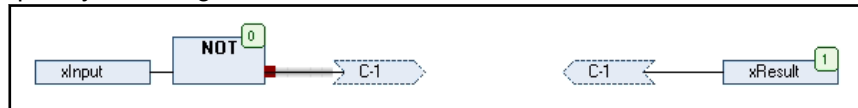


Fig. 9-5:

3. To change the name, click on the source connection mark.

→ The name is selected and can be edited.

4. Specify a name, e.g. SimpleMark for the source connection mark.

→ Source connection marks and target connection marks obtain the changed name automatically.

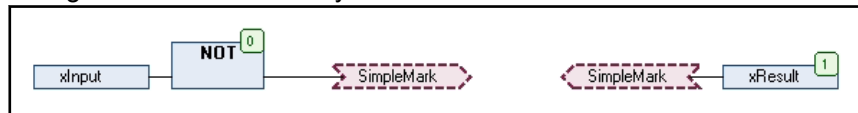


Fig. 9-6:

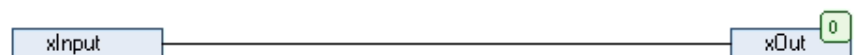
Also refer to

- [chapter ""Connection mark" command" on page 657](#)

Resolving collisions and fixedly setting connecting lines using control points

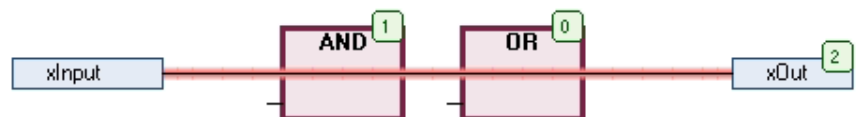
The following example shows how to use the **"Route All Connections"** command with control points.

1. Position the **Input** and **Output** elements as seen below and connect the elements



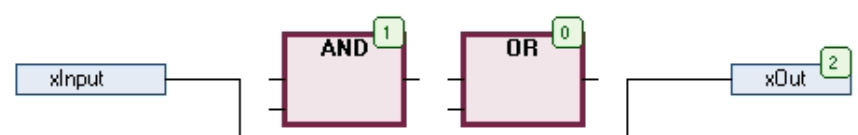
2. Position two **"Box"** elements on the line.

→ The connecting line and the function blocks are highlighted in red due to the collision.

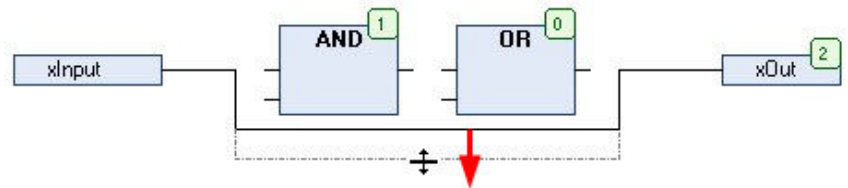


3. Click **CFC ► Routing ► Route All Connections**.

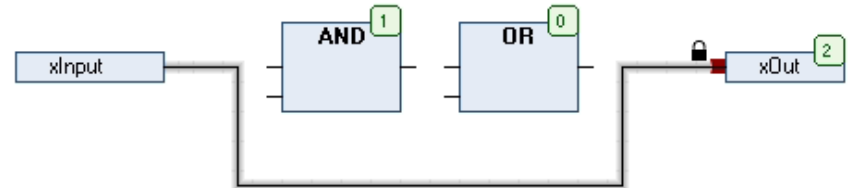
→ The collision is resolved



4. Offset the position of the connecting line.



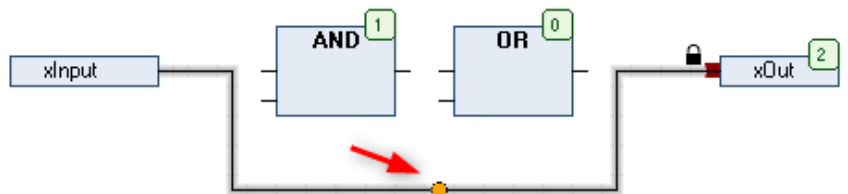
→ The connecting line was changed manually to block IndraLogic for automatic routing. This is shown by a lock at the end of the connection.



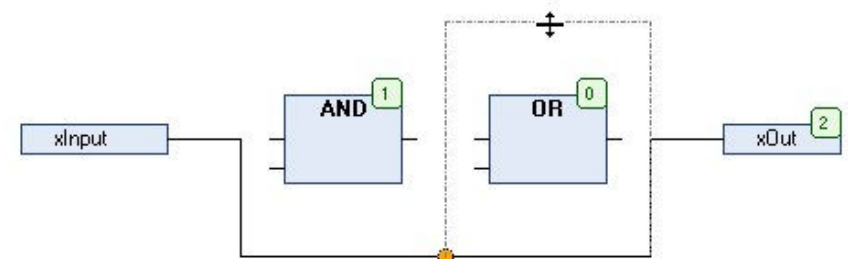
5. Place a control point on the connecting line.

Select the connecting line where the control point is to be set and select **CFC ▶ Routing ▶ Create control point**

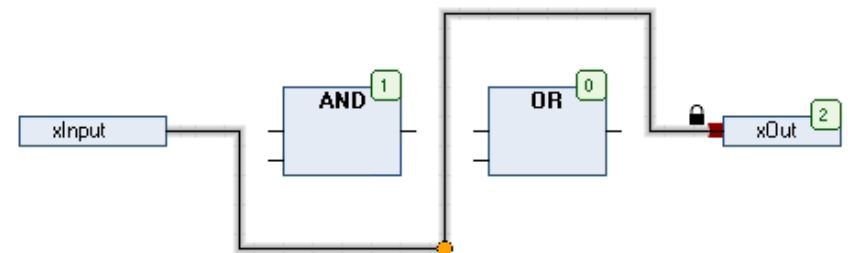
→ A control point is created on the connecting line at the selected position. The connecting line is fixedly set at the control point position in the editor.



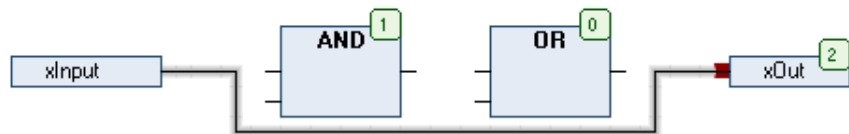
6. Change the connection as in the following example



Use the control point for changing the connecting line according. One connecting line can be fixedly set with multiple control points.



7. Click on **CFC ▶ Routing ▶ Remove Control Point** to remove the control point.
Control points can only be deleted individually.
8. To unlock the connection, click on **CFC ▶ Routing ▶ Unlock connection** or click on the lock symbol.
9. Select the connecting line and click on **CFC ▶ Routing ▶ Route all connections**.
→ The connecting line is routed automatically.



Connections in a group are not auto-routed.

Also refer to

- [chapter "Remove control point" command](#) on page 657
- [chapter "Unlock connection" command](#) on page 655
- [chapter "Route all connections" command](#) on page 656

Reducing the display of a function block

Prerequisite: A CFC programming function block is open. In the editor, its blocks with all declared pins are displayed.

1. Select a block whose pins are partially disconnected.

Example:



→ The function block used unnecessary depiction space for the unconnected connections.

2. To hide unconnected connections, select **CFC ▶ Pins ▶ Remove unused pins**

→ The function block requires less space and it is displayed only with the connected connections:



Also refer to

- [chapter 17.1.2 "Common functionalities in graphical editors"](#) on page 224
- [chapter "CFC editor"](#) on page 270
- [chapter "Page-oriented CFC editor"](#) on page 272
- [chapter "Elements"](#) on page 277

9.3.3 Structured Text (ST) and Extended Structured Text (ExST)

The ST editor is used to program POU's in the IEC-61131-3 programming languages Structured Text (ST) and Extended Structured Text. The Extended Structured Text provides some additional functions with regard to the IEC 61131-3 standard.

Structured text is a programming language that is comparable with other high level languages such as C or PASCAL which allows complex algorithms to be developed.

The program code consists of a combination of **expressions** and **instructions**, which can also be executed conditionally (IF... THEN... ELSE) or in loops (WHILE... DO).

An expression is a construct that returns a value after being evaluated. Expressions are also operators and operands together. Assignments can also be used as expressions. An operand can be a constant, a variable, a function call or another expression.

Instructions control how to process the expressions.

For this text editor, make various settings with regard to response, appearance and menus in the dialogs **Options** and **Adapt** in the **Tools** menu. The familiar Windows functions (for example IntelliMouse) are also available for this editor.

Also refer to

- [chapter "Programming in Structured Text \(ST\)" on page 101](#)
- [chapter ""Options" dialog - Text editor" on page 731](#)
- Dialog "Customize" - "Menu"

ExST - Extended Structured Text

Extended Structured Text (ExST) is an IndraLogic-specific extension with regard to the IEC 61131-3 standards for Structured Text (ST).

Also refer to

- [chapter "ExST assignment 'R='" on page 228](#)
- [chapter "ExST assignment 'S='" on page 228](#)
- [chapter "ExST assignment " := " as expression" on page 229](#)

Programming in Structured Text (ST)

Principle The programming languages "Structured Text" and "Extended Structured Text" are programmed in the ST editor.

The program code consists of a combination of expressions and instructions, which can also be executed conditionally or in loops.

Complete each instruction with a semicolon ; .

The variables are declared in the declaration editor.

Creating a POU in the Structured Text (ST) implementation language

1. Select an application in the Project Explorer.
2. Select **Add ► POU**.
The **Add object wizard** dialog opens.
3. Enter a name and select the **Structured Text (ST) implementation language**. Click on **Finish**.

The POU is added to the Project Explorer and opened in the editor. Now insert the variable declarations in the upper part of the POU and enter the ST program code in the lower part of the POU.

Also refer to

- [chapter "ST expressions" on page 226](#)
- [chapter 17.1.3 "Structured Text and Extended Structured Text \(ExST\)" on page 225](#)
- [chapter "ST - Calling function blocks" on page 235](#)
- [chapter "ST - Comments" on page 235](#)
- [chapter 9.2.1 "Using the declaration editor" on page 85](#)

9.3.4 Sequential Function Chart (SFC)

Use the SFC editor to program POU in the IEC 61131-3-compliant "SFC implementation" language (Sequential Function Chart, SFC).

Sequential function chart (SFC) is a graphically-oriented language to describe the chronological sequence of individual actions in a program. For this purpose, actions that are discrete programming objects, are assigned to the "step" elements. The processing sequence of the steps is controlled by "transition" elements.

Also refer to

- [chapter "SFC \(sequential function chart\) editor" on page 236](#)

Programming in Sequential Function Chart (SFC)

Creating a POU in the SFC implementation language

1. Select an application in the Project Explorer.
2. Select **Add ► POU**.
The "Add POU" dialog opens
3. Enter a name and select the **Sequential Function Chart (SFC)** implementation language.
Click on **Finish**.
IndraLogic adds the POU to the Project Explorer and opens it in the editor.

Adding step-transitions

1. Select the transition after the Init step.
The transition is highlighted in red.
2. Select **SFC ► Insert step-transition after**.
IndraLogic inserts the step **Step0** and the transition **Trans0**.
3. Select the **Trans0** transition and click on **SFC ► Insert step-transition**.
IndraLogic inserts the **Trans1** transition and the **Step1** step in front of the **Trans0** transition.

The **Step** and **Transition** elements can also be dragged from the **Tools** view into the diagram.

Also refer to

- [chapter "SFC elements "steps" and "transition"" on page 245](#)
- [chapter ""Insert step-transition" command" on page 642](#)
- [chapter ""Insert step-transition after" command" on page 643](#)

Adding entry actions

1. Select **Step0**.
2. Select **SFC ► Add input action**.

Normally, an input prompt is shown to determine the duplication mode for the step actions. This specifies whether the reference to the existing step action objects is copied when copying the step or whether the objects are "embedded". If they are embedded, new step action objects have to be created when copying the step. The duplication mode is defined in the step property "Duplicate when copying". As long as it is disabled, the copied steps call the same actions as the current step.

Displaying the input prompt can be generally disabled in the dialog or in the IndraWorks options.

Refer to: ["Prompt dialog to select the duplication mode" on page 106](#)

3. For this example, keep the default setting **"Copy reference..."** and press **"OK"**.

The "Add input action" dialog opens.

4. Enter the name **"Step0_entry"** and select **"ST"** as implementation language. Click on **Finish**.

IndraLogic inserts the **"Step0_entry"** action below the POU in the Project Explorer and opens the action in the editor.

In the **"Step0_entry"** entry action, program instructions to be executed only once when the **Step0** step is enabled.

5. Close the **"Step0_entry"** editor.

The step **Step0** is now marked with an **E** in the lower left corner. Double-click on this marking to open the GNVL editor.



Fig. 9-7:

The entry action **Step0_entry** is now located in the step properties below **"Entry action"**. If required, another action can also be selected.

6. Select and copy Step0. Press [Ctrl]+[V].

The inserted copy of the step is the same entry action inserted above. The new step calls the same action.

Also refer to

- ["Step actions" on page 248](#)

Adding exit actions

1. Select **Step0**.
2. Select **SFC ► Add output action**.

Normally, an input prompt is shown to determine the duplication mode for the step actions of the step as described under "Adding entry actions". Then, the "Add output action" dialog opens.

Refer to: ["Prompt dialog to select the duplication mode" on page 106](#)

3. Enter the name **"Step0_exit"** and select **"Structured Text "ST" as implementation language**. Click on **Finish**.

IndraLogic inserts the **"Step0_exit"** action below the POU in the Project Explorer and opens the action in the editor.

In the **"Step0_exit"** exit action, program instructions to be executed only once before the **Step0** step is disabled.

4. Close the "Step0_exit" editor.

The step **Step0** is now marked with an **X** in the lower right corner. Double-click on this marking to open the GNVL editor.



Fig. 9-8:

Define the exit action in the step properties of **Exit action**. If required, another action can also be selected.

Also refer to

- ["Step actions" on page 248](#)

Adding actions

1. Double-click on the **Step0** step.

Normally, an input prompt is shown to determine the duplication mode for the step actions of the step as described under "Adding entry actions". The **Add action** dialog opens.

Refer to: ["Prompt dialog to select the duplication mode" on page 106](#)

2. Enter the name "Step0_active" and select the implementation language "Structured Text "ST"". Click on **Finish**.

IndraLogic inserts the "*Step0_active*" action below the POU in the Project Explorer and opens the action in the editor.

In the *Step0_active* step action, program instructions to be executed as long as the step is active.

3. Close the "Step0_active" editor.

The **Step0** step is now marked with a black triangle in the upper right corner.



Fig. 9-9:

Define the action in the step properties of **Step action**. If required, another action can also be selected.

Also refer to

- ["Step actions" on page 248](#)

Adding an alternative branching

1. Select **Step1**.
2. Select **SFC ► Insert branching right**.

IndraLogic inserts the step **Step2** to the right of **Step1**. The steps are linked with a double line as parallel branching.

3. Select one of the two double lines.
The double line is shown in red.

4. Select **SFC ► Alternative**

This command converts the selected parallel branching into an alternative branching. The double line becomes a single line.

To convert an alternative branching into a parallel branching, click on **SFC ► Parallel**.

Also refer to

- [chapter "SFC element "Branching"" on page 249](#)
- [chapter ""Parallel" command" on page 643](#)
- [chapter ""Insert branch" command" on page 644](#)
- [chapter ""Insert branch right" command" on page 644](#)

Adding jumps

1. Select **Step2**.
2. Select **SFC ► Insert jump after**.
IndraLogic inserts the jump **Step** after the **Step2** step.
3. Select the "Step" jump target of the jump

Enter the jump target either manually or use the input assistant . Select "Step0".

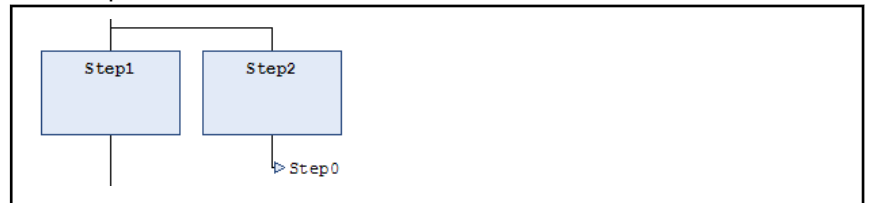


Fig. 9-10:

Also refer to

- [chapter "SFC element "Jump"" on page 251](#)
- [chapter ""Insert jump" command" on page 646](#)
- [chapter ""Insert jump after" command" on page 647](#)

Adding macros

1. Select **Step1**.
2. Select **SFC ► Insert macro after**.
IndraLogic inserts the macro **Macro0** after the step **Step1**.
3. Double-click on the **Macro0** element.
The macro opens in the implementation section of the editor. "Macro0" stands in the header.
4. Select **SFC ► Insert step-transition**.
IndraLogic inserts a step-transition combination.
5. Select **SFC ► Exit macro**.
Der implementation section shows the main diagram again.

Also refer to

- [chapter "SFC element "Macro"" on page 251](#)
- [chapter ""Insert macro" command" on page 647](#)
- [chapter ""Insert macro after" command" on page 647](#)

Adding associations

1. Select **Step2**.
2. Select **SFC ► Insert association**.
IndraLogic inserts an association to the right of the **Step2** step.

3. Click into the left field of the association to select the qualifier.
Enter the qualifier manually or select it via the input assistant . Select "P".
4. Click into the right field of the association to select the action.
The action can be entered manually or can be selected via the input assistant .

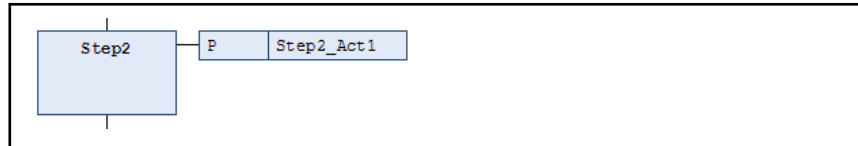


Fig. 9-11:

Also refer to

- ["IEC actions" on page 247](#)
- [chapter "Qualifiers for SFC actions" on page 239](#)
- [chapter ""Insert action association" command" on page 645](#)
- [chapter ""Insert action association after" command" on page 646](#)

Also refer to

- [chapter "SFC \(sequential function chart\) editor" on page 236](#)

Prompt dialog to select the duplication mode

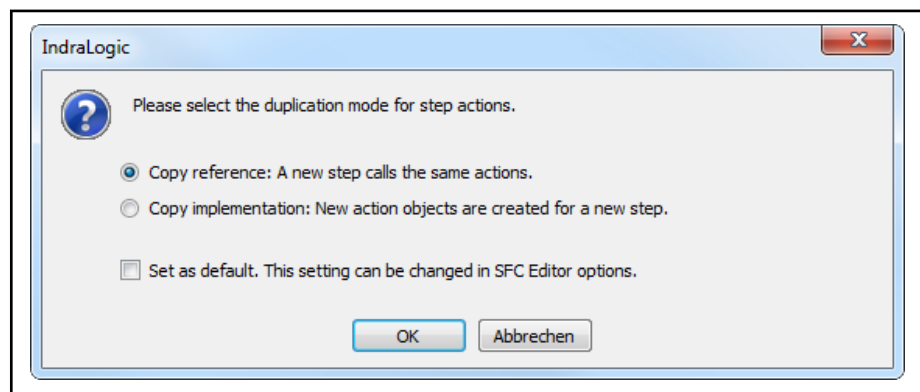


Fig. 9-12:

Options:

- *"Copy reference. A new step calls the same actions"*: If the step is copied in SFC, the link to the step actions is also copied. Steps copied from each other call the same actions.
- *"Copy implementation. New action objects are created for a new step"*: This means that the step actions for the copied steps are "embedded". The newly created action objects are displayed by default below the SFC function block in the "Application" node or in the "General module folder". At first, these objects contain a copy of the original implementation code of the respective action.

"Set as default": The selected object (standard insertion method) is set by default when adding a new entry/exit action. The prompt dialog is not displayed anymore.

The standard insertion method can be changed in the IndraWorks options:
Tools ► Options ► IndraLogic 2G ► SFC editor

9.4 Calling function block, function or method with external implementation

A runtime system can include the implementation of a function block, function or method, e.g. from a library. When creating a POU in your application with the same name with the **External implementation** property without an implementation, the existing implementation can already be executed.



Ensure that local variables are only declared in an external function block.

External functions and methods must not contain local variables.

Create POU with external implementation

When the application is downloading, IndraLogic searches for the associated implementation in the runtime system for each external POU and links it.

1. For your application (or the "General module folder"), select **Add ► POU** from the context menu.
2. Enable the **function block** or the **function** and enter the name of the respective implementation of the runtime system as name. Click on **Finish** to complete the dialog

The POU with the name of the runtime system POU is created in the Project Explorer.

3. Select the POU and enable **Properties** in the context menu.
The dialog opens.
4. Click on the **Build** tab.
5. Select the **External implementation (late linking in the runtime system)** option.

The POU is declared and a POU call can be implemented.

Create method with external implementation

1. Select a function block in the Project Explorer.
2. Select **Add ► POU method** and enter the name of the associated implementation of the runtime system. Press **Finish** to close the dialog.

The method is created.

3. Select the method and enable **Properties** in the context menu.
The dialog opens.
4. Click on the **Build** tab.
5. Select the **External implementation (late linking in the runtime system)** option.

The method is declared and a method call can be implemented.

9.5 Using input assistants

IndraLogic provides functionalities and assistants facilitating code entering while programming.

Input assistant

The input assistant provides all program elements that can be inserted at the current cursor position. To open the **Input assistant** dialog, click on **Edit ► Input assistant** or press <F2>.

- Also refer to*
- [""Input assistant"" dialog, tab ""Categories"" on page 587](#)
- "Declare variable" dialog** This dialog supports the declaration of variables.
- Also refer to*
- [chapter "Command 'Declare variable'" on page 588](#)
- "List components"** The "List components" function is an input assistant in textual editors to help you entering valid identifiers.
- Enable the function as follows:
- Select **Tools** ► **Options** ► **IndraLogic 2G** and then the **Smart coding** category. Select the **List components after typing a dot (.)** option.
- When entering a dot (.) instead of a global variable, a selection list opens with all available global variables. Double-click on a variable in the selection list or press <Enter> to insert the selected variable after the dot .
 - When entering a dot (.) instead of a global variable or behind a function block instance variable or a structural variable, IndraLogic opens a selection list with all global variables, all input and output variables of the function block or all structural components.
Double-click on a variable in the selection list or press <Enter> to insert the selected variable after the dot .
Note: To additionally obtain the local variables of function block instances, select the **Show all instance variables in input assistant** option in the IndraLogic options for smart coding.
 - If a component access (with a dot) for a list box has already happened, then the last selected entry is preselected at the next component access.
 - When entering any sequence of characters and then pressing <Ctrl> +<Space>, a selection list opens with all available POU's and global variables. The first element in this list that starts with the sequence of characters is selected by default. To insert it, double-click or press <Enter>.
Matches with the entered character string are highlighted in yellow in the list box. If the entered character string is changed, then the displayed list box is refreshed.
 - In the ST editor, you can filter the displayed list box by scopes:
Depending on the displayed list box, you can use the [Arrow right] and [Arrow left] keys to toggle between the following list boxes:
 - "All items"
 - "Keywords"
 - "Global declarations"
 - "Local declarations"
 - IndraLogic displays a tooltip when calling a function block, a method or a function and when entering an opening parenthesis to enter a POU parameter. This tooltip includes information on the parameters as they are declared in the POU. The tooltip remains visible until it is closed via mouse-click or by setting the focus outside the current view. If a tooltip is closed accidentally, open it again with <Ctrl>+<Shift>+<Space>.



The 'hide' pragma attribute can be used to exclude variables from the "list components" function.

Example:

Entering a structural variable:

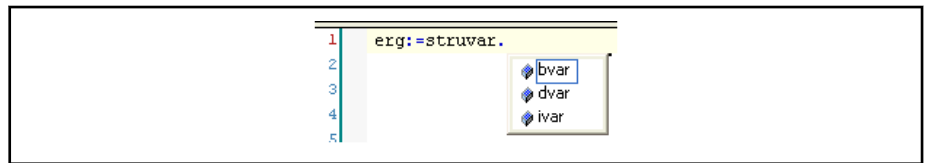


Fig. 9-13:

Calling a function block:

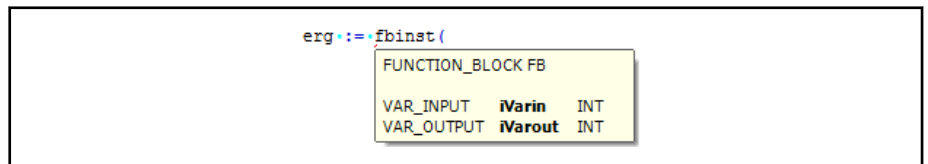


Fig. 9-14:

Also refer to

- [chapter ""Options"" dialog - "Smart coding "" on page 730](#)
- [chapter "Attribute 'hide'" on page 389](#)

Short form mode

The short form mode allows to enter abbreviated forms for variable declarations in the declaration editor and in textual editors, in which variable can be declared. To enable this mode, press <Ctrl>+<Enter> to end a declaration line.

IndraLogic supports the following short forms:

- All identifiers except the last identifier in a line become variable identifiers of a declaration.
- The declaration data type is specified by the last identifier in the line. The following applies:
 - B or BOOL results in BOOL
 - I or INT results in INT
 - R or REAL results in REAL
 - S or STRING results in STRING
- If no data type is defined using these rules, the data type is automatically BOOL and the last identifier is not used as data type (refer to example 1).
- Depending on the type of declaration, every entered constant becomes an initialization or a string length definition (refer to the examples 2 and 3)
- An address, such as %MD12, is automatically extended with the AT attribute (see example 4)
- Any text after a semicolon (;) is converted into a comment (see Example 3)
- All other characters in the line are ignored (see exclamation mark in example 5)

Example:

Example	Short form	Resulting declaration
1	A	A: BOOL;
2	A B I 2	A, B: INT := 2;
3	ST S 2; A string	ST: STRING(2); (* A string *)
4	X %MD12 R 5 Real Number	X AT %MD12: REAL := 5.0; (* Real Number *)
5	B !	B: BOOL;

Also refer to

- [chapter 9 "Programming applications" on page 83](#)

Smart tag functions

Smart tags make it easier to write program code by suggest appropriate commands directly at the programming element. When you place the cursor over a programming element that has a smart tag function, the  symbol appears. When you click the  symbol, the commands that you can choose from are shown.

Available smart tags:

- The smart tag function provides the "AutoDeclare" command for undeclared variables in the implementation part of the ST editor.

Also refer to [chapter "Command 'Declare variable'" on page 588](#)

9.6 Using pragmas

What is a pragma in IndraLogic?

A pragma is a special instruction in the source code of an application. This instruction affects the properties of one or more variables with regard to pre-compilation or compilation (generation of code).

There are pragmas for different effects:

Initialization of a variable, monitoring of a variable, adding a variable to a parameter list or to a symbol configuration, forcing the message output during the compilation, response of a variable under certain conditions, etc.



Observe upper and lower case!

How and where to insert a pragma

A pragma instruction is specified in curly parentheses.

Attributes and texts used within the instruction are enclosed by inverted commas.

The opening bracket may immediately follow a variable name. Opening and closing brackets have to be in the same line.

Examples

```
{warning 'This is not allowed'}
{attribute 'obsolete' := 'datatype fb1 not valid!'}
```

Possible insertion positions:

- In the declaration section of a POU
Enter pragmas directly as a line into the textual declaration editor.
In the tabular editor, enter pragmas that must be located above the first declaration line into the dialog **Edit declaration header / Attributes**.

- In a Global Variable List
- In the implementation section of a POU in a separate line or in a line together with code text

FBD/LD/IL: Pragmas in networks of the FBD/LD/IL editor are entered like a jump label:

Select the command **FBD/LD/IL ► Insert label** and replace the standard text **Label:** in the text field of the label with the appropriate pragma instruction.

To use a pragma in addition to a label, enter the pragma first and then the jump label.



Pragmas in IndraLogic are no one-to-one implementations of the C-preprocessor directives. They have to position a pragma like a normal statement. They may not use it within an expression.

Incorrect and correct positioning of a conditional pragma

Wrong:

```
{IF defined(abc)}
IF x = abc THEN
{ELSE}
IF x = 12 THEN
{END_IF}
y := {IF defined(cde)} 12; {ELSE} 13; {END_IF}
END_IF
```

Right:

```
{IF defined(abc)}
IF x = abc THEN
{IF defined(cde)}
  y := 12;
{ELSE}
  y := 13;
{END_IF}
END_IF
{ELSE}
IF x = 12 THEN
{IF defined(cde)}
  y := 12;
{ELSE}
  y := 13;
{END_IF}
END_IF
{END_IF}
```



In the **Properties** dialog, category **Compile**, defines, which can be queried in pragmas, can be specified.

Scope:

Depending on the type and contents of a pragma, it affects:

- The subsequent declarations
- Precisely the subsequent instruction
- All subsequent instructions until it is cancelled again by an appropriate pragma
- All subsequent instructions until the same pragma is executed with other parameters or the end of the code is reached.

"Code" in this context means: Declaration section, implementation section, Global Variable List, type declaration.

A pragma thus affects the entire object if it is located alone in the first line of the declaration section and if it is not replaced or cancelled by another pragma.

What kinds of pragmas are there?

The IndraLogic pragmas are divided into the following categories:

- **Attribute pragmas:** Impact on the compilation and the pre-compilation
- **Message pragmas:** Output of user-defined messages during the compilation
- **Conditional pragmas:** Affecting the code generation
- **User-defined pragmas**

Also refer to

- [chapter 9.2.2 "Using the dialog "Declare variable"" on page 86](#)
- [chapter 17.6.2 "Attribute pragmas" on page 379](#)
- [chapter 17.6.3 "Conditional pragmas" on page 413](#)
- [chapter 17.6.1 "Message pragmas" on page 378](#)

9.7 Using library function blocks

Libraries are collections of objects that can be linked to your application. Use the objects in the libraries like objects defined in the project.

Libraries can include:

- POU's, such as function blocks or functions
- Interfaces and their methods and properties
- Data types, such as enumerations, structures, aliases, unions and arrays
- Global variables, constants, parameter lists
- Text lists, image collections, visualizations, visualization elements
- External files (e.g. documentation)

The library manager is used for managing libraries in a project. For the previous installation of the library on the system, use the "Library repository" dialog.

Also refer to

- [chapter 16 "Creating and managing libraries" on page 219](#)

Using a library function block

The following instructions describe how to insert the counter function block CTUD from the Standard library into your program.

1. Open a POU in the editor and place the cursor in the declaration section.
2. Specify the name for the function block instance, followed by a colon (example: iCounter1:).
3. Press <F2> to open the Input Assistant.
4. In the category **Structured types**, select the CTUD function block from the Standard library (subfolder "Counter").
Select the **Insert with namespace prefix** option.
5. Close the dialog with **OK**.

The function block is inserted with a namespace prefix into the declaration section: `iCounter1:Standard.CTUD`.

6. Complete the line with `;`.

Also refer to

- [chapter 16 "Creating and managing libraries" on page 219](#)

9.8 Managing text in a text list

Text lists are used to provide text in multiple languages for one visualization.

Enter text in Unicode format to have all languages and fonts available.

Text lists can be exported and imported. Thus, it is possible to translate texts outside the current project.

IndraLogic distinguishes between static text managed in the **GlobalTextList** object and dynamic text managed in objects of the type **Text list**.

The **GlobalTextList** object is automatically created (when adding a visualization object in the project) in the "General module folder".

The remaining text list objects can be **added** to the "General module folder". This can take place either project-globally or below the respective application for a control.

Static texts are texts within the visualization that can change their language only at runtime. The text ID remains constant.



The list of **Static texts** is filled automatically in the columns "ID" and "Standard/default" if texts are entered into the visualization editor.

The language-dependent columns (e.g. "en_US") can be translated manually in the "GlobalTextList" or externally after an export. They can then be imported again.

Dynamic texts can be assigned to an IEC variable containing the text ID. It can be used to display varying text in the visualization element at runtime. A text field can for example be configured to output an error text corresponding to an error number.



The list of the **Dynamic texts** has to be created completely by the user (ID, standard/default).

The language-dependent columns (e.g. "en_US") can be translated manually in the text list or externally after an export. They can then be imported again.

Both text list types contain a table with text entries.

An entry consists of an ID for identification purposes, of the source text and its translations.

The source text can be translated into any number of languages in a text list or in a global text list.

Language selection and language toggling are based on the translations.



The names of the language-dependent columns should be identical in all tables.

- Filling GlobalTextList**
1. Open your visualization object and enter the texts into the text fields for example.
 2. Open **GlobalTextList** in the "General module folder".

The **GlobalTextList** contains the automatically saved texts.

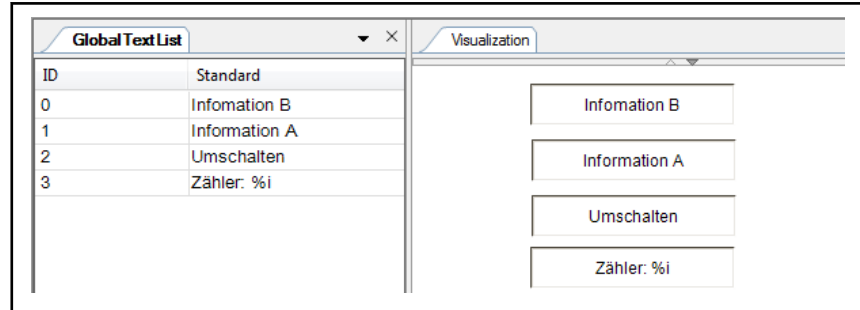


Fig. 9-15: Automatic creation of GlobalTextList (example in German)

- Create text list "Text_list_A" and fill it**
1. Add a text list to the "General module folder" or below an application. Name the list "Text_list_A" for example.
 2. Enter the IDs and texts.

The figure shows manually entered IDs and texts in the "Standard/default" column

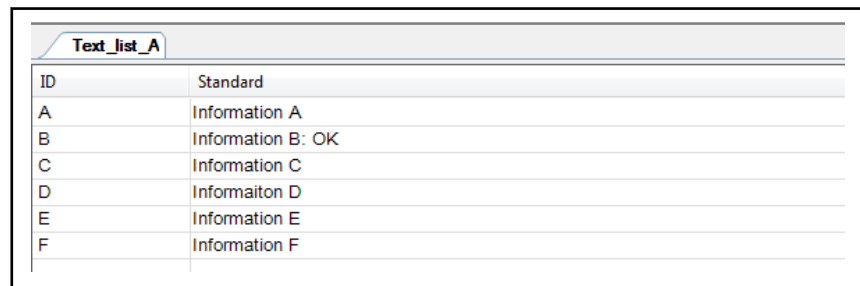


Fig. 9-16: Manual creation of "Text_list_A" (example in German)

Adding languages and translating text

Prerequisite: A project with text list or global text list is open.

1. In the Project Explorer, double-click on an object of the type **Text list** or **GlobalTextList**.

The **Text list** menu is displayed in the menu bar and the text list opens in the editor.

2. Select **VI-Logic text list ► Insert language**
3. Enter a name for the language, e.g. `en-US`. Close the dialog with **OK**.
A column with the `en-US` header.
4. Type in the translation of the source text into the column.



You can correct the name of a language in the table by means of the "Rename Language" command in the context menu of the text list.

Exporting text list

Prerequisite: A text list or the global text list is open in the project.

1. Double-click on the **GlobalTextList** object or on an object of the type **Text list**.

The object opens.

2. Select **VI-Logic text list ► Import/export text lists**

The **Import/export** dialog opens.

3. Click in **Select export file** on  and select the directory and the file name, e.g. Text_lists_exported
4. Select the **Export** option.
5. Close the **Import/export** dialog with **OK**.

IndraLogic exports the text list entries of all texts lists of the project to the file. The table contains a column with the text list names.

	A	B	C	D
1	TextList	Id	Default	en_US
2	GlobalTextList		Information B	Information B_en
3	GlobalTextList		Information A	Information A_en
4	GlobalTextList		Umschalten	Switch
5	GlobalTextList		Zähler: %i	Counter : %i
6	Text_list_A	A	Information A	Information A_en
7	Text_list_A	B	Information B: OK	Information B_en: OK
8	Text_list_A	C	Information C	Information C_en
9	Text_list_A	D	Information D	Information D_en
10	Text_list_A	E	Information E	Information E_en
11	Text_list_A	F	Information F	Information F_en

Fig. 9-17: Export file "Text_lists_exported.csv"

Export file "Text_lists_exported.csv"

```
TextList;Id;Default;en_US
GlobalTextList;;Information B;Information B_en
GlobalTextList;;Information A;Information A_en
GlobalTextList;;Umschalten;Switch
GlobalTextList;;Zähler: %i;Counter : %i
Text_list_A;A;Information A;Information A_en
Text_list_A;B;Information B: OK;Information B_en: OK
Text_list_A;C;Information C;Information C_en
Text_list_A;D;Information D;Information D_en
Text_list_A;E;Information E;Information E_en
Text_list_A;F;Information F;Information F_en
```

Importing file with text list entries

An importable file is a .csv file. The first line is a header, e.g. TextList Id Default en_US.

The other lines contain text list entries. Such a file is created if the text lists of the project is exported to a file. The text list entries can be edited outside of IndraLogic and the file can then be imported.

Upon the import, IndraLogic handles the texts list entries for the Global-TextList and for the dynamic text lists differently.

GlobalTextList

- If the ID is unknown, IndraLogic does not create new text list entries.
- IndraLogic ignores changes related to the ID or the source text.
- IndraLogic applies the translation changes

Text list

- If the ID is new, IndraLogic supplements the respective text list by one text list entry
- If the ID already exists, but it does not match the one in the source text, the source text of the text list is overwritten with the source text of the file.
- IndraLogic applies the translation changes.

Importing file

Prerequisite: A project with text list or global text list is open.

1. Double-click on the **GlobalTextList** object or on an object of the type **Text list**.
The object opens.
2. Select **VI-Logic text list ► Import/export text lists**.
The **Import/export** dialog opens.
3. In the **Choose file to compare or to import** input field, click on  and select the directory and file, e.g. `Text_lists_corrected.csv`.
4. Select the **Import** option.
5. Close the dialog with **OK**.
IndraLogic imports the text list entries of the file into the corresponding text lists.

Example

Content of the file `Text_lists_corrected.csv`

```
TextList;Id;Default;en_US
GlobalTextList;;Information B;Information B_en
GlobalTextList;;Information A;Information A_en
GlobalTextList;;Umschalten;Switch
GlobalTextList;;Zähler: %i;Counter : %i
Text_list_A;A;Information A;Information A2_en
Text_list_A;B;Information B: OK;Information B_en: OK
Text_list_A;C;Information C;Information C_en
Text_list_A;D;Information D;Information D_en
Text_list_A;E;Information E;Information E_en
Text_list_A;F;Information F;Information F_en
Text_list_A;G;Information G;Information G_en
```

This content is applied to the project in the text lists with the same name.

Also refer to

- [chapter "Import/export text lists" command" on page 686](#)

Compare text lists with file and export the difference

Prerequisite: A project with text list or global text list is open.

1. Double-click on the **GlobalTextList** object or on an object of the type **Text list**.
The object opens.
2. Select **VI-Logic text list ► Import/export text lists**.
The **Import/export** dialog opens.
3. In **Choose file to compare or to import**, click on  and select the directory and file name of the comparison file, e.g. `Text_lists_corrected.csv`.
4. In **Select export file**, click on  and select the directory and file name of the file that will contain the result of the comparison.
5. Select the **Export only text differences** option.
6. Close the dialog with **OK**.
IndraLogic reads the import file and compares the text list entries having the same ID. If they do not match, IndraLogic writes the text list entries of the text list to the export file.

For global text lists, IndraLogic compares the translations of identical source texts. If they do not match, IndraLogic writes the text list entries to the export file.

Also refer to

- [chapter ""Add language" command" on page 684](#)
- [chapter ""Import/export text lists" command" on page 686](#)
- [chapter ""Remove language" command" on page 687](#)
- [chapter 18.1.23 ""Text list" object" on page 553](#)

9.8.1 Managing static text in global text lists

The global text list is the central location for texts that are displayed in the visualization ("General module folder").

When configuring a text for the first time in an element in the visualization, IndraLogic creates the global text list.

When writing another text, IndraLogic supplements the table. Therefore, the table includes all texts automatically that you create in the project visualizations.

IndraLogic assigns incremental IDs as integers, starting with "0".

The global text list can be checked against visualizations, updated and compared.

Neither the source text nor the ID can be edited directly in the table. You can replace a source text with another source text by creating and importing a replacement file. Menu commands are provided for this purpose.

Configuring visualization elements with static text

A text in a **GlobalTextList** can contain a format definition.

*Prerequisite: A project with visualization is open. The **GlobalTextList** object contains the texts that are defined in the project visualizations.*

1. Double-click on the visualization.
The editor opens.
2. Select an element provided with the **Text** property (e.g. **Text field**).
3. Enter some text into the **Text** property (example: `Static information A`).

IndraLogic extends the new text to the global text list in the POUs view.

Checking global text list

*Prerequisite: A project with visualization is open. The **GlobalTextList** object contains the texts that are defined in the project visualizations.*

1. Double-click on the **GlobalTextList** object in the Project Explorer.
The table opens with the static texts.
2. Select **VI-Logic text list ► Check visualization text IDs**
IndraLogic reports if a source text of the text list does not match the static text identified via the ID. The source text in the global text list and the text in the visualization having the same ID do not match.

Updating IDs of the global text list

*Prerequisite: A project with visualization is open. The **GlobalTextList** object contains the texts that are defined in the project visualizations.*

1. Double-click on the **GlobalTextList** object in the Project Explorer.
The list opens with the text entries.

2. Select **VI-Logic text list ► Update visualization text IDs**
IndraLogic adds text to the global text list if a text in the **Static text** property does not match the source text in the project visualizations.
- Removing the global text list and creating current IDs again**
- Prerequisite: A project with visualization is open. The "GlobalTextList" object contains the texts that are defined in the project visualizations.
1. Right-click on the "GlobalTextList" object in the "General module folder" and select the "Delete" command.
⇒ The object is removed.
 2. Open a visualization.
 3. Click on **Visualization ► Create global text list**.
⇒ A new "GlobalTextList" object is created in the "General module folder". The global text list contains the static text from the existing project visualizations.
- Removing IDs from the global text list**
- Prerequisite: A project with visualization is open. The **GlobalTextList** object contains the texts that were defined in the project visualizations.
1. Double-click on the **GlobalTextList** object in the Project Explorer.
Table with texts opens.
 2. Select **VI-Logic text list ► Remove unused text list entries**
IndraLogic removes the text list entries whose IDs are not referenced in the project visualizations.
- Editing global text lists with a replacement file**
- A replacement file has the CSV format. The first line is a header: defaultOld defaultNew REPLACE. The following lines contain the old source texts, the new source texts and the REPLACE command. Tabs, commas, and semicolons are permitted separators. A combination of separator characters in a file is not permitted.
- Example (tab as separator character)*
-
defaultalt	defaultneu	REPLACE
Information A	Information A1	REPLACE
-
- When importing a replacement file, IndraLogic processes the replacement file line by line and makes the specified replacements in the **GlobalTextList**. In addition, IndraLogic replaces the previous text with the replacement text in the visualizations. If the replacement text already exists as static text, IndraLogic detects that, harmonizes the static text and leaves only one text list entry.
- Prerequisite: A project with text list or global text list is open.
1. Double-click the **GlobalTextList** object.
The object opens.
 2. Select **VI-Logic text list ► Import/export text lists**
The **Import/export** dialog opens.
 3. At the **Choose file to compare or to import** input field, click on  and select the directory and file (e.g.: ReplaceGlobalTextList.csv).
 4. Select the **Import replacement file** option.
 5. Close the dialog with **OK**.

The texts in the text lists and the visualizations are replaced.

Example

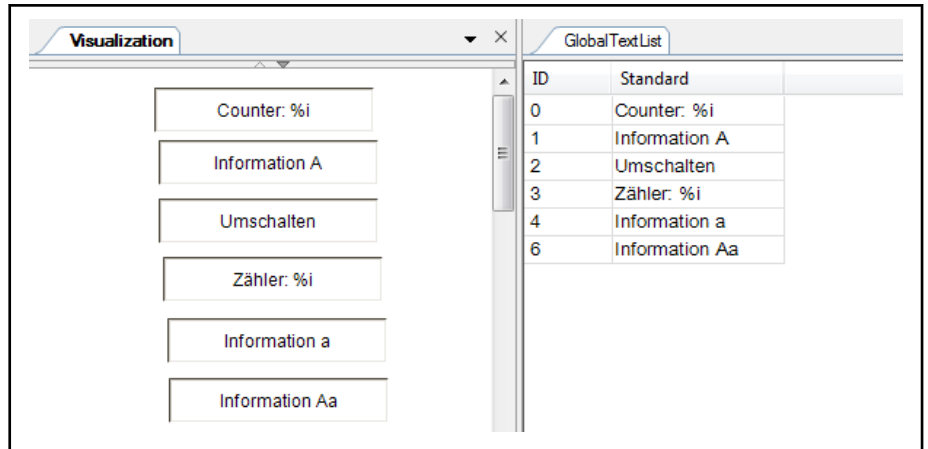


*Fig. 9-18: The global text list contains the following source texts:
The replacement file contains the following replacements:*

```
defaultalt;defaultneu;REPLACE
Counter: %i;Counter1: %i;REPLACE
Zähler: %i;Counter2: %i;REPLACE
Information A;Information A2;REPLACE
Information a;Information A2;REPLACE
Information Aa;Information A2;REPLACE
Umschalten;Switch2;REPLACE
```

IndraLogic detects duplicate text list entries and removes them.

The texts in the visualization have been replaced.

The global text list contains the following entries:

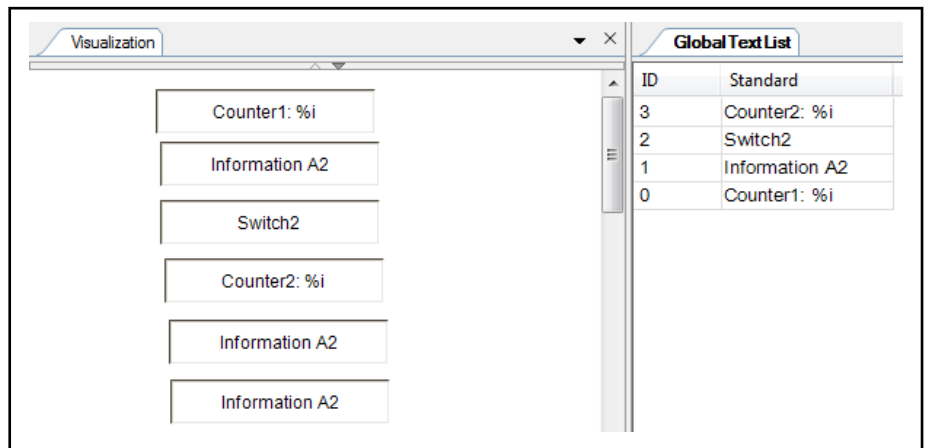


Fig. 9-19: Visualization and GlobalTextList after replacement

Also refer to

- [chapter ""Create global text list" command" on page 684](#)
- [chapter ""Check visualization text IDs" command" on page 687](#)
- [chapter ""Update visualization text IDs" command" on page 688](#)
- [chapter 18.1.10 ""GlobalTextList" object" on page 512](#)

9.8.2 Managing dynamic text in a text list

Texts can be created and translated in a "text list for dynamic text" to subsequently dynamically output them in a visualization.

The object of the type **Text list** can be located globally in the "General module folder" or below an application in the Project Explorer and contains a table with text list entries that can be edited and extended.

A text list entry consists of an ID for identification purposes, of the source text and its translations. New text list entries can be added to a text list. Menu commands are provided for this purpose.

Creating text list for dynamic text output

Prerequisite: A project with visualization is open.

1. Select the "General module folder" or an application in the Project Explorer and select **Add**.
2. Select **Text list**.
3. Enter a name, for example `Text_list_A`. Press **Finish** to close the dialog.

An object of the type **Text list** is created.

4. Click below the **Standard** column and the input field opens. Enter a text, e.g. `Information A`.

The source text is created. It is used as key in the table and as source text for translations.

5. Enter any character string into the column **ID**, e.g. `A`.

A text list entry is defined with source text and ID. When configuring the **dynamic texts** property of an element in a visualization, the text list `Text_list_A` can be selected and the ID `A` can be assigned.

6. Double-click on the empty line at the end of the table under **Standard/default** and enter more text list entries.

ID	Standard
A	Information A
B	Information B
C	Information: OK

Fig. 9-20:

Dynamic text output

Configure the property **Dynamic texts** of an element to configure the dynamic output of texts, which were created in a text list, in a visualization. Text list and ID can be directly assigned as well as IEC variables. Values can be programmatically to these IEC variables.

Prerequisite: A project with visualization is open and a text list is in the Project Explorer.

1. Open the text list, e.g. `Text_list_A`.
2. Double-click on the visualization.

The editor opens.

3. Drag an element, e.g. of type **Text field** to the visualization.

4. Configure its property **Dynamic texts** by selecting a text list in the **Text list** property, e.g. 'Text_list_A' and insert an ID from the text list into **Text index**, e.g. 'A'.

Note the inverted commas.

One IEC variable of type `STRING` can be assigned for the text list name and one IEC variable for the ID.

The IEC variables programmatically allow the access to the texts of the text lists.

5. Compile the application, download it to the control and start it.

The visualization outputs the text from the text list in the text field: `Information A`.

Also refer to

- [chapter 18.1.23 ""Text list" object" on page 553](#)

9.9 Using image pools

An image pool is a table of image files. IndraLogic can unambiguously reference an image file by the ID and name of the image pool when used in the project (e.g. in a visualization). One project can include several image pools. Image pools can be created in the Project Explorer below the application or in the "General module folder".

In a library project, you can use the object properties of an image pool to turn it into a symbol library for the visualization.



We recommend to reduce the size of image files as much as possible before integrating them. This optimizes the loading time of the visualization.

When inserting an screen/image element into a visualization via the names (property of the element "Screen" or "Screen changer"), IndraLogic automatically creates a global image pool. IndraLogic uses the default name **GloballImagePool**. The ID (**Static ID**) is derived from the file name of the image.

Please note the following when the ID of an image file is displayed in several image pools:

- Search order: When selecting an image managed in the **GloballImagePool**, the name of the image pool does not have to be specified.

The search sequence for image files corresponds to the following sequence:

- 1. GloballImagePool
- 2. Image pools assigned to currently active applications
- 3. Image pools that, in addition to GloballImagePool, are located in the "General module" folder
- 4. Image pools in libraries

- Unique access: The image can be directly and uniquely referenced with the ID preceding the name of the image pool in the following syntax

`<PoolName>.<Image ID>`.

Also refer to

- [chapter 18.1.13 ""Image pool" object" on page 515](#)

Creating image pools

1. Select the **Application** object in the Project Explorer.
Select **Add ► Image pool**.
The **Add image pool** dialog opens.
2. Enter a name for the image pool (e.g. "Images1") and click on **Finish**.
The image pool is added to the Project Explorer.
3. Double-click to open the "Image pool" object.
4. Double-click on the **ID** field and assign an appropriate ID (e.g., "Icon1").
As an alternative, add new images to the list via **Add image file** from the context menu.
5. Double-click on the **File name** column. Click on .
The **Select image** dialog opens.
6. Click on  and select the image file.
The image file is displayed in reduced format in the **Image** column. The file name is displayed in the **File name** field.

The image file can now be referenced via the name `Images1.Icon1`.

Also refer to

- [chapter ""Insert image on page 676](#)

Using image files in the "Image" visualization element

When inserting an image element into a visualization, the image type can be defined:

- **Static image:** Enter the image file ID or the name of the image pool + ID into the configuration of the element (**Static ID** property). Observe the notes on search sequence and access.
- **Dynamic image:** Enter the variable defining the image file ID, e.g. `PlcProg.imagevar`, into the element configuration (**Bitmap ID variable** property). A dynamic element can be replaced in online mode with regard to a variable

Also refer to

- [chapter "Element 'Image'" on page 856](#)

Using image file for the visualization background

An image can be specified in the background definition of a visualization. As described above for a visualization element, the image can be defined using the name of the image pool and the file name.

Also refer to

- [chapter 20.2.11 "Command "Background"" on page 773](#)

9.10 Programmed access to I/Os

IndraLogic provides the following functionalities to map project variables to input addresses, output addresses and memory addresses:

- Assigning project variables to input addresses, output addresses and memory addresses in the **I/O Mapping** tab of the device editor
- Programmed access to I/Os
 - Variable configuration
 - AT declaration



We recommend to define the mapping of project variables to input addresses, output addresses and memory addresses in the **I/O mapping** tab of the editor of the respective device.

Also refer to

- [chapter 8.3 "Configuring devices and I/O mapping" on page 73](#)

9.10.1 Variable configuration - VAR_CONFIG

Use the "Variable configuration" to map function block variables to the process mapping.

For declarations in the function block, assign the variables to the device inputs/outputs without providing the full address.

Subsequently, the exact address is provided centrally for all function block instances of the application in a Global Variable List with `VAR_CONFIG` declarations. This Global Variable List with the `VAR_CONFIG` declarations is called variables configuration.



For changes to variables that are assigned to I/O addresses, IndraLogic displays them immediately in the process mapping. For changes to variables that are mapped by a variables configuration, IndraLogic displays them only at the end of the responsible task.

Declaration of variables in the functions block:

When declaring variables in a function block, declare the variables between the keywords `VAR` and `END_VAR`. Assign incomplete addresses to the variables. Label these incomplete addresses with an asterisk (*).

Syntax:

```
<Identifier> AT %<I|Q>*: <DataType>;
```

Example

Two local I/O variables are defined: The input variable `xLocIn` and the output variable `xLocOut`

```
FUNCTION_BLOCK locio
VAR
  xLocIn AT %I*: BOOL := TRUE;
  xLocOut AT %Q*: BOOL;
END_VAR
```

Final definition of addresses in the variable configuration of a Global Variable List:

In the Global Variable List used as variable configuration, define the variable declarations with the exact addresses between the keywords `VAR_CONFIG` and `END_VAR`.

The `VAR_CONFIG` variables have to be specified including the complete instance path. The individual POU names and instance name are separated by a dot.

The declaration has to include an address, whose class (input/output) matches the class of the incompletely specified address (%I*, %Q*) in the function block. The data type has to match as well.

Syntax:

```
<InstanceVariablePath> AT %<I|Q><location>: <DataType>;
```

If the path instance does not exist, an error is reported. IndraLogic outputs an error as well if no address configuration is available for a variable declared with an incomplete address.

Example

The `locio` function block in the example above is used in a program as follows:

```
PROGRAM PlcProg
VAR
    locioVar1: locio;
    locioVar2: locio;
END_VAR
```

A correct variable configuration in a Global Variable List could look as follows:

```
VAR_CONFIG
    PlcProg.locioVar1.xLocIn AT %IX1.0 : BOOL;
    PlcProg.locioVar1.xLocOut AT %QX0.0 : BOOL;
    PlcProg.locioVar2.xLocIn AT %IX1.0 : BOOL;
    PlcProg.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

Also refer to

- [chapter 17.2.13 "Configuration variable - VAR_CONFIG" on page 291](#)
- [chapter 9.10.2 "AT declaration" on page 125](#)
- [chapter 17.4.13 "Addresses" on page 348](#)

Creating a variable configuration

Prerequisites: A project is open that includes a control configuration with a field device. The project contains a program, e.g. `PlcProg` and a function block, e.g. `func1`. The field device is provided with inputs and outputs. The textual view is selected in the options for the declaration editor.

In the function block, assign variables to device I/Os with incomplete addresses and then create a variables configuration.

1. Double-click on the function block in the Project Explorer (e.g. `func1`). The function block editor opens.
2. Enter the following between the keywords `VAR` and `END_VAR` in the declaration section:
`xLocIn AT %I*: BOOL := TRUE;` and in the next line
`XLocOut AT %Q*:BOOL;`
 An input variable `xLocIn` was declared and assigned to the incomplete input address `%I*` of a field device. The declared output variable was assigned to the incomplete output address `%Q*`.
3. Click on the `PlcProg` object in the Project Explorer and add the following to the declaration section of the program between `VAR` and `END_VAR`:
`locioVar1: func;`
`locioVar2: func;`
4. Select the **Application** object in the Project Explorer and then select **Add ► Global Variable List** from the context menu. Then, click on **Finish** in the **Add Global Variable List** dialog.

The Global Variable List is added to the Project Explorer and its editor opens.

5. Change the keyword `VAR_GLOBAL` to `VAR_CONFIG`.

6. Select **Add all instance paths**

The following instance paths are added:

```
PlcProg.logioVar1.xLocIn AT %I*;
PlcProg.logioVar2.xLocIN AT %I*;
PlcProg.logioVar1.xLocOut AT %Q*;
PlcProg.logioVar2.xLocOut AT %Q*;
```

7. Now, replace the incomplete addresses `%I*` and `%Q*` with the absolute, complete addresses.

Also refer to

- [chapter "Command 'Add all instance paths'" on page 679](#)

9.10.2 AT declaration

In the variables declaration, the keyword `AT` assigns a project variable to a specific input address, output address or memory address of the control configured in the Project Explorer. The assignment of variables to an address in the **I/O mapping** dialog of the device can also be specified in the control configuration.

Syntax:

```
<variable name> AT <address> : <data type>;
```

```
<address> : %<memory area prefix> (<size prefix> ? <memory position>)
```

The `AT` declaration allows you to give the address a meaningful name. Make any necessary changes for the incoming or outgoing signal at only one location, e.g. in the declaration.

Examples:

VAR wInput AT %IW0 : WORD; END_VAR	Variable declaration with address information of an input word
VAR xActuator AT %QW0 : BOOL; END_VAR	Boolean variable declaration Note: For Boolean variables, one byte is allocated internally if a single bit address is not specified. A change in the value of <code>xActuator</code> affects the range from <code>QX0.0</code> to <code>QX0.7</code> .
VAR xSensor AT IX7.5 : BOOL; END_VAR	Boolean variable declaration with explicit specification of a single bit address. On access, only the input bit 7.5 is read.
VAR xSensor AT IX* : BOOL; END_VAR	For the address specification, the placeholder <code>*</code> is given instead of the memory position. The final address specification is done in the variables configuration. Note: This is possible in function blocks.

When assigning a variable to an address, observe the following:

- Write access is not possible to variables assigned to an input. This leads to a compiler error.
- AT declarations can only be made for local and global variables, not for input variables and output variables of blocks.
- Furthermore, AT declarations cannot be used in persistent variable lists!
- If you use AT declarations for structure components or function block variables, then all instances use the same memory. This corresponds to the use of "static variables" in classic programming languages, such as "C".
- The memory layout of structures also depends on the target system.



If you do not specify a single bit address explicitly, then Boolean variables are allocated byte-by-byte.

Example:

```
PROGRAM PLC_PRG
VAR
  xVar AT %QW0 : BOOL;
END_VAR
```

```
xVar := TRUE;
```

Also refer to

- [chapter 9.10.1 "Variable configuration - VAR_CONFIG" on page 123](#)
- [chapter 17.2.13 "Configuration variable - VAR_CONFIG" on page 291](#)
- [chapter 17.4.13 "Addresses" on page 348](#)

9.11 Checking syntax and analyzing code

IndraLogic provides useful functions for detecting errors. It also assists when creating the program. The syntax check already flags errors and outputs error messages while programming.

The static code analysis in IndraLogic also assists you in complying with defined coding guidelines and detecting weak constructs.

Also refer to

- [chapter 9.11.1 "Checking syntax" on page 126](#)
- [chapter 9.11.2 "Analyzing code statically" on page 127](#)

9.11.1 Checking syntax

When entering code, the precompilation already runs some basic checks in IndraLogic. Wavy red underlines are shown below the incorrect location in the editor and an error message is output in the message view.

After programming, compile the application. For execution, select **Create ► Build** or press <Shift>+<F11>. The compiler checks the program and outputs possible further errors.

IndraLogic automatically generates the application code from the source code written in the development system before downloading the application to the control. Before the application code is generated, assignments, data types and library availability are checked. In addition, the memory addresses are allocated when generating the application code. To explicitly execute this command, select **Build ► Generate code**. This is useful for detecting any errors in your source code, even when the PLC is not connected yet

IndraLogic outputs all errors and warnings in the message view in the "Build" category. Double-click on the error message to open the respective POU in the editor. The incorrect location is highlighted. As an alternative, jump to the incorrect locations via the context menu of the error message.

Please note the settings in the IndraLogic options.

Also refer to

- [chapter "Options" dialog - "Smart coding" on page 730](#)

9.11.2 Analyzing code statically

IndraLogic allows to statically analyze (also called "lint") the source code during the code generation.

When "linting", it is determined whether the source code complies with the coding guidelines you defined - according to the idea behind the "lint" analysis tool.

Define the coding guidelines or rules in the "General module folder" of the Project Explorer in the "Project settings" → "Static Analysis Light".

The check is performed automatically each time code is generated, for example when you click on "Build" → "Generate code".

If there are divergences from the rules, they are reported as an error messages in the "Build" category of the message window.

The reported errors have the prefix: SA<Number>.



For static code analysis with "Static Analysis Light", only the application code of the project is checked.

Libraries are excluded from the check.

GVL variables in the "General module" folder are not always checked:

If there are projects with several applications, only the objects in the active application are checked.

If there is only one application in the project, the objects in the "General module folder" are also checked.

Deactivating lines of code in the implementations with pragmas from the static analysis

Use the pragma {analysis ...} to mark the program code so that the specified rules are not checked. As a result, the marked lines of code are not subjected to static analysis. The marked code is ignored during the check.

Syntax:

```
{analysis <Sign><RuleNumber>|,<More combinations of sign and rule number, separated with comma>}
```

-<RuleNumber>: Disable the check of the rule SA<RuleNumber>.

+<RuleNumber>: Enable the check of the rule SA<RuleNumber>.

Excluding implementation code

Prerequisite: The rules for the static analysis are enabled in the "Project settings".

Procedure to exclude a code area from the analysis:

1. Add the pragma {analysis -<number>} above the line of code that contains the beginning of the code not to be checked. e.g. for the rule SA0024

→ The line of code is the first line of the code snippet that is not checked with rule SA0024.

2. Add the pragma `{analysis +<number>}` below the line of code that contains the end of the code not to be checked. e.g. for the rule SA0024

→ The line of code above is the last line of the code snippet that is not checked with rule SA0024.

Example: Ignore untyped literal

The following code example shows how a rule for checking can be disabled for two line, i.e. "SA0024: Untyped literals only" is deactivated for two lines.

No error is issued due to the disabling even though the code line `nTest := #99;` is incorrect (correct code: `nTest := DINT#99;`).

```
{analysis -24}
nTest := 99;
iVar := INT#2;
{analysis +24}
```

Example: Ignoring several rules

The following code example shows how to disable and enable multiple rules simultaneously:

Program:

```
{analysis -10, -24, -18}
...
{analysis +10, +24, +18}
```

- SA0010: Arrays with only one component
- SA0024: Untyped literals / constants
- SA0018: Unusual bit access

Excluding programming objects with pragmas from the static analysis



The rule "SA0004: Multiple write access on output" cannot be disabled using a pragma.

Syntax:

```
{attribute 'analysis' := '<RuleNumber>[,<More negative rule numbers, separated with comma>']}
```

When you insert the attribute pragma in the declaration part of a programming object, the specified rules are excluded for the entire programming object.

If multiple rules are excluded, then the rules are each comma-separated with a dash and a number. A pragma statement for activation is not required.

Examples:

Code example to ignore the rules SA0033 and SA0031 for the structure `LocalData`.

```
{attribute 'analysis' := '-33, -31'}
TYPE LocalData :
STRUCT
  iLocal : INT;
  uiLocal : UINT;
  udiLocal : UDINT;
END_STRUCT
END_TYPE
```


Code example to ignore the rule SA0100 in the ARRAY big.

```
{attribute 'analysis' := '-100'}
big: ARRAY[1..10000] OF DWORD;
```

Also refer to

- [chapter "“Project settings” dialog - “Static analysis light”" on page 716](#)

9.12 Orienting and navigating

9.12.1 Using the cross-reference list to find places of use

The occurrences of symbols of a variable, a POU (program, function block, function), or a DUT can be displayed in a so-called "cross reference list". From the list it can be jumped to the respective position in the project.

There are two ways to search for occurrence locations of a symbol:

- Plain text search: Manually specify a text (symbol name, placeholder) in the "Cross reference list" view.
- Search for a specific declaration:
 - In the "Cross-reference list" view, select the declaration from the input wizard.
 - The focus is on a symbol name in the block editor. Start the cross reference search using a context menu command.
 - The focus is on a symbol name in the block editor, the "Cross-reference list" view is opened, and the cross-reference search is executed automatically.
 - You are in the "Cross reference list" view which already lists occurrence locations for several declarations, you limit these results to a specific declaration.

Cross references via text search by symbol name

Prerequisite: The "Cross reference list" view is open.

1. Specify a string in the field next to the name, for example the identifier of the variable for which you want to find the occurrence location in the project. Example: "iCounter".

For text search, use the placeholders "*" (any number of characters) or "?" (exactly any one character) in combination with a partial string of a variable identifier.

Use "%" to search for IEC addresses. Examples: "%MW8", "%M*".

2. Click on the  button to start a text search in the project.

The view "Cross reference list" opens and displays the occurrence locations for the iCounter variable. The declaration parts are always displayed in the project with the occurrence location indented.

3. Double-click on an occurrence location in the cross reference list.
The respective object opens in the editor with the marked place of use.

Cross references for a specific symbol declaration ...from the block editor, with a menu command

Prerequisite: A block is open in the editor.

1. Position the cursor on the identifier of the symbol (variable, block name) in the declaration part or the implementation part.
2. In IndraWorks, select **Edit ► Browse ► Output cross references** or **Browse ► Output cross references** from the context menu.

The **Cross reference list** view opens and shows the places of use of the variables or the POU.

...from the block editor, automatic

Select the "Automatically list selection in cross reference view" option in the IndraLogic options: Menu **Tools** ▶ **Options** ▶ **IndraLogic 2G** ▶ **SmartCoding**.

The cross-reference list automatically shows the occurrence locations for the respective symbol.

Select the name of the symbol in the block, or position the cursor in the name.

... in the cross reference list view
with input wizard

In the "Cross reference list" view, use the input assistant  to specify a symbol name in the field next to "By declaration".

The cross reference list displays the occurrence locations for the symbol.

... in the cross reference list view,
limited to a specific declaration

If multiple declarations for a symbol are listed in the "Cross-reference list" view, for example after a text search, reduce the display to one result:

Select the line with the desired declaration and click on the  button or click on "Limit results to current declaration" in the context menu.

The cross reference list includes only the occurrence locations for the selected declaration.

Also refer to

- [chapter ""Cross reference list" command" on page 590](#)
- [chapter "Command "Browse cross-references in classic view"" on page 594](#)
- [chapter ""Options" dialog - "Smart coding "" on page 730](#)

9.12.2 Finding declaration - "Go to definition"

IndraLogic allows to browse through the entire project for the definition location of a variable or function. The function block containing the definition opens in the editor with the declaration is highlighted.

Finding the declaration of a variable

Prerequisite: A POU is open in the editor.

1. Place the cursor on an identifier in the implementation section.
2. Select **Edit** ▶ **Browse** ▶ **Go to definition**.

The POU with the declaration opens in the editor and the variable definition is highlighted. If the definition is located in a "compiled" library, the respective function block opens in the library manager.



The command can be used in online mode and in offline mode.

Examples

The following block includes a function block definition (fbinst), a program call (prog_y()) and a function block call (fbinst.out):

```
VAR
  fbinst:fb1;
  ivar:INT;
END_VAR
prog_y();
ivar:=prog_y.y;
res1:=fbinst.out;
```

Place the cursor on `prog_y` and the command opens the `prog_y` program in its editor.

Place the cursor on `fbinst` and the command sets the focus to the declaration window to the line `fbinst:fb1`;

If the cursor is placed on `out`, the command opens the function block `fb1` in its editor.

Also refer to

- [chapter "'Go to definition' command" on page 577](#)

9.12.3 Setting and using bookmarks

Use bookmarks to facilitate the navigation in long programs. Bookmarks can be used in all editors of the programming languages except for SFC (sequential function chart). Use commands to directly navigate to the highlighted program locations.

Toolbar: Bookmark

Setting and deleting bookmarks

Prerequisite: The POU is open in the editor

1. Place the cursor at any program position.
2. Select **Edit ► Bookmark ► Toggle bookmark**.

A bookmark is set at this position in the program. This is marked by the bookmark symbol .

3. Set multiple bookmarks at different program locations.
4. Place the cursor to any program line with bookmark.
5. Select **Edit ► Bookmark ► Toggle bookmark**.

The bookmark is removed again. The bookmark symbol is deleted. Alternatively, delete one or more bookmarks in the "Bookmarks" view by clicking the  button. For this purpose, the corresponding bookmarks have to be selected in the "Bookmarks" view.



Select **Edit ► Bookmark ► Delete bookmark** to remove all bookmarks.



In order to delete all bookmarks in a project, click "Clear All Bookmarks". The command is not part of the standard command of the IndraWorks user interface. To add the command to the menu, go to **Tools ► Customize... ► Commands ► Edit**.

Also refer to

- [chapter "'Toggle bookmark' command" on page 576](#)
- [chapter "'Delete bookmark' command" on page 577](#)

Jumping to bookmarks within a POU

Prerequisite: The POU is open in the editor. Multiple bookmarks are set.

1. Select **Edit ► Bookmark ► Next bookmark**.

Depending on the current cursor position, the cursor jumps to the next bookmark (down).

2. Select **Edit ► Bookmark ► Previous bookmark**.

Depending on the current cursor position, the cursor jumps to the previous bookmark (up).

Also refer to

- [chapter "'Next bookmark' command" on page 576](#)
- [chapter "'Previous bookmark' command" on page 577](#)

9.13 Project-comprehensive "Find and replace"

In IndraLogic, character strings in individual objects as well as in the complete project can be searched and replaced by another character string.

1. In the **Edit ► Search and replace** menu, select the **Search** command.
The **Search** dialog opens.
2. Enter the character string to be searched for into the **Search for** field.
3. Select the search option.
4. Specify the objects to be browsed through in the **Browse in** selection list.
5. Click on **Find next**.
The first hit is shown.
6. Click on **Replace** to replace the searched term by another one.
7. Click on **Find all** for an overview on all hits.

9.14 Refactoring

In general, refactoring is a technique for improving the design of existing software code without changing the way it functions.

In IndraLogic, refactoring provides functionalities to rename object names and variable names and to update function block connections.

All places of use of renamed objects and variables can be displayed and then renamed all at once or individually.

To additionally configure the locations where IndraLogic prompts for refactoring, go to **Tools ► Options ► IndraLogic 2G**.

Renaming a global variable

Prerequisite: A project is open with at least one function block (FB) and one Global Variable List (GVL). The Global Variable List **GVL** is now opened in the editor and contains the declaration of a variable, e.g. `iGlobal`. **FB** uses `iGlobal`.

Renaming a global variable across the project

1. Select the global variable name, e.g. `iGlobal` in **GVL**.
2. In the context menu, select the command **Refactoring ► Rename iGlobal**
3. Enter a new name in the input field **New name** in the **Rename** dialog, e.g. `iGlobalOK` and click on **<OK>**.

The **Refactoring** dialog opens. In the Project Explorer in the left window, the **GVL** and **FB** objects are highlighted in red and yellow. The **FB** function block is open in its editor in the right window and `iGlobal` has already been renamed to `iGlobalOK`.

4. Click on **<OK>**.
- 5.

No global variable `iGlobal` is contained in your project. `iGlobalOK` is now contained in the project.

Renaming a global variable across a project (with the exception of a POU)

1. Select the global variable name, e.g. `iGlobal`.
2. In the context menu, select the command **Refactoring ► Rename iGlobal**
3. Enter a new name in the input field **New name** in the **Rename** dialog, e.g. `iGlobalTest` and click on **<OK>**.

The **Refactoring** dialog opens. In the Project Explorer in the left window, the **GVL** and **FB** objects are highlighted in red and yellow. The **FB** function block is open in its editor in the right window. `iGlobalTest` is listed instead of `iGlobal`.

4. Place the cursor in the right window and open the context menu.
5. Select **Discard this object** and press **<OK>**

The global variable `iGlobal` is now available in **FB** in your project. In objects that previously contained the variable, the variable `iGlobalTest` is now specified.

In the message window, the error message is reported that the identifier `iGlobal` is not defined.

Also refer to

- [chapter "'Rename refactoring' command" on page 579](#)
- [chapter "Command 'Refresh affected connections'" on page 583](#)

Adding and removing an input variable

In the declaration section of blocks, you can add and delete input and output variables by using the refactoring commands.

IndraLogic updates at the places of use/calling locations of the function blocks. The places of use can be accepted or rejected individually. The **Refactoring** dialog opens.

Prerequisite: The **FCT** POU of the function type is open in the editor. The function already contains the input variables `input1`, `input2`, and `inputx`. They are called in the **PlcProg** and **POU** programs.

1. Set the focus in the declaration section of the **FCT** function.
2. Select **Refactoring ► Add variable**.

The default dialog opens to declare variables.

3. Declare the variable `input_3` with the scope of `VAR_INPUT` and data type `INT`. Close the dialog with **<OK>**.

The **Refactoring** dialog opens (see figure below). The affected locations are highlighted in yellow (1)+(2).

4. In the upper right corner, select **Add inputs with placeholder text** (3).
5. In the left side of the window, click on one of the highlighted objects (e.g., `PLC_PRG`). Select **Accept whole project** from the context menu to add the new variable at the new places of use of **FCT** for the entire project.

The change in the implementation section of `PLC_PRG` can be seen on the right side of the window: The `_REFACTOR_` placeholder is displayed at the location where the new variable was inserted.

6. Press **OK** to close the <Refactoring> dialog.
7. Select **Edit ► Search**. Search in the project for "`_REFACTOR_`" to check and edit the affected locations.
- 8.



As an alternative, insert the new variable with another initialization value without working with a placeholder first.

In this case, select "Add inputs with the following value" in step 4 and enter the value into the field on the right.

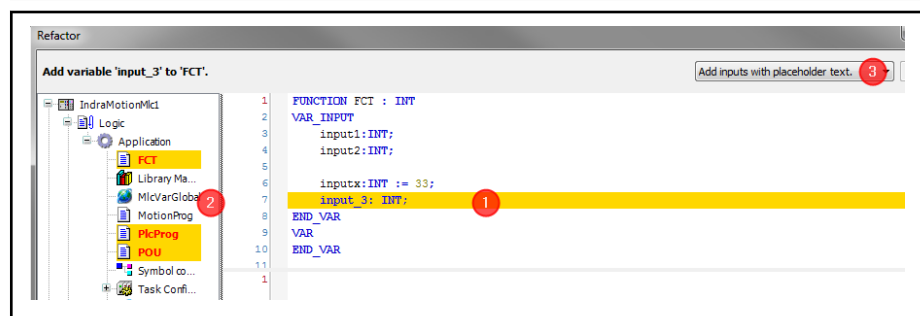


Fig. 9-21: Example of a new variable with placeholder text

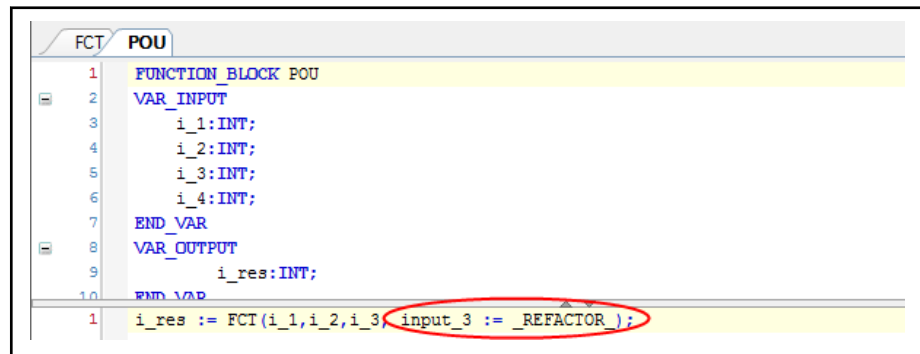


Fig. 9-22: Example of a new variable with placeholder text in an ST function block:

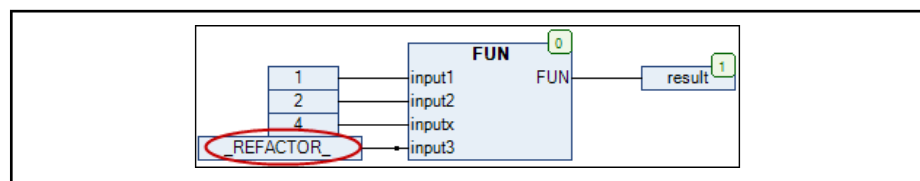


Fig. 9-23: Example of a new variable with placeholder text in a CFC function block



Please note that variables can also be removed with refactoring.

Also refer to

- [chapter "Command 'Add variable'" on page 580](#)
- [chapter "Command 'Remove variable'" on page 582](#)

Reordering variables in the declaration

In the declaration part of function blocks, you can change the order of declarations by refactoring.

This is possible for declarations with scope `VAR_INPUT`, `VAR_OUTPUT`, or `VAR_IN_OUT`.

Prerequisite: The declaration part of a POU is open and includes declarations, for example:

```
VAR_INPUT
invar2 : INT;
invar1 : INT;
in : DUT;
bvar : BOOL;
invar3 : INT;
```

1. Right-click in this declaration block to access the context menu.
2. Click **Refactoring ► Reorder variables**.
The "Reorder" dialog box opens with a list of `VAR_INPUT` variables.
3. For example, select the entry `invar1 : INT;` and drag the entry in front of the `invar2` entry.
The `invar1` declaration is now at the top position.
4. Close the dialog with <OK>.
The "Refactoring" dialog is displayed. The affected locations are marked in yellow (see figure above).
5. Click <OK> to accept the new order for the function block.

Change variable declaration und automatically use refactoring

A wizard supports the user in renaming of variables in the declaration (by using an auto declaration) of refactoring.

Prerequisites:

- The following refactoring options are enabled:
 - "During renaming of variables"
 - "For adding or removing variables or for changing the namespace"
 Call: Menu **Tools ► Options ► IndraLogic 2G ► Refactoring**
- POU with function block, in the example `fb_A`
- A declared variable, in the example `iTest`

Example: Changing a variable name

1. Double-click on the function block `fb_A`.
2. In the declaration of `fb_A`, select a variable, e.g. `iTest`. Optionally, the cursor can be set in front of or in the variable.
3. Open the "Declare variable" dialog via the menu command **Edit ► Declare variable**, or via the shortcut [Shift]+[F2].
The "Declare variable" dialog opens. It contains the settings of the `iTest` variable.
In the dialog, the option "Changes by means of refactoring" is enabled.
4. Change the name of `iTest` to `iCounter_A` and confirm the change with "OK".
The dialog "Rename refactoring from `iTest` to `iCounterA`". The positions affected by the renamed variables are highlighted.

5. Close the dialog with "OK".
The changes are applied.

Also refer to:

- [chapter "'Rename refactoring' command" on page 579](#)
- [chapter "Command 'Refresh affected connections'" on page 583](#)
- [chapter "Command 'Add variable'" on page 580](#)
- [chapter "Command 'Remove variable'" on page 582](#)

9.15 Task configuration

In the task configuration, define one or more tasks for controlling and executing the application program on the control.

Each application has to be provided with a **Task configuration** object.

A task is a time-based sequence unit of an IEC program.

Define a task with a name, a priority and a type, which determines the condition triggering the start of the task. Define this condition either by time (cyclic-interval, freewheeling) or by an internal or external event to start the task processing. An event is for example the rising edge of a global project variable or an interrupt event of the control.

A task calls one or multiple program function blocks (POUs). These programs can be application-specific (objects below the application in the Project Explorer) or project-specific (objects available in the "General module folder"). In the latter case, the application instantiates the project-global program.

If IndraLogic processes the task in the current cycle, the programs are executed for one cycle.

Combining priority and condition determines the temporal sequence when executing the tasks. For each task, a watchdog can be configured. Start, stop and reset can also be coupled automatically to the execution of the project function block.

Rules for the processing order of the defined tasks:

- If the task condition is met, IndraLogic processes the task.
- If several tasks simultaneously meet the condition for processing, IndraLogic processes the task with the highest priority first.
- If several tasks with the same priority level meet the condition for processing at the same time, IndraLogic processes the task waiting for the longest first.
- The program calls are processed in the order assigned in the "Configuration" dialog of the task.
- If a called program has the same name in the Project Explorer of the application and in a library or project-globally in the "General module folder", the application program is used.

Important notes

All tasks share a process image.

Reason:

When each task has its own individual process image, performance is reduced. However, the process mapping can only be consistent to one task.

When creating a project, ensure that the application copies the input data and output data to a safe location in case of conflicts.

Function blocks of the SysSem.library for example allow to solve consistency and synchronization problems.

There can also be consistency problems when accessing other global objects, such as global variables or function blocks. Consistency problems always occur if several tasks use one variable for reading and writing. Modules, such as the library SysSem, are available.

Also refer to

- [chapter 18.1.25 ""Task configuration" object" on page 562](#)
- [chapter 18.1.26 ""Task" object" on page 565](#)

9.15.1 Creating task configurations

Prerequisite: The open project includes a POU of type "program" and a **Task configuration** with a **Task** object was inserted below **Application** in the Project Explorer.

1. In the Project Explorer, double-click on the task object below **Task configuration**.

The **Configuration** tab of the task object opens.

2. Select **Cyclic** in the selection list **Type**.

The **Interval (e.g. t#200ms)** input field is displayed.

3. In the **Interval (e.g. t#200ms)**, enter the value `t#300ms`.

4. Click on **+Add call**.

The input assistant opens.

5. In the **Input wizard ► Categories** select the POU from the **Category Program** and click on **OK**.

IndraLogic inserts the selected POU into the POU list of the **Configuration** tab and below the task object in the Project Explorer.

When the application is executed on the control, IndraLogic executes the selected POU in cyclic intervals of 300 ms.

- [chapter ""Configuration" tab" on page 566](#)

9.16 Data persistence

The lifespan of a variable and its data begins at the time when the variable is created and ends at the time when the variable is deleted and its memory is freed. The time when the variable is created, initialized, or instantiated depends on the declared scope. The time when the memory is freed usually depends on the scope as well. For example, the memory of global variables is freed by exiting the application.

They can retain data longer than usual. The following mechanisms are provided for this purpose.

Mechanisms for data retention:

- (A): Persistent global variable list  with the keyword `PERSISTENT RETAIN`

Persistent variables retain their values when the application is reloaded. Moreover, the values are restored after a download, warm start, or cold start.

Programming applications

- (B): Retain variables with the keyword `RETAIN`
Retain variables retain their values after a warm start, but not after re-loading the application, a download, or a cold start.
- (C): Recipe variables
Recipe variables and their values are stored in a recipe file.

Comparing mechanisms

Which mechanism is suitable for which application?

The following table lists some common use cases from the building control field.

Comparison of mechanisms and use cases

	Use case	(A) Persistent variables	(B) Retain variables	(D) Recipe variables
1	The application must maintain device settings. Example: After a power failure, the building control has to have information on how long a window blind needs to be raised.	Suitable¹ Preferred use case In this case, you can also use retain variables instead of persistent variables. This is advantageous for variables whose declaration is often changed.	Suitable Preferred use case Retain variables are an advantage when their declarations are changed often.	Possible, but very complicated and therefore not recommended.
2	The application has to maintain values also after program changes or extensions.			
	2a: Rare extensions Example: An application programmer extends the program with a new switch and installs a new light. The building control must still have saved values available until then.	Suitable¹ Preferred use case	Suitable	Possible, but complicated
	2b: Unrestricted changes, including deleting or changing the data type of variables The building control is running and is persistent. When an application programmer adds a new functionality to the controller and therefore adds another persistent variable to a function block, the values saved up to that point must be retained. For example, the program in a function block is extended with a variable that controls the automatic switching off of a previously uncontrolled lamp after a certain time. The building control must have the times of all controlled lamps available after the extension.	Not suitable	Suitable Data from retain variables are preserved as far as possible after an online change.	Possible if textual, but complicated
	2c: The application has to maintain values after a download.	Suitable	Not suitable	Suitable

	Use case	(A) Persistent variables	(B) Retain variables	(D) Recipe variables
3	The application must be able to use different value sets. Example: The operating settings for summer, winter, and holidays must be saved and imported when needed.	Not suitable	Not suitable	Suitable Preferred use case
4	The application must be able to use settings from another system. It must be possible to transfer settings to another plant using similar variables.	Not suitable	Not suitable	Suitable²
5	The application must provide human readable data. The user must be able to read, compare, and edit the data.	Not suitable	Not suitable	Suitable²

1

Advantage: Speed; recommended use case: 1 and 2a

2

Advantage: Externally editable, transferable. Disadvantage: Complicated

Lifespan of variables when calling online commands

User input in the Online menu	Variable with usual life-span (neither RETAIN nor PERSISTENT)	RETAIN	PERSISTENT RETAIN PERSISTENT PERSISTENT RETAIN
Command "Online Change"	X	X	X
"Reset warm" command	-	X	X
"Reset cold" command	-	-	X
Command "Download"	-	-	X ¹
"Reset origin" command	-	-	-

X

Variable retains its value

-

Variable is initialized

1

Note: For the structure of persistent data, see the information in ["Download mechanism" on page 140](#)

Also refer to:

- [chapter "Command 'Online change'" on page 619](#)
- [chapter "Command 'Reset origin'" on page 624](#)
- [chapter "Command 'Reset cold'" on page 623](#)
- [chapter "Command 'Load'" on page 618](#)

Lifespan of variables when downloading a boot project

The values of ordinary variables lose their value and are reinitialized.

The values of persistent variables are protected if:

- The structure of the persistent variable in memory matches the structure in the persistent data list

The values of retain variables are protected if:

- The structure of the persistent variable in memory matches the structure in the persistent data list
- The persistent variables match the application (GUID has to match)

A "Retain mismatch" occurs when the requirements for restoring the values of retain variables and persistent variables are not met when the application is booted.



For the structure of persistent data, see the information in ["Download mechanism" on page 140](#)

9.16.1 Retaining data using persistent variables

Persistent variables retain their values after reloading the application, and after a download, warm start, or cold start.

Data is not backed up in the non-volatile NVRAM memory of the control and does not require additional time.

Behavior *Value retained for:*

- Uncontrolled exit
- Warm start by calling the "Reset Warm" command
- Cold start by calling the "Reset Cold" command
- Repeated download of the application

Reinitialization for:

- Call of the "Reset Origin" command

Therefore, persistent variables are reinitialized only if you reset the controller to the factory settings, for example, click on **Debug ► Reset origin**.

If, on the other hand, you download the application again, the persisted data is retained if possible. That depends on how profound the changes that led to the download were. Changing the application name always leads to a full reinitialization. Changes to the implementations never lead to a reinitialization: the data persistence is completely preserved. Changes made to the declarations initialize the new variables only if the existing variables are persistent and if the following is considered: When changing the declarations, ensure that the persistent variable list remains consistent. This is the case when you add a new variable or delete an existing one. Inconsistencies can occur if you edit and change the identifiers or data types of previously declared persistent variables.

Download mechanism Editing the variable list in the persistence editor causes the variable list to be edited automatically before it is saved, not to be saved as it is shown in the editor.

During post-processing, a variable that you have removed is replaced by a placeholder variable with the same memory requirement. As a result, the subsequent variables retain their addresses in the process image. Moreover, a variable you add is moved to the end of the list. Post-processing can neutralize changes that would lead to a loss of persistence. But you create gaps that use additional memory.

When downloading, the CRC value of the variable list and the length of the list (number of variables) are stored on the controller. When downloading again, the new test value is compared with the test value currently on the controller. Then the variable list is compared successively up to the specified length. If you have edited a declaration (for example, the name or data type), then the variable is reinitialized. Otherwise its value is retained. When the download is repeated, IndraLogic checks whether the variable list declared in the persistence editor is still consistent with the variable list already on the controller.

The mechanism is reliable if the variables themselves are not modified significantly. Too extensive changes of the identifiers and the data types cause re-initialization and loss of persistence. If frequent changes are probable due to application requirements, a variable list is not recommended. If it is changed online after a data type change, a persistent variable is less robust than a common variable without persistence.

It is recommended to clear the gaps from the variable list. Select the context menu item "Reorder list and clear gaps" in the declaration section of the persistence editor.



Clearing the variable list initializes all persistent variables, as the cleared list does not match the list on the control.

The persistence of all variables on the control is lost during the initialization.

Recovering data using the recipe manager

To clean up the global persistent variable list without losing persistence, save data in a recipe using the recipe manager. This creates a list for all variables of the persistent variable list in the recipe manager, and at the same time its current values are stored by the controller as a recipe. Select the context menu item "Reorder list and clear gaps" in the declaration section of the persistence editor and download again. Select the context menu item "Restore values from recipe" and the values saved in the recipe are restored.

Changing an existing declaration in the persistent variable list

Changing the name or data type of a variable is interpreted as new declaration and causes an initialization of the variables upon the next online change or download. For complex data types, a change occurs when a new component is added, or when changing the type of a variable from INT to UINT in the depth of a structure used for example.

Basically, complex user-defined data types are not suitable to be managed in a persistent variable list, as even a small change causes the variable to be initialized with all components.

Double allocation of memory for instance paths

The variables can be made persistent for:

- Global variables
- Locally declared variables in a function block
- Variables declared in the program

Add the variable declaration using the keyword `PERSISTENT`.

Add the instance path to the persistent variable additionally to the persistent Global Variable List using the context menu item "Add all instance paths" in the persistence editor.

Persistence is guaranteed by the following mechanism:

- The cyclic tasks in which the variable is accessed are determined.
- At the end of the first cyclic task (in each cycle), the variable is copied to the persistent global variable list.
- After restarting the controller, the value of the persistent variable is copied to the ordinary variable.

The disadvantage of this mechanism is that memory is allocated both at the place of declaration and at the place of the instance path. The persistent variable requires double the memory allocation. Moreover, the data is copied to both places in each cycle. Copying can be time consuming, especially if large, structured values are involved.

Memory location for persistent function block instances

A function block instance is always stored completely in memory. This is necessary so that the same code can work on different instances. If already one variable in a function block is labeled with `PERSISTENT`, the function block

instance is stored completely with all variables in remanent memory, although only the one variable is treated as persistent. However, non-volatile memory is not available to the same extent as main memory.

A function block with a pointer to an instance in NVRAM as a variable is not stored in the protected memory.

See:

- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter "Command 'Add all instance paths'" on page 679](#)

9.16.2 Preserving data with retain variables

Retain variables preserve their values after a warm start. However, the degree of value retention for persistent variables is higher.

Retain variables are not backed up in the non-volatile NVRAM memory of the control and does not require additional time.

Declaration To declare a retain variable, add the `RETAIN` keyword to a variable declaration.

Behavior *Value retained for:*

- Uncontrolled exit
- Call of the "Reset Warm" command

Reinitialization for:

- Repeated download of the application
- Call of the "Reset Cold" command (in contrast to persistent variables)
- Call of the "Reset Origin" command

When you restart an application, its variables are usually initialized with explicitly preset initial values or with a default value.

Variables labeled with the keyword `RETAIN` are managed in the NVRAM memory of the control and retain their value even if the control fails in case of a power failure for example. One use case for the use of retain variables is for example a parts counter in a production unit that has to continue counting correctly after the power failure.

Memory location of persistent function block instances

Function block instances are stored "as one block" in the memory. This is necessary so that the same code can work on different instances. If a variable is marked with `RETAIN` in a function block, then each instance of the function block is protected with all variables. This is also true for the variables of the function block that are not marked this way. However, non-volatile memory is not available to the same extent as main memory.

A function block with a pointer to an instance in SRAM as a variable is not stored in the protected memory.

Refer to [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)

9.16.3 Preserving data with recipes

Variables are managed persistently in the Recipe Manager. The "Recipe manager" does not need any special memory on the controller in order to preserve values and data.

Declaration A recipe definition consists of a set of variables with values and is created and edited in the "Recipe Manager" object and saved to a file.

Functionality

- You can include a variable in multiple recipes, each with different values.

- In online mode, you can read in the actual values of the variables from the controller and save them as recipe values (specified value).
- You can use the "Recipe Management" library to programmatically implement the creation and editing of a recipe.
- You can save and backup a recipe as a recipe file.

Refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter ""Save current values to recipe" command" on page 678](#)
- [chapter ""Rearrange list and clean gapson page 678](#)
- [chapter ""Restore values from recipe" command" on page 679](#)
- [chapter 18.1.21 ""Recipe manager" object" on page 549](#)
- [chapter 18.1.22 ""Recipe definition" object" on page 552](#)

9.16.4 Declaring VAR PERSISTENT variables

The following steps provide an example of how to create a Persistent Variable List and how to declare a persistent variable in a Persistent Variable List and in a POU.

Prerequisites:

- A project is open and contains a program POU
- The option for textual view is enabled in the IndraWorks options **Tools ► Options ► IndraLogic 2G ► Declaration editor**

Example: Creating a Persistent Variable List and declaring a persistent variable

1. To add the **Global persistent variable list...** object to the application object, select **Add** from the context menu.

The "Persistent variables" dialog opens. If required, enter a name for the Persistent Variable List ("PersistentVars" by default).

2. Press "Finish" to confirm the dialog.

IndraLogic inserts the persistent variable list  **PersistentVars** below the application object in the Project Explorer.

3. Double-click in the Project Explorer to open the editor of the Persistent Variable List "PersistentVars".
4. In the editor, enter a variable declaration between `VAR_GLOBAL PERSISTENT RETAIN` and `END_VAR`
E.g. `ivarpersist1: INT;`
5. Double-click on the POU in the Project Explorer in which the persistent variable is to be declared.

The editor of the POU opens.

6. Enter the following declaration in the declaration section:

```
VAR PERSISTENT
ivarpersist2 : INT;
END_VAR
```

7. Select **Create ► Compile**.

The message window opens. If IndraLogic has compiled the application without errors, close the message window and continue with the next step. Otherwise, recover the error(s) and select **Create ► Compile** again from the context menu.

8. Go to the editor of the Persistent Variable List **PersistentVars** to select the Persistent Variable List as active editor.

Place the cursor in the declaration section and select the menu command **Add all instance paths**.

IndraLogic adds the persistent variable from the persistent variable list "PersistentVars" to the POU:

```
// instance path of the persistent variables created
POU.ivarpersist2: INT
```

Also refer to

- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter 18.1.12 "'Persistent variable list' object" on page 514](#)
- [chapter "Command 'Add all instance paths'" on page 679](#)

9.16.5 Saving the values of a persistent variable list in a recipe

Prerequisite: A project is open and there is a persistent variable list with declarations of persistent variables below an application object.

1. Select your application object in the Project Explorer and select **Set active application** from the context menu.

The application object is displayed in bold.

2. Select **Debug ► Login**

Your application is logged into the control and the application object in the Project Explorer is highlighted in green.

3. Double-click on the persistent variable list and select **Declarations ► Save current values to recipe** from the context menu.

IndraLogic creates the objects **Recipe Manager** and **PersistentVariables** under the application object.

4. Select **Debug ► Logout**.

The application is logged out from the control.

Also refer to

- [chapter ""Save current values to recipe" command" on page 678](#)
- [chapter 9.16 "Data persistence" on page 137](#)

Troubleshooting

F: Upon (initial) saving of current values in a recipe, saving is denied and an error message is output.

A: Check whether a "Library info" object ("General module folder") is used as "project information" in your project ([chapter 18.1.20 "'Library info' object" on page 545](#)). Disable the meta datum "Released" under "Summary" and save again.

9.17 Using function block for implicit check

IndraLogic provides special POU's that implement implicit monitoring functions. At runtime, they check the array limits or subrange types, the validity of pointer addresses or a division by zero.

1. Select the **Application** object in the Project Explorer.
Select **Add ► POU templates ► IndraMotion MLC base**
The **Add POU for implicit checks** dialog opens.
2. Select the functions.
3. Click on **Add**.
The selected POU's are inserted below the **Application** in the Project Explorer.
4. Open the POU's in the editor.
5. Adapt the implementation suggestion to your requirements.

CAUTION

Functionality of the monitoring functions

To obtain the functionality of monitoring functions, do not make any changes in the declaration section. However, adding local variables is the only exception.

Also refer to

- [chapter 18.1.18 "Monitoring the PLC program of the application at runtime" on page 542](#)

9.18 Object-oriented programming

IndraLogic supports object-oriented programming with function blocks and provides the following functionalities and objects:

- POU methods, interface methods
- POU properties and interface properties
- Interfaces
- Inheritance
- Method call, virtual function call
- Definition of function blocks as extensions of other function blocks

Also refer to

- [chapter "Method" object " on page 532](#)
- [chapter "Interface" object \(INTERFACE\)" on page 529](#)

9.18.1 Extension of a function block

The extension of a function block is based on the concept of inheritance in object-oriented programming. An inherited function block thereby "extends" a basic function block and in doing so is given the properties of the basic function block in addition to its own properties.

The extension of a function block means:

- The derived function block contains all data and methods defined by the basic function block.

An instance of the derived function block can always be used if IndraLogic expects a function block of the type "basic function block".

- The derived function block may overwrite the methods defined in the basic function block.

That means:

The derived function block can define a method with the same name, the same inputs and the same outputs as defined by the basic function block.



You have the following support when overwriting methods, actions, attributes, and transitions that are inherited by the base block:

When you insert a method, action, etc. below an inherited block, the "Add Object" dialog box includes a combo box with a list of methods, actions, etc. used in the base block. You can accept these and adapt them accordingly.

- The derived function block must not contain function block variables with the same names as used by the basic function block. The compiler reports this as an error.

The only exception: If a variable was declared as `VAR_TEMP` in the basic function block, the derived function block may define a variable with the same name. In this case, the derived function block can no longer access the variable of the basic function block.

- The variables and methods of the basic function block can be accessed directly within the scope of the derived function block using the `SUPER` pointer.



Multiple inheritance is not permitted!

Exception: A function block can implement several interfaces and an interface can extend other interfaces.

Also refer to

- [chapter 17.2.14 "SUPER" on page 291](#)
- [chapter 17.2.15 "THIS" on page 293](#)
- [chapter "'Method' object " on page 532](#)
- [chapter "Object 'Property'" on page 537](#)
- [chapter "Object 'Action' on page 540](#)
- [chapter "'Transition' object" on page 541](#)

Extending a basic function block with a new function block

Prerequisite: The open project is provided with a basic function block (e.g. `POU_1(FB)`) that is to be extended by a new function block.

1. Select the **Application** object in the Project Explorer and then select **Add ► POU** from the context menu.

The **Add object wizard** dialog opens.

2. Type the name for the new function block in the **Name** input field, for example `POU_Ex`.
3. Select **Function block**.
4. Click on **Advanced** and then on

5. In the category **Function blocks** under **Application** in the input assistant, select the POU(FB) to be used as basic function block, e.g. POU_1 and click on **OK**.
6. As an option, you can select an **Access modifier** for the new function block from the selection list.
7. From the combo box **Implementation language**, select for example **Structured text (ST)**.
8. Click on **Add**.

IndraLogic adds the POU_Ex function block to the Project Explorer and the editor opens.

In the first line:

```
FUNCTION_BLOCK POU_Ex EXTENDS POU_1
```

The function block POU_Ex extends the basic function block POU_1.

Extending a basic function block with an existing function block

Prerequisite: The open project possesses a base function block (example: POU_1 (FB)) and another function block (example: POU_Ex (FB)). The function block POU_Ex (FB) should also contain the properties of the basic function block. That means: POU_Ex (FB) is to extend POU_1 (FB).

1. In the Project Explorer, double-click on the function block POU_Ex (FB).

The function block editor opens.

2. Extend the existing entry in the top line `FUNCTION_BLOCK POU_Ex` with `EXTENDS POU_1`

The function block POU_Ex extends the basic function block POU_1

Also refer to

- [chapter "Function block" object \(FUNCTION_BLOCK\)" on page 523](#)

9.18.2 Implementing interfaces (IMPLEMENTS)

Implementing interfaces is based on the object-oriented programming. Different function blocks of the same kind can be used similarly using common interfaces.

A function block, which implements an interface, has to include all methods and attributes defined in this interface (interface methods and interface properties).

That means: Name, inputs and output of the respective method or property have to be the same.

When creating a new function block that implements an interface, IndraLogic adds all methods and attributes of the interface automatically below the new function block in the Explorer.



When adding more interface methods afterwards, IndraLogic does not add these methods automatically to the respective function blocks.

For this update, select the **Implement interfaces** command explicitly.

For inherited function blocks, ensure that any methods or attributes derived through the "inheritance" of an interface also receive the appropriate implementation. Otherwise, delete them if the implementation provided in the basis should be used. Respective compile error messages or warnings are displayed, prompted automatically by added pragma attributes. Refer to [chapter ""Implement interfaces" command" on page 699](#).



- Assign the interface of a function block to a variable of the interface type before a method can be called via the variable.
- A variable of the interface type is always a reference of the assigned function block instance.

A variable of the interface type is a reference to instances of function blocks. Such a variable can reference to every function block that implements the interface. If a variable is not assigned, the variable is 0 in online mode.

INTERFACE ITF_1 contains the method "GetName":

```
METHOD GetName : STRING
```

The function blocks FB_A and FB_B implement the interface ITF_1 each:

```
FUNCTION BLOCK FB_A IMPLEMENTS ITF_1
END_FUNCTION_BLOCK
```

```
FUNCTION BLOCK FB_B IMPLEMENTS ITF_1
END_FUNCTION_BLOCK
```

This means that the method "GetName" has to be present in both functions and is automatically attached below in the Project Explorer when the function blocks are created.

It has to be implemented separately for each function block.

Function blocks FB_A, method "GetName" (example)

```
METHOD GetName : STRING
  GetName:= 'FB_A';
```

Function blocks FB_B, method "GetName" (example)

```
METHOD GetName : STRING
  GetName:= 'FB_B';
```

Look at the "DeliverName" function with its input of a variable of type of the interface ITF_1:

Function "DeliverName": STRING

```
FUNCTION DeliverName : STRING
VAR_INPUT
  I_i: ITF_1;
END_VAR
DeliverName:= I_i.GetName(); // in the case, it depends on the "actual" type of I_i,
                             // if A.GetName or B.GetName is called.
```

Implementing an interface in a new function block

This input variable can receive all function blocks that implement "interface ITF_1".

- [chapter ""Implement interfaces" command" on page 699](#)

Prerequisite: The currently opened project is at least provided with one interface object.

1. Select **Application** in the Project Explorer and **Add ► POU** from the context menu.
The **Add object wizard** dialog opens.
2. Enter a name for the new function block into the input field **Name**, e.g. **POU_Im**.
3. Select **Function block**.
4. Select **Implemented** and press .
5. In the input assistant, select the interface from the category **Interfaces**, e.g. **ITF1** and click on **OK**.
6. To insert more interfaces, click on  and select a another interface.
7. Optionally, select an **Access modifier** for the new function block from the selection list.
8. Select from the selection list **Implementation language**, e.g., **Structured text (ST)**.
9. Click on **Add**.

IndraLogic adds the **POU_Im** function block to the Project Explorer and the editor opens. In the first line:

```
FUNCTION_BLOCK POU_Im IMPLEMENTS ITF1
```

The interface and its methods and properties are now inserted below the function block in the Project Explorer. The program code can be entered into the implementation section of the interface and its methods.

Implementing an interface in an existing function block

Prerequisite: The currently open project is provided with a function block (e.g. **POU_Im**) and at least one interface object (example: **ITF1**).

1. In the Project Explorer, double-click on the POU **POU_Im(FB)**.
The editor of the POU opens.
2. Extend the existing entry in the top line `FUNCTION_BLOCK POU_Im` with `IMPLEMENTS ITF1`.
The **POU_Im** function block implements the **ITF1** interface.

9.18.3 Extending an interface

Interfaces can be extended like function blocks. The interface also obtains the interface methods and interface properties of the basic interface in addition to its own.

Creating an interface that extends another interface.

1. Select the **Application** object in the Project Explorer.
2. Select **Add ► Interface**.
The **Add interface** dialog opens.
3. Enter a name for the new interface.

4. Select the **Extended** option and click on .
 5. The input assistant opens.
 6. From the category **Interfaces**, select the interface to be extended by the new interface.
- [chapter "Interface" object \(INTERFACE\)" on page 529](#)

9.18.4 Method call

To implement a method call, the actual parameters (arguments) are passed to the interface variables. As an alternative, the parameter names can be omitted.

Depending on the declared access specifier, a method can be called only within its own namespace (**INTERNAL**), only within its own POU and its derivatives (**PROTECTED**), or only within its own POU (**PRIVATE**). The method can be called from anywhere for **PUBLIC**.

Within the implementation, a method can call itself recursively, either directly with the **THIS** pointer with local variable for the assigned function block.

Method call as a virtual function call

Virtual function calls can occur due to inheritance.

With virtual function calls, the same call can call various methods in a program source code at runtime.

The method call is dynamically bound in the following cases:

- A method is called via a pointer to a function block (e.g. `pfub^.method`).
In this case, the pointer can point to instances of the type of the function block and to instances of all derived function blocks.
- Call a method of an interface variable (e.g. `interfacel.method`).
The interface can refer to all instances of function blocks that implement this interface.
- A method calls another method of the same function block. In this case, the method can also call the method of a derived function block with the same name.
- A method is called via a reference to a function block. In this case, the reference can point to instances of the type of the function block and to instances of all derived function blocks.
- Assign **VAR_IN_OUT** variables of a basic function block type to an instance of a derived function block type.
In this case, the variable can point to instances of the type of the function block and to instances of all derived function blocks.

Example: Overloading methods

The function blocks `fub1` and `fub2` extend the function block `fubbase` and implement the interface `interfacel`. There are the methods `method1` and `method2`.

```
PROGRAM PlcProg
VAR_INPUT
    b : BOOL;
END_VAR
VAR
    pInst : POINTER TO fubbase;
    instBase : fubbase;
    inst1 : fub1;
    inst2 : fub2;
    instRef : REFERENCE to fubbase;
END_VAR
```

```

IF b THEN
    instRef REF= inst1;          (* reference to fub1 *)
    pInst := ADR(instBase);
ELSE
    instRef REF= inst2;          (* reference to fub2 *)
    pInst := ADR(inst1);
END_IF
pInst^.method1(); (* If b is TRUE, fubbase.method1 will be called,
                   otherwise fub1.method1 is called *)
instRef.method1(); (* If b ist TRUE, fub1.method1 will be called,
                   otherwise fub2.method1 is called*)

```

Assuming that fubbase in the example above contains the two methods method1 and method2, overwrites fub1 method2, but not method1. method1 is called as follows:

```
pInst^.method1();
```

If b is TRUE, IndraLogic calls fubbase.method1. Otherwise,, fub1.method1 is called.

Also refer to

- [chapter ""Method" object " on page 532](#)
- [chapter 17.2.15 "THIS" on page 293](#)
- [chapter 17.2.14 "SUPER" on page 291](#)
- [chapter ""Function" object \(FUNCTION\)" on page 525](#)

Additional outputs

In accordance with the IEC 61131-3 standard, methods can have additional outputs declared, like normal functions. With the method call, you assign variables to the additional outputs.

Detailed information about this can be found in the topic "Function".

Syntax upon call

Program:

```

<function block name>.<method name>(<first input
name> := <value> (, <further input assignments>)+ , <first output name> => <first output variable name> (,<further output assignments>)+);

```

Example:

Declaration

```

METHOD PUBLIC DoIt : BOOL
VAR_INPUT
    iInput_1 : DWORD;
    iInput_2 : DWORD;
    sInput_3 : STRING(12);
END_VAR

```

Call

```
bFinishedMethod := fbInstance.DoIt(sInput_3 := 'Hello World ', iInput_2 := 16#FFFF, iInput_1 := 16);
```

When the method is called, the values of the method outputs are written to the locally declared output variables.

Calling a method even if the application is in STOP state

In the device description, it is possible to define that a certain function block instance (of a library function block) always calls a certain method in each task cycle. If the method contains the input parameters of the following example, IndraLogic processes the method even if the active application is presently in STOP state:

Example

```
VAR_INPUT
```

Programming applications

```

pTaskInfo :          POINTER TO DWORD;
pApplicationInfo : POINTER TO _IMPLICIT_APPLICATION_INFO;
END VAR
(*Now the status of the application can be queried via pApplicationInfo
and the instructions can be implemented: *)

IF pApplicationInfo^.state = RUNNING THEN <instructions> END_I
F;

```

Calling methods recursively


Use recursions mainly for processing recursive data types such as linked lists. Generally, we recommend that you be careful when using recursion. An unexpectedly deep recursion can lead to stack overflow and therefore to machine downtime!

A method can call itself within the implementation:

- Directly using the `THIS` pointer
- Indirectly by means of a local function block instance of the basic function block

Usually, a compiler warning is issued for a recursive call. If the method is provided with the pragma `{attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}`, then the compiler warning is suppressed. For an implementation example, see the "Attribute 'estimated-stack-usage'" chapter.

Also refer to:

- [chapter "Attribute 'estimated-stack-usage'" on page 385](#)
- [chapter 17.2.15 "THIS" on page 293](#)
- [chapter 17.2.14 "SUPER" on page 291](#)
- [chapter ""Method" object " on page 532](#)
- [chapter "Object 'Property'" on page 537](#)

10 Working with control networks

With the following functionalities, IndraLogic supports communication between controls (PLC) and the insertion of a safety control below a PLC:

- **Symbol configuration:** IndraLogic creates symbols with certain access rights for the variables in an application. Use these symbols to access the variables externally, e.g. from an OPC server.
- **Data source manager:** Manages the connection settings and the data transmission to removed devices (data sources). The transmitted data is mapped in data source variables. Access to the data is possible in the visualization or the local application. An example of a visualization is a control panel controlling removed devices and that visualizes the status of devices visualized as HMI application.
- **Network variables:** Network variables are variables whose values can be accessed by different control devices in the network. The variables have to be defined in rigid, identical lists in the sender device as well as in the receiver device. These lists are assigned to applications, but they can be located in different projects.
- A safety control can be inserted below a control in the Project Explorer. The communications links of the safety controls to the field devices, control networks, and development system are routed via this control.

Also refer to

- [chapter 10.2 "Symbol configuration" on page 156](#)
- [chapter 10.4.1 "Basics of the data source manager" on page 162](#)
- [chapter 10.3 "Network variables" on page 158](#)
- [chapter 10.1.1 "Network topology" on page 153](#)
- [chapter 10.1.2 "Communication in the Control Link via Gateway" on page 154](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)

10.1 Network and addressing

Design a control network hierarchically to ensure an extensive self-configuration.

In IndraLogic, the network topology is mapped to unique addresses and the routing algorithm is kept simple by structured addresses. There is direct and relative addressing and automatic address determination while booting the system.

Also refer to

- [chapter 10.1.1 "Network topology" on page 153](#)
- [chapter 10.1.2 "Communication in the Control Link via Gateway" on page 154](#)

10.1.1 Network topology

Information and recommendations
for the topology of a control network

It is recommended to set up a network system to allow the following:

- Extensive self-configuration (address assignment)
- Transparent support for every communication medium
- Transport of data packages between different networks

The routing mechanism should be so simple that each network node can re-route data packets, even with a low memory capacity. Therefore, avoid extensive routing tables, complex calculations or queries at runtime.

Design the control network hierarchically. Each node may possess a parent node and any number of child nodes. A node without a father is a "top level" node. Cycles are not permitted, i.e. each control network has the structure of a tree.

Father-child relationship results from the specification of certain network areas. A network area can be a local Ethernet or a serial point-to-point connection for example. We distinguish between the main network (main net) and the subnetworks (subnet). Each node belongs at the most to one main network, to which its parent node, if one exists, also belongs. For each node, any number of subnets can be configured, for which the node acts as a parent in each case.

A network area may only have one father node. Therefore, a configuration - in which a network area is defined as a subnet of several nodes at the same time - is invalid.

Also refer to

- [chapter 10.1.2 "Communication in the Control Link via Gateway" on page 154](#)

10.1.2 Communication in the Control Link via Gateway

The PLC communication from an engineering PC to one or several control devices (MLC or MTX system) is always established via an IndraLogic gateway.

the following figure shows the communication relation.

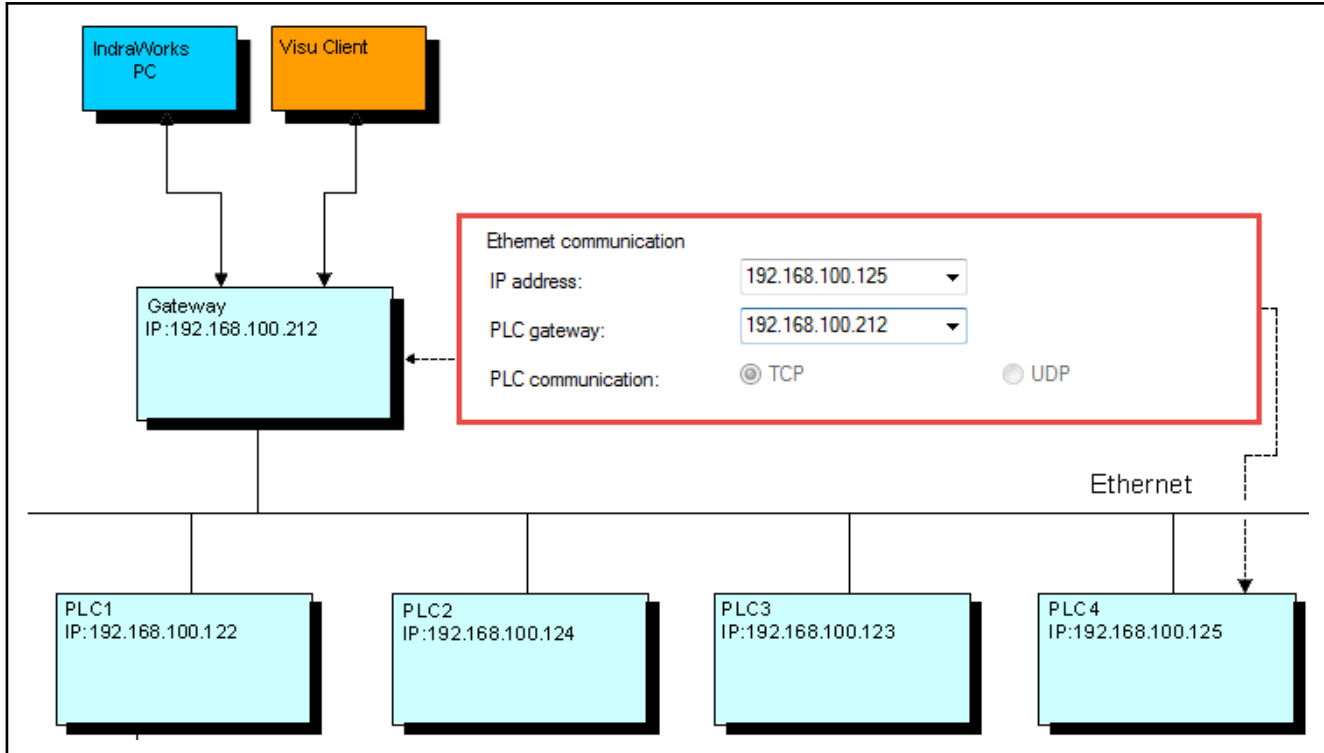


Fig. 10-1: Communication relations

The required settings take place in the respective device wizard.

An IndraLogic gateway can either run on the own engineering PC (setting: local host or 127.0.0.1) or issued in the control network (Setting: IP address of the gateway PC).

Fig. 10-2: Setting IP and gateway address

After a successful connection test, the IP address to the control is displayed (here 192.168.100.125). The TCP address can be directly set in the visualization device and in the OPC server.

If the PLC communication is not successful, investigate the following error causes. The error recovery is exemplarily shown under Windows XP. The solution can be different for other Windows versions.



A PLC communication to the device can only be established via the device-engineering interface!

Checking connection to gateway

If there is no gateway connection, the following message is output after a connection test:

- IndraLogic gateway not found.
- IndraLogic gateway: No communication. Gateway offline?

Ensure that the gateway server is running.

The gateway server is automatically started as service at system start.

Check whether the symbol  is displayed in the toolbar at the lower margin of the screen.

The gateway is stopped if the symbol is displayed as follows: .

The program symbol provides start and stop commands in a context menu opened via the right mouse button. The service can now be started and stopped at any time.

Ensure as well that the services <IndraLogic Service Control> and <IndraLogic V14 Gateway> run in the Windows **Control Panel** under **Administrative Tools - Services**.

Only one <IndraLogic Service Control> service and <IndraLogic V14 Gateway> service may run at a time.

Close possible further services and set the AutoStart type of these services to "Disabled".

If the connection test keeps on failing, change the settings for the IndraLogic gateway from <localhost> to <127.0.0.1> and subsequently set your own IP address.

Alternatively, the gateway of another engineering PC can be used by entering the IP address below the IndraLogic gateway in the device wizard.

Checking connection to device If there is no device connection, the following messages are output after a connection test:

- No connection to the device. Device offline?
- No connection to the device. Device offline? Error: <>

Check the control first. Did the control crash? Reboot the control.

Is a firewall enabled? Especially if the control is running on a PC. Disable the firewall.

Were changes made at the Gateway.cfg or GWClient.cfg file? Use the original files without modifications.

Was CoDeSys installed parallelly to IndraWorks? Close the CoDeSys simulation.

Incorrect response from the device

If the communication with the device is possible at all, device type or device version might not match. In this case, the following messages are output at a connection test:

Different device types:

- The selected target system does not match the connected device.
ID mismatch:
requested=1001 0003, online=1001 0103

Different device versions:

- The selected target system does not match the connected device.
Version mismatching:
Requested=14.12.0.0, online=13.11.4.0

Update the device firmware via the IndraWorks firmware management.

10.2 Symbol configuration

Use the symbol configuration for preparing symbols with specific access rights for project variables.

Use these symbols to access the variables externally, e.g. from an OPC server.

When generating code, IndraLogic also generates a symbol file (*.xml) that includes the description of the symbols.

The symbol file is saved in the project directory.

The name of the symbol file is composed as follows: <ProjectName>.<DeviceName>.<ApplicationName>.xml

Example:

```
proj_xy.PLC1.application.xml
```



Also generate the symbol file with **Create ► Generate code**. This is very useful when downloading to the PLC is not possible.

The variables that you export as symbols can be defined in the symbol configuration editor or in the variables declaration using the `{attribute "symbol"}` pragma.

The SFC editor provides another option:

Define the implicitly generated element variables to be exported to the symbol configuration in the "Properties" editor of an element.

The name of the symbol is generated in the symbol configuration acc. to the following syntax: `<ApplicationName>.<POU Name>.<VariableName>`.

When accessing the variable, you must always provide the complete symbol name in this syntax.

Example:

`MyApplication.PlcProg.a` or `MyApplication.SymFB.a`



As a rule, read-only access applies to symbols for input addresses and for variables that are mapped to input channels.

Write access is possible for testing purposes in simulation mode only.

The symbol file is loaded to the control together with the PLC.

Also refer to

- [chapter 18.1.24 ""Symbol configuration" object" on page 553](#)
- [chapter "Attribute "symbolon" page 410](#)
- [chapter "SFC element properties" on page 252](#)
- [chapter ""Applications" tab" on page 491](#)

Creating a symbol configuration

Prerequisite: The project can be compiled without errors

The **Symbol configuration** object is normally created automatically. It is also possible that it was deleted. The new creation is executed as follows:

1. Select the **Application** object in the Project Explorer.
2. Select **Add ► Symbol configuration** from the context menu.

The **Symbol configuration** object is added to the Project Explorer and the object editor opens.

3. Open the **View** menu of the editor and activate the categories of variables that should be provided in the configuration editor: Select the **Build** command in the symbol configuration editor.

All available variables (according to the currently defined filter in the **View** menu) are displayed in a tree structure.

4. Select the individual variables by selecting the checkboxes.



Please comply with the current settings; see "Settings" interface in the menu bar of the editor.

In the field below the menu bar of the editor, information is provided about the current situation with accompanying instructions, as well as controls for corrective actions.

5. Follow the prompt in the field below the menu bar. In the following case, this should be only the information that the modified symbol configuration is transferred with the next download or online change.

Click on **Build ► Generate code** in the main menu.

The file `<ProjectName>.<DeviceName>.<Application-Name>.xml` is created in the project directory.

IndraLogic applies the symbol configuration to the control during an application download or an online change.

Also refer to

- [chapter 18.1.24 ""Symbol configuration" object" on page 553](#)

10.3 Network variables

The values of network variables can be exchanged between different controlling devices in a network. The variables have to be defined in strict, identical lists on both the sender device and receiver device. However, only one device application may define the network variables. The lists can be in one or more projects.

The network variable list in the **sender** is a Global Variable List. Specific log parameters and transfer parameters are defined in its object properties.

Add these properties to generate a "Global Network Variable List (sender)" from a "regular" GVL.

An object  **Global Network Variable List (sender)** can also be inserted directly into the Project Explorer if these parameters are already set for this object.

The Network Variable List in the **receiver** is of type  **Network Variable List (Receiver)**.

Select the respective network variable list of the server upon creation. This variable list can also be loaded from an export file previously generated from the sender list. An export file is required if the sender list is defined in another project.

The network variables are transmitted as **broadcasting** in one direction only (sender to receiver). However, a device can also contain a sender as well as a receiver lists.

From version 3.5.7.0 of the `NetVarUdp` library, no reception channel is assigned anymore if no confirmed transfer is selected. Thus, network variables can also be exchanged between two controls on one hardware device.



- If the exchanging devices are to be used as **senders AND receivers**, the variable list IDs have to be unique to prevent an abnormal operation! The variable list IDs are defined in the **Properties** dialog of a GVL object.
- Data exchange via network variables is not possible if
 - it is not supported by the device (target system)
 - a firewall blocks the communication
 - another client or application uses the UDP port set in the properties of the network variable list
- Only arrays with limits defined with a literal or a constant are transmitted to the receiver application!
Constant expressions are not allowed for this purpose!
Example:

```
arrVar : ARRAY[0..g_iArraySize-1] OF INT ; is not transmitted
```

```
arrVar : ARRAY[0..10] OF INT ; is transmitted.
```
- The maximum size permitted of a network variable is 255 bytes. The possible number of network variables is unlimited.
- If the size of the Global Variable List (GVL) exceeds the maximum length of the network telegram, data is distributed on multiple telegrams. Depending on the configuration, data inconsistencies can result.



A communication using network variables is also possible when the controls operate with applications from **different versions of the programming system** (IndraLogic 1x).

However, in this case, the export/import mechanism cannot be used for the matching variable lists in the sender and receiver projects.

Reason:

A variable export file (*.exp) that can be generated from IndraLogic 1x does not include the information necessary to create a receiving NVL in IndraLogic 2G. There is no network parameter configuration as *.gvl file, which was previously exported from the sender.

For this file, "create" the 1x NVL in 2G first.

Then, generate an export file and create a receiving NVL in 2G.



An alternative to data exchange between controls is the use of a **data sources**. As opposed to the broadcasting method for exchanging network variables, defined point-to-point connections are created between an application and a remote data source.

Also refer to

- [chapter "The "Properties"- "Network variables" dialog" on page 708](#)
- [chapter 18.1.11 ""GVL" object - Global Variable List" on page 512](#)

10.3.1 Configuring a network variable exchange

1. Creating a Network Variable List in the sender device and generating an export file

The following steps are necessary for exchanging network variables between the sender device and receiver device:

Prerequisites: An application is inserted into the Project Explorer of the control device (sender device) used.

1. Select the application and insert a **Global Network Variable List (sender)** object.

Make the following settings in the **Add Global Network Variable List (sender)** dialog:

Network type: UDP, example:  NVL_Sender.

2. Double-click on the NVL object to open the respective editor and enter the declarations of the network variables.

Example of NVL_Sender

```
VAR_GLOBAL
    iglobvar:INT;
    bglobvar:BOOL;
    strglobvar:STRING;
END_VAR
```

3. Select the NVL object in the tree open the **Properties** from the context menu. Open the **Network variables** tab in the **Properties** dialog.

It shows the following settings:

Network type UDP; variable list ID 1, pack variables, cyclic transmission every 50 ms.

- 4.



An existing GVL can also be converted into a Network Variable List by configuring its network variable properties accordingly.

5. Click on the **Link to file** tab in the **Properties** dialog of the **NVL_Sender**. Define a file name `<export>.gvl` and a storage location in the file system for the export file of the GVL. Select the option **Export before compile**.
6. Select **Create ► Build** to compile the application.

The export file for the Network Variable List is now located in the defined folder.

Also refer to

- [chapter "The "Properties"- "Network variables" dialog" on page 708](#)

2. Creating a respective Network Variable List in the receiver device

Prerequisites:

There is a sender device and a receiver device in the Project Explorer.

An application with a task configuration is inserted below both devices.

An NVL or a GVL is created below the sender device as Network Variable List to be sent.

1. Select the application of the receiver in the Explorer and click on **Add ► Global Network Variable List (receiver)**.

The **Global Add Network Variable List (receiver)** dialog opens.

2. In the dialog, select the previously created NVL of the sender device and enter a name:

Example:  NVL_Receiver.

IndraLogic populates this recipient list automatically with the variable declarations from the sender list.

Automatically generated: NVL_Receiver

```
//This global variable list is received via the network.
//Sender: NVL Sender [IndraMotionMlc1: Logic: Application]
//Protocol: UDP
```

```
VAR_GLOBAL
  iglobvar:INT;
  bglobvar:BOOL;
  strglobvar:STRING;
END_VAR
```



As an alternative, select the **Import from file** option and load the previously generated export file from the sender list.

3. Testing the network variable exchange

Prerequisites: There is a Network Variable List (sender) in the sender device and a Network Variable List (receiver) in the receiver device. Both lists have identical variable declarations.

1. Below the application in the sender device, create a program that increments a network variable.
For example: `NVL_Sender.iglobvar:=NVL_Sender.iglobvar+1;`
2. Configure the application task so that this program calls it.
3. Below the application in the receiver device, create a program that writes the value of this network variable to a local variable.
For example: `ilocalvar:=NVL_Receiver.iglobvar;`
4. Configure the application task so that this program calls it.
5. Load both applications to the controls and start them. (Set application to active, log in, load, start)
6. In the online views of the editor of both programs, check whether the values of `iglobvar` in the receiver match those in the sender.

Configuring a network variable communication between 1.x controls and 2G controls

Prerequisites: There is a Global Variable List in a 1.x project with network variables to be read in your 2G project.

In the 2G project in the Project Explorer, there is an application with a program, which uses the variables, below the control.

1. Open the **Properties** dialog of the GVL in the 1.x project (sender).
2. Insert another control into the project as dummy with an application and a task. Then, create the 1.x GVL below it as follows:
3. Below the dummy application, insert a **Global Network Variable List (sender)** object named "dummy1x".
4. Go to the context menu of **dummy1x** to open the **Properties** dialog and click on the **Network properties** tab. Enter the same network properties as the GVL in the 1x project.
5. Fill **dummy1x** with the same variable declarations as present in the 1.x GVL.
6. Define a storage location for an export file "dummy1x.gvl" in the **Link to file** tab of the **Properties** of **dummy1x**.

7. Set the application of the dummy control to active and press <F11> for compilation. IndraLogic creates the export file.
8. Set the application of the real control to active and select **Add** from the context menu.
9. Select **Global Network Variable List (receiver)** and specify a name, e.g. "NVL_from_1x".
10. Select the option **Open from file** and then the previously generated `dummy1x.gvl`.
11. Check the newly created **NVL_from_1x**. It has to contain the same variable declarations as those in the sender GVL in the 1.x project.

Also refer to

- [chapter 18.1.15](#) "'Global Network Variable List (sender)" object" on page 519
- [chapter 18.1.16](#) "'Global Network Variable List (receiver)" object" on page 519

10.4 Data Source Manager

An application can establish a point-to-point connection to one or multiple remote data source(s) using the data source manager.

The variable values from other control applications can thus be used in the "local" application, typically in a visualization.

The data source manager refreshes the values of the variables at configurable time intervals.

Also refer to

- [chapter 10.4.1](#) "Basics of the data source manager" on page 162
- [chapter 10.4.2](#) "Adding initial data source" on page 164
- [chapter 10.4.3](#) "Editing data source variables" on page 167
- [chapter 10.4.5](#) "Updating data interface" on page 170
- [chapter 10.4.6](#) "Using remote data" on page 171
- [chapter 18.1.3](#) "'Data source manager" object" on page 482

10.4.1 Basics of the data source manager

In order to have read/write access to remote devices and their running applications, you can add a data source manager to your application with one or more data sources.

The functionality of the data source manager allows for establishing connections and communication to remote devices, and it makes its data available through data source variables. At this time, the partners communicate by means of a point-to-point connection.

During the development phase, a variable list can be created offline by means of local symbol configuration files. Thus, a local application can be developed offline without connection to the data source using symbol information.

Refer to [chapter 18.1.3](#) "'Data source manager" object" on page 482

Data source type

The communication between remote and local devices communicate by means of data source types. The following data source type is available in IndraLogic:

"CODESYS ApplicationV3": Data is accessed by means of address monitoring.

This requires that the address information between the remote control and the local device match. Disadvantage: However, for changes to the remote application, the local application (for example the HMI application) is updated! The advantage is that a symbol configuration is not required in the remote application.



Use this communication for embedded or mini controls when there are no available resources for the symbol configuration.

Refer to ["Dialog 'Add data source'" on page 482](#)

Connection type

The functionality of the data source manager allows for establishing a point-to-point connection from a local device to one or more remote devices (even through networks). IndraLogic supports the connection type "CODESYS Application V3".

For this connection type, the runtime system of the local application needs valid communication parameters to establish the connection.

The network scan function can support you when configuring the data source.

Refer to [chapter 18.1.6 "'Communication' tab" on page 486](#)

Data transfer

When the local application is in runtime mode, the data source variables that appear in the data source editor of the "Variables" tab are updated in configurable time intervals. The remote application is also executed at this time. Variables that are configured in the visualization, in the trend, as alarms, or for recipes are transferred and stored automatically. When a variable is accessed in IEC code only, the variable is not updated automatically. In this case, you must activate the "Update always" option in the data source editor of the "Variables" tab.

The data source types support the (read or write) data access to variables of the source control for the following data types:

- Scalar value at highest level, example `PLC_PRG.hugo`
- Property to a program or GVL by means of a call when it is marked with `{attribute monitoring := ,call'}`.
Example: `PLC_PRG.PropertyCall`
- Variable that is mapped to bit addresses. Example in `PLC_PRG`: `x AT %MX0.5 : BOOL;`
- Variable (type BIT) in a function block, example: Declaration in `DUT`:
`x, y: BIT, Bit access: PLC_PRG.dutInst.y`
- Structured obtainable variable, example `PLC_PRG.outerInst.innerInst.dwVar`
- Property to block instance when it is marked with `{attribute monitoring := ,call'}`.
Example: `PLC_PRG.instance.PropertyCall`
- Property at highest level and to an instance when it is marked with `{attribute monitoring := ,variable'}`.
Note: This cannot be written by monitoring or by the data sources.
- Array access with literal index, example: `PLC_PRG.arrOfInts[3], PLC_PRG.inst.arrOfBool[1]`

- Nested access (for example, array of structures or structure of arrays)

⚠ CAUTION

Bit access used in visualizations that are transferred via a data source connection function only if they contain literal offset specifications.

A visualization cannot process an offset specification by defined constants.

See ["Variables" tab" on page 483](#)

10.4.2 Adding initial data source

To exchange data between your local device and a remote device, add a "Data source manager" object and then a "Data source" object in the Project Explorer below the "Application" node. An assistant guides you through the configuration of the data source. Then you can modify the settings in the editor of the object at any time.

When adding a data source, select a data source name. Then specify the connection settings of the point-to-point connection to the remote device. Ideally, the remote device is being executed in the meantime and IndraLogic can establish the connection immediately. In this case, all available data source variables are displayed from the remote control. Select the variables that should be transferred. You can also select all variables. Then the data source is initialized automatically, the data source variables are created below the "DataSources_Objects" folder, and an additional data source is added below the data source manager.

The initial settings can be modified at any time in the editor of the data source.

Refer to [chapter 18.1.3 "Data source manager" object" on page 482](#)

Initially connecting devices with address monitoring

Prerequisites:

The remote device and its application are running.

The IndraLogic project of the remote device is located on your development computer.

The developed application there does not contain a symbol configuration.

1. Insert a "Data source manager" object below your "Application" node in Project Explorer.
2. Select the "Data source manager" object and click on the context menu item **Add ► Data source**.

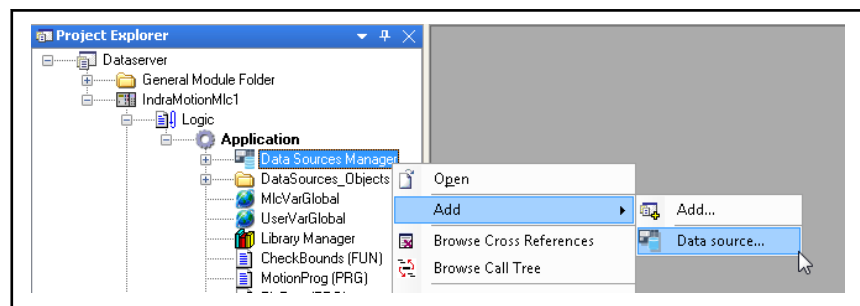


Fig. 10-3:

The "Add data source dialog" opens.

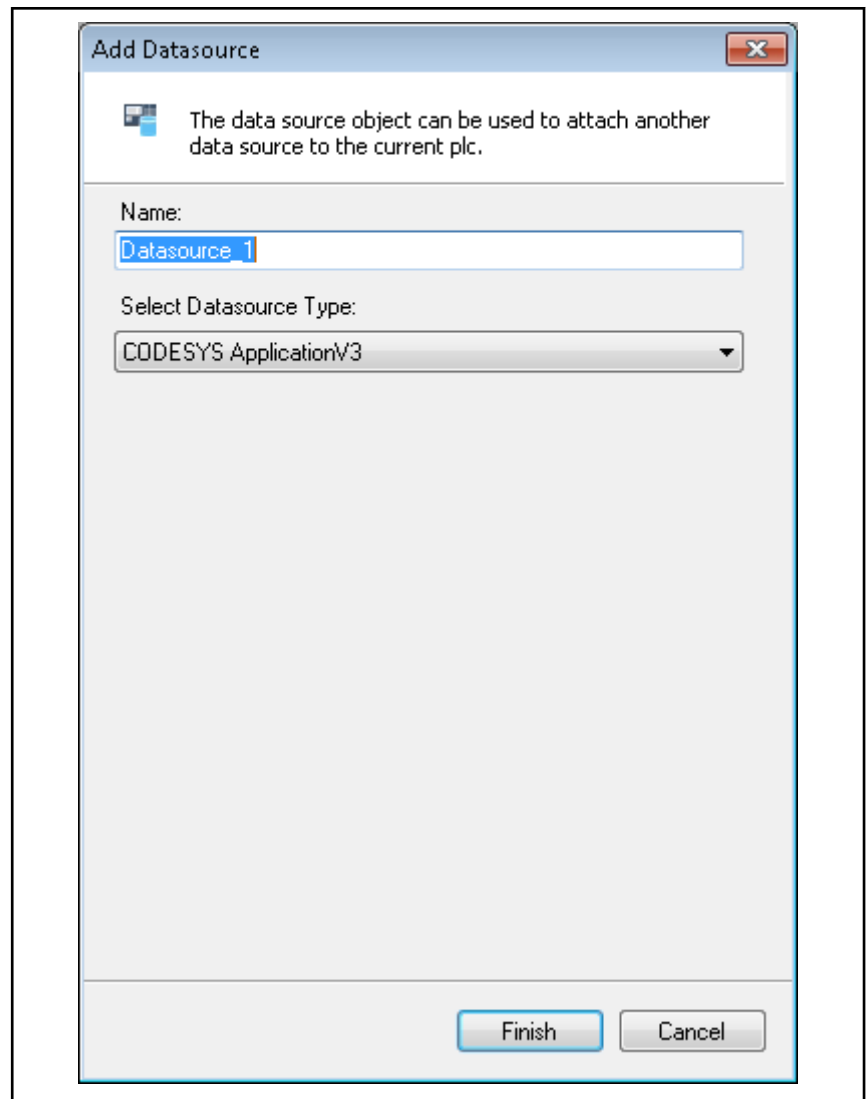


Fig. 10-4:

3. Enter the data source name under "Name".
Selecting the data source type is preset to "CODESYS ApplicationV3" and cannot be changed.
With this setting, data is transferred via Adressmonitoring.
Confirm with "Finish".
The "Initialize data source wizard - Provider settings" dialog opens.
4. Select the "Other project" option for "Select project type" (1).

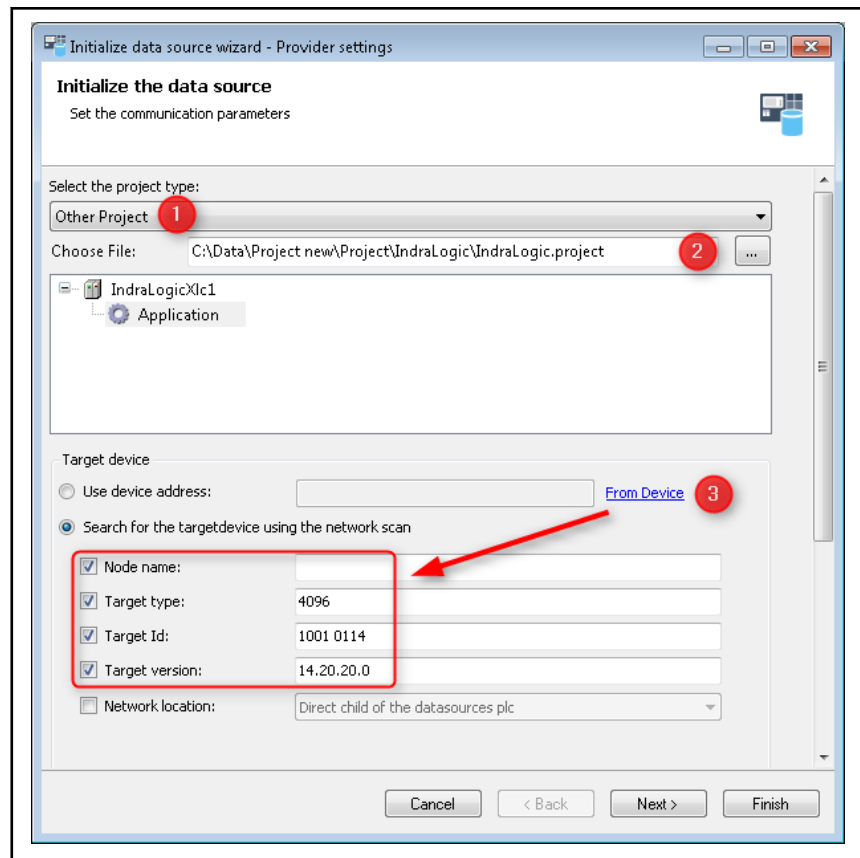


Fig. 10-5:

5. For "Choose file", specify the file and storage location of the IndraLogic project on the remote device (2).

Example: C:\Data\Projects\Project\IndraLogic\IndraLogic.project.

Note: The IndraLogic project is part of the IndraWorks project

The remote device is displayed in the tree view of the window below, and as a result the connection was established.

6. Select the link "From device" (3).
The connection parameters to the remote device are read and displayed in the dialog. The connection is configured.
7. Click "Next>".
The "Initialize data source wizard - Search data source variables" dialog opens.
Below the "Variables" item, the remote control variables are displayed in the tree view.

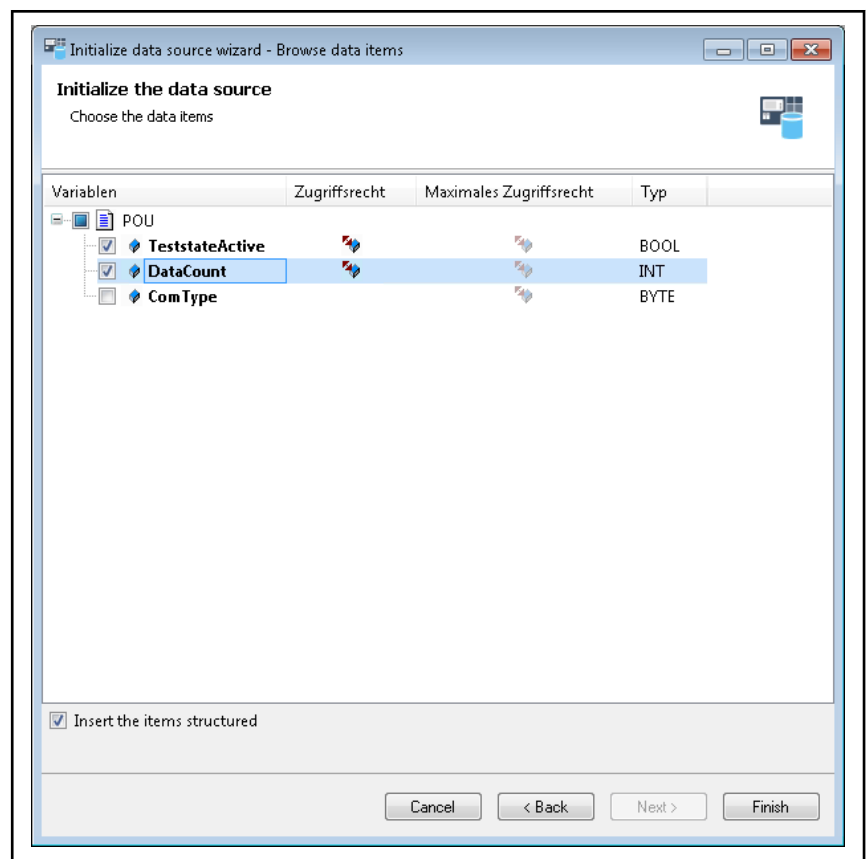


Fig. 10-6:

8. In the tree view, select the control variables to be transmitted, and click "Exit"

The data source is configured. A connection is established. The settings are stored in the object and can be modified in the editor of the object.

See:

- [chapter 10 "Working with control networks" on page 153](#)
- [chapter 18.1.6 "'Communication" tab" on page 486](#)

10.4.3 Editing data source variables

In runtime mode, the remote data is saved to the data source variables. The data source variables and their mapping to the remote variables are displayed in the data source editor below of the "Variables" tab.

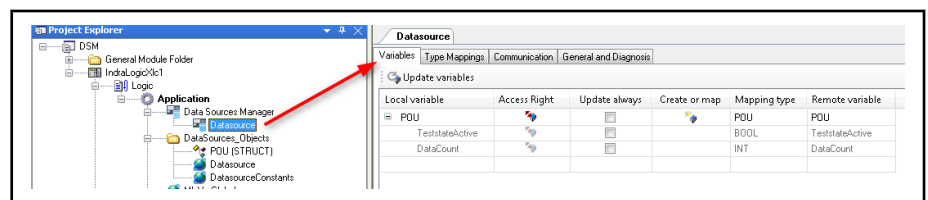


Fig. 10-7:

If the local and remote variables have the same names and the same data types, then the data is mapped 1:1. The variables and the data types are created automatically. That is the regular procedure.

You can also map to existing variables. For example, this is required if a visualization contains a data type in the interface. The same data type has to be

transferred to this visualization. In this case, the declared local variable and the remote variable have the same data type, e.g. of a library. Moreover, you can map a local variable with a conforming data type to a remote variable. The data type can be created in the "Type mapping" tab.

The specifically created variables and data types are declared in the "Data-Sources_Objects" folder.

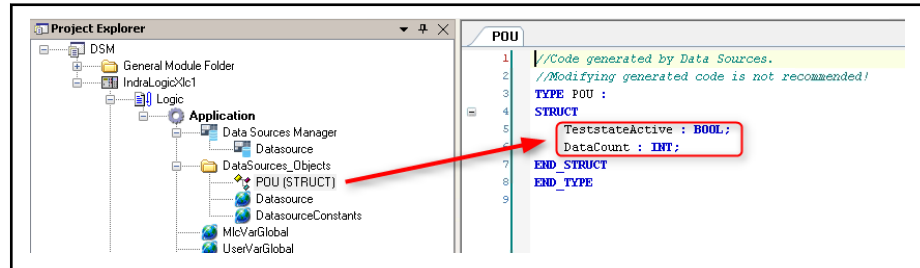


Fig. 10-8:

For each data source, a global variable list of the same name as the data source is declared there. Moreover, the data source variables usually have the identical or conforming data type as the remote control variable and they are declared as user-defined data types (DUT objects). Considering all data sources, multiple declaration of the same data types is avoided.



Do not edit the data interface in the "DataSources_Objects" folder manually.

It is created initially when adding a data source.

Changes can be made later in the editor of the data source.

Refer to

- [chapter 10 "Working with control networks" on page 153](#)
- [chapter 10.4.5 "Updating data interface" on page 170](#)
- [chapter 18.1.4 "'Data source' object - 'Variables' tab" on page 483](#)

Selecting variables for data transfer

The selection of the data source variables can be edited.

Prerequisites:

The remote device and its application are running.

A data source manager is already inserted below the local application with a data source.

1. Open the editor of the data source.
2. Select the "Variables" tab.
3. Click "Update variables".

The "Browse variables" dialog opens.

4. Activate the variables that should be transferred and click "OK" to close the dialog.

The data source variables are modified according to the selection. The declaration of variables and data types is also modified.

The "Variables" tab shows the modified selection. Moreover, the mapped remote variable is listed in the "Remote variable" column.

Refer to

- [chapter 10 "Working with control networks" on page 153](#)

Mapping remote variables to local variables with a conforming data type

- [chapter 10.4.5 "Updating data interface" on page 170](#)
- [chapter 18.1.4 "'Data source' object - 'Variables' tab" on page 483](#)

Symbol: 

First, create a conforming data type and then use it for a variable.

Prerequisite:

A data source manager and a data source below it are located in the project of the local application.

The remote data that should be transferred is displayed in the editor of the data source in the "Variable" tab

1. Open the editor of the data source.
2. Select the "Type mappings" tab.
3. Select the data type to be edited in the list.

The elements of the data type are listed in the window below the data type list

4. Specify a name for the data type.

Select the name for the remote data types to which the local type should conform.

5. Modify it in the window below the data type list and remove the elements that are not necessary for the data transfer.

6. Select the  symbol for this data type in the "Create or map" column.
The data type is declared in the "DataSources_Objects" folder.

7. Select the "Variables" tab.

8. Specify a name in the "Local variable" column.

9. Under the "Generate and map" column, select the  symbol

10. Enter the data type created in step 6 into the "Type mapping" table.

11. Select the remote variable with the data that should be transferred. Use the input assistance for this.

The local variable declared in step 8 is automatically declared with the user-defined data type of the removed variable. During data transfer, it receives the data of the mapped remote variables.

The operating sequence demonstrates how variables of the data source are created. At this time, new variables are created, data is mapped to existing data types and their variables, and new data types are created with type-conforming mapping.

Refer to [chapter 10 "Working with control networks" on page 153](#)

10.4.4 Editing communication

To edit the connection parameters of a data source, go to the data source editor in the "Communication" tab. The status line in the lower part of the editor informs on the data source type, the IndraLogic project directory and the current connection type. The connection type can subsequently not be changed.

Refer to

- [chapter 10.4.2 "Adding initial data source" on page 164](#)
- [chapter 18.1.6 "'Communication' tab" on page 486](#)

10.4.5 Updating data interface

The data source variables are updated cyclically at runtime. Only data currently used in the visualization or with the property "Update always" is updated.

You can define the time interval. Moreover, you can define variables whose data is transferred in each update interval, and therefore they are always up-to-date. To update variables that are not used in the application code, you can implement an update programmatically with the help of interface functions from the data source manager.



If data traffic between the remote and local device is too high, then the refresh rate is reduced automatically.

This can cause an incomplete transfer.

Refer to [chapter 18.1.4 "Data source" object - "Variables" tab](#) on page 483

Setting the refresh rate

1. Open the editor of the data source.
2. Click on the "General and Diagnosis" tab.
3. Specify a value in the "Refresh rate" field.

Example: 100

The data from the remote device to the local device is transferred every 100 ms.

Refer to [chapter 18.1.7 "Data source" object - "General and Diagnosis" tab](#) on page 488

Selecting the variable for "Update always"



Avoid to always update too many variables.

Each update produces additional data traffic at the connection between the remote and local devices.

When data traffic is too high, the refresh rate is reduced automatically.

This can cause an incomplete transfer.

1. Open the editor of the data source.
2. Activate the option "Update always" for a variable.

The data of the variables is transferred at each update cycle, even when the data has not changed.

Refer to ["Variables" tab](#) on page 483

Updating data programmatically

The data source manager provides interface functions in the `Datasources` library. If a data source manager is integrated in the application code, then the global variable `g_Datasources` is instantiated automatically. This provides access to the interface functions.

Then you can update individual variables that are not called in the active visualization.

Example:

The variable `ivar` is activated and deactivated by means of methods from the `Datasources` library so that its value is transferred. Furthermore, you can configure that the variable is updated only over a defined duration in order to save transfer capacity.

Program:

```
//Synchronize with DatasourcesTask and block
until access is possible
//Regard the feedback in ERR_OK or in
ERR_DS_MULTITASKING_LOCKED
g_Datasources.BeginDataConfiguration(TRUE);
// Activate variable
g_DataServer.UseData(ADR('RemoteDevice.Applicat
ion.PLC_PRG.iVar'));
// Deactivate variable
//
g_DataServer.ReleaseData(ADR('RemoteDevice.Appl
ication.PLC_PRG.iVar'));
g_DataServer.EndDataConfiguration();
```

The data configuration is started with `BeginDataConfiguration(TRUE)`, thus initializing the synchronization of the task `DatasourceTask` with the application task. The value `TRUE` blocks the processing until the access to the variable is possible; `FALSE` repeats access attempts without blocking. The return values `ERR_OK` and `ERR_DS_MULTITASKING_LOCKED` provide feedback about the access attempts.

When synchronization is successful, the variable is activated by means of the `UseData` method. Then the data configuration is completed with the `EndDataConfiguration` method and the synchronization triggered again with the task `DatasourceTask`.

The `ReleaseData` method is used in the same way for deactivating the variable again at the desired processing time.

Refer to [chapter 10 "Working with control networks" on page 153](#)

10.4.6 Using remote data

The variables that are listed in the data source editor of the "Variables" tab (and declared in the "DataSources_Objects" folder) can be used in your application like IEC variables.

One use case is to display the variable using a visualization element.



If multiple data sources are available and therefore conflicts occur regarding unique variable names, then you must specify the data source name as the prefix. If no conflicts occur, then this is not necessary and you can map the variables without a data source prefix.

`<DataSourceName>.<FunctionBlockName>.<VariableName>`

Outputting a variable value of a remote device in a visualization

Procedure to output the variable value of a remote control (e.g. `iTemp`) in the local application using an IndraLogic visualization with a visualization element (with data source manager).

Initial situation:

A data source "dsRemotePLC" is created below the local data source manager where the connection to the remote device is configured. In addition, the variable `iTemp` is selected in the data source editor of the "Variables" tab.

1. Select the visualization element in the editor window. Select the property "Text variables" - "Text variable" in the "Properties" window.
2. Select the variable `iTemp`.

The variable mapping is qualified.

Example: `dsRemotePLC.PLC_PRG.iTemp`

3. Select the "Text" property of the visualization element and enter the following: `Temperature: %s`
The value of the `iTemp` variable from the remote device `RemoteDevice` is displayed.
4. Download and start the remote application.
5. Download and start the local application.
The visualization starts and displays the actual value of `iTemp`.



The visualization integrated in IndraLogic does not display actual values of variables that are transferred by means of a data source connection. The integrated visualization displays only the initialization values or the last otherwise delivered values because they do not establish a connection to the data sources.



If variables are used that are not called in the visualization code, then the variables must be updated in the application code by means of functions from the data source interface.

10.5 Subordinate safety control

If a safety control is below the standard control, the standard control is used to communicate and exchange data between the safety control and the programming system.

The communication links of the safety controller can interrupt the execution of commands that affect the standard controller.

There is a note for each command.

Possible interruptions

- **Temporary interruption:**

During the execution of the command (for example: download), the connections with the safety controller are interrupted first and then are automatically available again afterwards.

If the interruption time is too long, then safety-oriented reactions can occur in the output devices and connected network variable receiver safety controllers.

The corresponding communication errors have to be acknowledged in the safety control to end the safety-oriented reactions (if there is no automatic acknowledgement).

This affects the connection to the field devices of the safety control and the network variable receiver connections to other sender safety controls. In the case of a connected safety controller with network variable senders, the communication errors must be acknowledged in the other safety controllers.

- **Permanent interruption:**

The execution of commands (for example: delete) leads to an interruption that ends again by another action (for example: download).

As a result of the interruption, safety-oriented reactions can occur in the output devices and connected network variable receiver safety controllers.

After ending the interruption, the corresponding communication errors must be acknowledged in the safety controller to end the safety-oriented reactions (if there is no automatic acknowledgement).

For a subordinate safety control, it is routed via the routing runs via "<Name of the SafetyApp> Mapping". In some cases, it can happen that the user can see this application in the Project Explorer.

** CAUTION**

No commands may be executed in the application "<Name of SafetyApp> Mapping".

Also refer to

[chapter 10 "Working with control networks" on page 153](#)

11 Transferring applications to the PLC

To transfer your application to the control, the program has to be compiled without any errors and the connection settings have to be set for the control.

If these prerequisites are met, the application is loaded to the control upon login.

11.1 Configuring the connection to the PLC

A gateway is used to connect to the control. This gateway can be your development computer or any network computer connected to the control.

Set the communication channel either when **adding** the control to the project in the dialog "Step 2: Hardware/communication" or at a later point via **<Control> ► Properties**, tab page "Hardware/communication".



Check the connection via **Execute**.

Also refer to

- [chapter 10.1.2 "Communication in the Control Link via Gateway" on page 154](#)

11.2 Generating application code

The application code is the machine code executed by the control when starting an application.

IndraLogic automatically generates the application code from the source code written in the development system before downloading the application to the control. Before the application code is generated, assignments, data types and library availability are checked. In addition, the memory addresses are allocated when generating the application code.

To explicitly execute this command, select **Build ► Generate code**. This is useful to detect errors in the source code, even if the control is not yet connected. The errors are output in the messages view in the "Build".

Explicit generation of application code

Prerequisite: The application can be compiled without errors.

1. Select **Build ► Generate code**.

The application code is generated. Detailed information on memory allocation is output in the message window.

Also refer to

- [chapter 11.4 "Generating boot applications" on page 177](#)
- [chapter ""Generate code" command" on page 611](#)

Messages when generating application code

When generating application code, IndraLogic outputs information on memory allocation in the message window. Due to the incremental compilation, memory is only allocated again for changed and new function blocks and variables and thus, gaps arise in the memory. Online changes have the same effect. This fragmentation reduces the available memory. To completely re-allocate the memory in this case, select **Create ► Clean** and thus increase the free memory.

For more information on the messages during code generation: Syntax errors and error identified by IndraLogic during the code generation and memory allocation are output in the message window in the "Build" category.

After "cleaning" the application, the memory is newly assigned. In this case, it is possible that the specified "alignment" (in rule 8) causes small gaps. Bigger gaps result when changing a data without "cleaning", e.g. when increasing the array area. In this case, only the affected function blocks are recompiled. Online changes have the same effect.

Please note the options for generating applications.

Also refer to

- [chapter ""Properties" - "Application build options" dialog" on page 707](#)

11.3 Loading application code — Logging in and starting the PLC

To download the source code of your application to the control, log in to the application on the control. If there are several applications (on multiple controls) in the project, explicitly switch the correct application to active first.

When loading an application to the control, IndraLogic performs the following checks:

The list of applications on the control is compared to those available in the project. If they do not match, you are prompted whether to load the application that is not yet on the control or to delete existing applications.

For "externally implemented" function blocks in the application to be loaded, IndraLogic checks whether these are available on the control. If they are not available, the message "unresolved reference(s)" is output in a message box and in the message window.

IndraLogic subsequently compares the parameters (variables) of the function blocks in the application to be loaded and the parameters of the function blocks with the same name in the application already present on the control (signature¹⁾ check). If they do not match, a corresponding message ("Invalid signature(s)") is output in a message box and in the message window.

If the option "Download application info" is enabled for an application under **Properties**, tab **Application build options**, information on the application content is additionally loaded to the PLC.

Also refer to

- [chapter 18.1.2 ""Application" object" on page 480](#)

Transferring the application and starting the program

Prerequisite: The application has no errors and the communication settings of the control are correct. The application does not yet exist on the control.

1. Select the required application in the Project Explorer. Continue with step 3 if there is only one application.
2. Select **Set active application**.
The application is displayed in bold.
3. Select **Debug ► Login**
A dialog is displayed, prompting whether to create the application.
4. Confirm the dialog with **Yes**
The application is loaded to the control
5. Select **Debug ► Start** or press <Alt>+<F5>.
The application is running on the control.

¹⁾ Name/type of the function block, name/type/sequence of the inputs and outputs

11.4 Generating boot applications

A boot application is the application that is started automatically when the control is switched on or started.

Thus, the application on the control has to be in a file `<Application-Name>.app`.

For each application that is running on the control, a boot application can also be stored there.

By default, IndraLogic generates the boot application automatically when an application is downloaded and transfers it to the PLC. For the default settings for the automatic generation, go to the **Properties** of the application object, category **Boot application**. When logging in with a changed application, you are still prompted whether to generate a new boot application.

To create a (new) boot application at any time in online mode, select **Debug ► Generate boot application**.

A boot application can also be generated and saved locally in offline mode. External tools can then be used to copy this application to the control. This transfers an application to the control, even if it is not connected to IndraLogic.

Generating boot applications on the control automatically and explicitly

Prerequisite: Offline mode. The application is compiled without errors. The connection to the control is configured and the control is running. The application is active. The following steps demonstrate the options:

1. Select **Debug ► Login**.

The boot application file `<ApplicationName>.app` is generated on the control next to the checksum of the boot application `<ApplicationName>.crc`.

2. Select **Debug ► Generate boot application** explicitly.

The files on the control are replaced by new files.

3. Log out.

4. Change the application. Log in again to the control.

It is prompted, whether to perform an online change. The **Update boot application** option is shown in the same dialog. It is disabled by default. However, the default settings can be changed in the **Properties** of the application object, category **Boot application**.

5. Do not enable the option and continue with the login.

No new boot application files are generated.

6. Log out. Close the project. Stop the control. Restart the control.

The boot application created above is running on the control.



An encrypted boot application can be saved on the control. These settings are defined in the **Properties** of the application object, category **Application build options**.

Also refer to

- [chapter "Properties" - "Application build options" dialog](#) on page 707
- [chapter "Properties" - "Boot application"](#) on page 705

Transferring applications to the PLC

Creating boot applications in off-line mode

- [chapter 14.1 "Executing an online change" on page 213](#)
- [chapter "Command 'Create boot application'" on page 617](#)

Prerequisite: Offline mode. The application is compiled without errors. To generate a boot application for an application and to save it in the file directory for copying it later to the control without IndraLogic, but using external tools instead.

1. Select **Online ► Generate boot application**.
A dialog opens to specify a memory location in the local file system.
2. Select a storage path and click on **Save**.
3. If the application has changed since the last boot application was generated, you are prompted to use a new code for the boot application. In this case, click on **Yes**.
The **Save as** dialog opens.
4. Select a directory and click on **Save**.
The boot application file <ApplicationName>.app is created in the given path.
You are prompted whether to save the build information for the boot application.
5. Click on **Yes**.
The build information is saved to the project directory as file <ApplicationName.compileinfo> (e.g. C:\ProgramData\IW-Projects\x\Project\IndraLogic\IndraLogic.IndraMotionMlc1.Application.*.compileinfo). It is a prerequisite for a possible online change upon the next application update.



Select **Build ► Clean** to delete this file.

Transferring PLC sources together with the boot application

Transfer PLC sources (POUs, tasks, etc.) together with the boot application to the control to additionally perform a detailed comparison for PLC elements in the online comparison.

Enable the option **Properties ► PLC** from the context menu of the control.

Also refer to

- [chapter "Command 'Create boot application'" on page 617](#)

12 Testing and debugging

IndraLogic provides various options for testing your application and detecting errors. You can start your application in simulation mode, even without connecting any hardware. Use breakpoints and commands for a step-by-step processing of specific points in the program. Writing values to variables can affect the running program.

Commands are provided that reset your application differently. The different resets are: **Reset warm** (all variables are reset except for RETAIN and PERSISTENT variables), **Reset cold** (all variables are reset except for PERSISTENT variables) and **Reset origin** (all variables are reset).

Also refer to

- [chapter 12.1 "Testing in simulation mode" on page 179](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 12.3 "Stepping a program" on page 181](#)
- [chapter 12.5 "Resetting applications" on page 185](#)

12.1 Testing in simulation mode

Use the simulation mode to test and debug your program if no physical target device is available. In this mode, the application is started on a simulated device.

The command is only available if logged out.

Prerequisite: The program has no errors (compiler error messages or compile errors) and you are not logged in.

1. *Enable the simulation mode:*
 - Select the control in the Project Explorer and then select **Simulation** from the context menu.

In the status line, "Simulation" is displayed. It is highlighted in red. **Simulation** is selected from the context menu.

2. Select **Debug ► Login**.
3. When logging in with the active application for the first time, a request is generated whether to create and load the application Sim.<Device-Name>.<ApplicationName>. Confirm by clicking on **Yes**.

The application is logged in on the control.

4. The program sequence can now be checked and corrected. Respective commands are available in the **Debug** menu.
5. Log out from the control and end the simulation mode.

12.2 Using breakpoints

Breakpoints are commonly used for debugging programs. IndraLogic supports breakpoints in all IEC editors.

Breakpoints can be set at specific positions in the program to force an execution stop and to monitor variable values.

The stop at the breakpoint can be attached to additional conditions.



The **Breakpoints** view provides an overview on all defined breakpoints. It additionally provides commands to simultaneously change multiple breakpoints.

In the editor, the following symbols identify the status of breakpoints:

-  Breakpoint enabled
-  Breakpoint disabled
-  Stop at breakpoint
-  Conditional breakpoint enabled
-  Conditional breakpoint disabled

Also refer to

- [chapter "Command 'Breakpoints'" on page 625](#)

Breakpoints in applications with several tasks

The following applies: Debugging is not possible on several tasks at the same time. When working on a task with breakpoints or step-by-step processing, breakpoints in other tasks are ignored.

If a function block with breakpoint is used by multiple tasks, only the ("debug task") is stopped, as it reaches the breakpoint first. All other tasks continue.

The **Call stack** dialog displays the currently paused task.

If a breakpoint should affect only one specific task, define the breakpoint properties accordingly.

Breakpoints operate separately for each application. Thus, a "Stop at BP" does not affect other applications.



The I/Os operated by the "debug task" are not updated at a stop in the breakpoint, even if the **Update I/Os in stop** option is enabled in the PLC settings.



If the application on the PLC stops at a breakpoint, an online change or a download stops all tasks. This results in a PLC stop!

In this case, IndraLogic displays a corresponding message and prompts whether to continue with the login.

Also refer to

- [chapter "Command 'Call list'" on page 627](#)

Setting a simple breakpoint (example in ST editor)

Prerequisite: The application is in online mode and running. The operating mode is **Debug**.

1. In the editor, open a POU programmed in Structured Text (ST).
2. Place the cursor in the line where a breakpoint is to be set.
3. Select **Debug ► Toggle breakpoint** or press <ALT>+<F9>.

The line is highlighted in red and marked with the icon  (breakpoint enabled). If the program is stopped at the breakpoint, the line is marked with the icon  (stop at breakpoint). Processing the program stops. This is signaled in the status line by the red-highlighted **HALT ON BP** status.

4. Select **Debug ► Start** or press <ALT>+<F5>. The program continues.
5. Set more breakpoints and check the variable values at the breakpoint positions.
6. Place the cursor in the line where a breakpoint should be deleted.
7. Select **Debug ► Toggle breakpoint** or press <ALT>+<F9>. The marking disappears. The breakpoint is deleted.

Also refer to

- [chapter "Command 'Switch breakpoint'" on page 627](#)

Defining a breakpoint condition (example in ST editor)

1. In the editor, open a POU programmed in Structured Text (ST).
2. Place the cursor in the line where a breakpoint is to be set.
3. Select **Debug ► New breakpoint**. The **New breakpoint** dialog opens.
4. Select the **Condition** tab.
5. Select the **Break when the hit count is a multiple of** option in the **Hit count** section.
Enter the value "5" in the field to the right.
6. In addition, define a Boolean condition specifying when the breakpoint should be active. Select the option **Stop if TRUE**. Enter a Boolean variable in the field to the right.
7. Select the option **Enable breakpoint immediately**.
8. Close the dialog.

The line is highlighted in red and marked with the icon .

Monitor the running program. As long as the Boolean variable for the condition is `FALSE`, the breakpoint condition is not fulfilled and the program continues to run. Set the variable to `TRUE` and the condition is fulfilled and the program stops at the breakpoint at every fifth run.

Also refer to

- [chapter 18.3.2 "'Breakpoint properties' dialog" on page 701](#)

12.3 Stepping a program

Use "stepping" to execute an application step-by-step and to navigate through the program code. Stepping is useful to determine the status of the program code at runtime. You can check the call process, track variable values, or locate errors.

Stepping commands are provided for a step-by-step processing in the "Debug" menu. The commands are only available if breakpoints are set during online mode and if the execution stops at a breakpoint.

→ The application is in the "HALT ON BP" (debug mode). During debug mode, the current break position is highlighted in yellow and marked with the  symbol in the text editors.

Switching to "Debug" mode

1. Log into the application in the Project Explorer.
If required, the application code is loaded onto the control.

→ The application is in online mode. The operating state of the control is displayed in the Project Explorer.

2. In the POU's, set breakpoints at the locations in the code that you want to examine.
3. Start the application in the Project Explorer via the context menu item **Start**

→ The application starts and the code is processed until the first breakpoint.

The application is in debug mode.

The application is labeled with "[Halt on BP]" in the Project Explorer.

The status bar provides information about the operating state:

HALT ON BP

The editor is open at the current break position. The line of code with an active breakpoint where program execution was halted is highlighted in yellow and marked by the  symbol. The statement highlighted in yellow has not been executed yet.

Select the various stepping commands or display the call tree.



Alternatively, you can first start the application and then set a breakpoint.

Behavior of the stepping command in the "Debug" menu

The following table lists all available stepping commands and their effects on the program code editing:

step over (step over) Symbol: 	The statement at the breakpoint position is executed. Program execution halts before the next statement in the POU. If the statement contains a call (from a program, function block instance, function, method, or action), then the subordinate POU is processed completely in one step.
step into (step into) Symbol: 	The statement at the breakpoint position is executed. Program execution halts before the next statement. If the statement contains a call (from a program, function block instance, function, method, or action), then the program execution jumps to this subordinate POU. The first statement in the subordinate programming function block is executed and the program execution halts before the next statement. The new current breakpoint position is then in the called POU.
step out (step out) Symbol: 	The command executes the POU from the current breakpoint position to the end of the POU and then jumps back to the calling POU. Program execution stops at the calling position (in the line with the call). If the current breakpoint position is in the main program, then the POU is run through to the end. Then the program execution jumps back to the beginning (to the program start at the first line of code in the programming function block) and halts there.
run to cursor (run to cursor) Symbol: 	First set the cursor at any line of code and then execute the command. The program is executed from the current breakpoint position and halts at the current cursor position without executing the code of this line.

Set next statement (set next statement) Symbol: 	First set the cursor at any line of code (also before the current breakpoint position) and select the stepping command "Set next statement". The statement marked with the cursor is executed next. All statements in between are ignored and skipped.
Show next instruction (show next statement) Symbol: 	If you do not see the current breakpoint position, execute the stepping command "Show next statement" Then the window with the current breakpoint position comes into focus and the breakpoint position is visible.

Tab. 12-1: *IndraLogic - Supported stepping commands*

To show the complete call path of the position currently reached in the program processing, select **Debug ► Call list** from the context menu



The "Call tree" view shows the position of the function block in the call structure of the program at any time, also before compiling the application.

Also refer to

- [chapter "Command 'Single step'" on page 630](#)
- [chapter "Command 'Procedure step'" on page 629](#)
- [chapter "Command 'Execute to return'" on page 630](#)
- [chapter "Command 'Execute to cursor'" on page 630](#)
- [chapter "Command 'Determine next statement'" on page 631](#)
- [chapter "Command 'Show next statement'" on page 631](#)
- [chapter "Command 'Call tree'" on page 592](#)

12.4 Forcing and writing variables

In IndraLogic, variable values on the control can be changed in online mode. It is distinguished between forcing and writing.

When forcing, IndraLogic writes the value in each cycle to hold the variable permanently on the forced value.



Please note:

1. Forced outputs keep their value also in stop state and after an IEC exception
1. Forced variables have to be canceled explicitly by the user

IndraLogic sets the value once while writing. The program can thus overwrite the value at any time.

If forced variables are still present when logging out, a dialog is displayed.

Forcing in the declaration section

Prerequisite: Your application contains a function block with declarations. The application is in online mode.

1. Open the function block in the editor.
2. In the declaration section of the editor, double-click into the column **Prepared value** of a variable from ①.

The field can be edited and a value can now be entered.

Device.Application1.PLC_PRG			
Expression	Type	Value	Prepared value
iColorR	INT	0	100 1
iColorY	INT	0	200
iColorG	INT	0	300

Fig. 12-1:

3. Execute step 2 for further variables.
4. Select **Debug ▶ Force values**.
The variable values are overwritten with the prepared values. The values are marked with the symbol

Device.Application1.PLC_PRG			
Expression	Type	Value	Prepared value
iColorR	INT	100	
iColorY	INT	200	
iColorG	INT	300	

Fig. 12-2:

Forcing in the implementation section

Variable values can also be forced in the **Watch** view.

Prerequisite: The application is in online mode.

If the sequence control is enabled (**Debug ▶ Sequence control**), the values can neither be prepared for forcing nor for writing in the implementation.

1. Open the function block in the editor.
2. In the implementation section of the editor, double-click on an Inline monitoring field from ①.
The **Prepare values** dialog opens.
3. Enter the new value in the field **Prepare a new value for the next write or force operation**.
The prepared value is displayed in the “Inline monitoring” field (1).

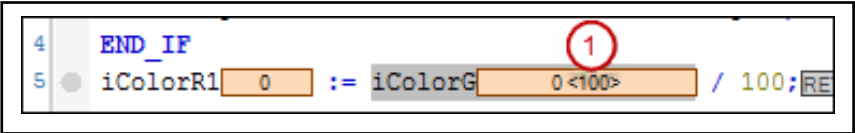


Fig. 12-3:

4. Select **Debug ▶ Force values**.
The variable value is overwritten with the prepared values. The values are marked with the symbol

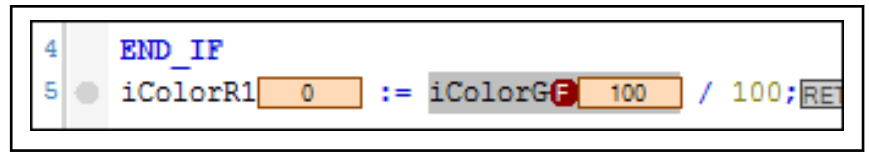


Fig. 12-4:

Seeing and editing all forced variables in one list

Prerequisite: The application is in online mode. Multiple variables are in the forced state.

1. Select **Debug ► Monitoring ► Display all forces**

The **Display all forces** view is shown. It includes a watch list of all currently forced variables of the application.

2. Select all the lines in the list and select **Unforce ► Unforce and retain all selected values** in the selection list on the upper left in the view.

Forcing is canceled for the variables. They obtain the values they had before forcing.

Also refer to

- [chapter 18.3.3 "Prepare value"](#) on page 701
- [chapter "Command 'Force values'"](#) on page 631
- [chapter "Command 'Cancel forcing for all values'"](#) on page 633
- [chapter "Command 'Write values'"](#) on page 632
- [chapter 13.1.2 "Using watch lists"](#) on page 196

12.5 Resetting applications

Resetting the application stops the program and resets the variables to their initialization values. Depending on the type of reset, retain variables and persistent variables are also reset.

- **Reset warm:** All variables are reset, except `RETAIN` and `PERSISTENT` variables.
- **Reset cold:** All variables are reset, except `PERSISTENT` variables.
- **Reset origin:** All variables are reset

The following example program and instructions show the response of the various resets.

Also refer to

- [chapter 17.2.11 "Persistent variables - PERSISTENT"](#) on page 288

Example program

Declaration

```
VAR
    iVar: INT := 13;
END_VAR
VAR RETAIN
    iVarRetain: INT := 14;
END_VAR
VAR PERSISTENT
    iVarPersistent: INT := 15;
END_VAR
```

Implementation

```
iVar := iVar + 1;
```

```
iVarRetain := iVarRetain + 1;
iVarPersistent := iVarPersistent + 1;
```

Executing a "Reset warm", "Reset cold" and "Reset origin"

1. Insert the **PersistentVars** object below the application and open it in the editor.
2. Select **Create ► Build**.
3. Select **Declaration ► Add all instance paths**.
The instance path of the persistent variables is inserted.

4. Load the application to the control.
Prerequisite: The example program runs on the control.

1. Select **Debug ► Login** to switch to online mode.
2. Start the program (**Debug ► Start** or context menu of the application)
3. Monitor the variables `iVar`, `iVarRetain` and `iVarPersistent`.
4. Select **Online ► Reset warm**.
You are prompted whether you really want to execute the command.
5. Select **Yes** to confirm the dialog.
The application is reset. The `iVar` variable is set to the initialization value (in the example 13). The other two variables retain their values.

6. Restart your program.
7. Select **Online ► Reset cold**.
You are prompted whether you really want to execute the command.

8. Select **Yes** to confirm the dialog.
The application is reset. The `iVar` and `iVarRetain` variables are set to the initialization values (in the examples 13 and 14). The `iVarPersistent` variable retains its value.

9. Restart your program.

10. Select **Online ► Reset origin**.
You are prompted whether you really want to execute the command.

11. Select **Yes** to confirm the dialog.
The application is reset (not loaded). The variable values cannot be read anymore. Load the program again (**Debug ► Load**) and start it again (**Debug ► Start**).

Also refer to

- [chapter "Command 'Reset warm'" on page 622](#)
- [chapter "Command 'Reset cold'" on page 623](#)
- [chapter "Command 'Reset origin'" on page 624](#)

12.6 Sequence control

Use the sequence control to monitor the processing of the application program. The sequence control is provided for the language editors ST, FBD, LD and IL.

With an enabled sequence control, IndraLogic displays the variable values and results from function calls and operations at the respective processing position and processing time. In this way, the exact lines of code and networks that run through the current cycle are marked in colors.

For comparison:

In standard monitoring, IndraLogic provides only the value that a variable has between two processing cycles.

The sequence control operates in all currently visible parts of the editor window. **Sequence control enabled** is displayed in the status line as long as the function is active and the sequence control positions (processed parts of code) are visible in an editor window.

Values can only be written and forced in the declaration section. (Double-click on the value display field and the "Prepare value" dialog opens. Then, enter the value).



When enabling the sequence control, the cycle time of the application is prolonged!

When flow control is activated, it is not possible to use breakpoints or step through the program.

Also refer to

- [chapter "Command 'Switch sequence control mode'" on page 633](#)

Display of the sequence control in different language editors:

By default, IndraLogic displays the flow control positions of the processed parts of code as green fields. Unprocessed parts of code are displayed in white.



Please note that the displayed value of an unprocessed code position is an ordinary "monitoring" value. This is the value between two task cycles.

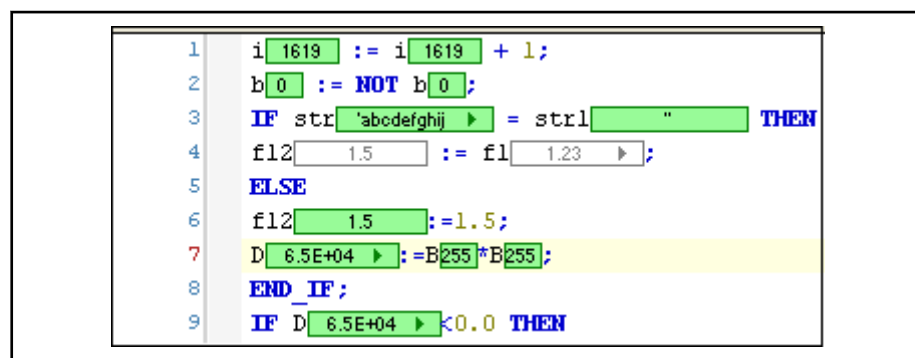


Fig. 12-5:

In network editors, IndraLogic marks the processed networks with bars on the left edge in the "color of the sequence control".

In LD, IndraLogic displays the currently processed connecting lines in green and all other lines in gray. The current value of the connection is shown, as well: TRUE by a bold blue line, FALSE by a bold black line, and unknown or analog values by thin black lines. Due to the combination of the relevant information, this might lead to dashed lines.

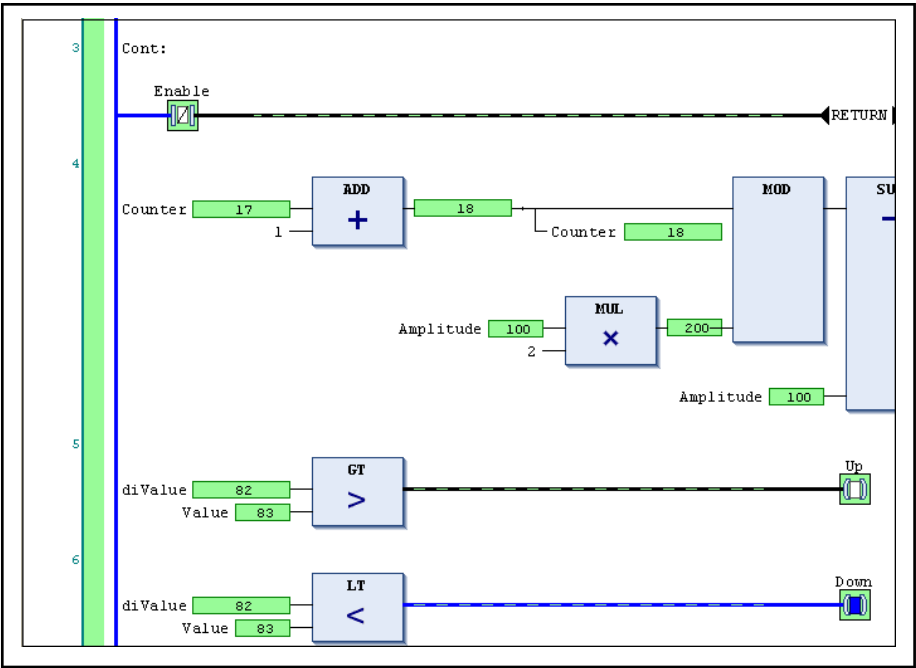


Fig. 12-6:
In IL, IndraLogic uses two fields for each instruction to display the actual values. One to the left of the operator with the current accumulator and one to the right of the operands with the operand value.

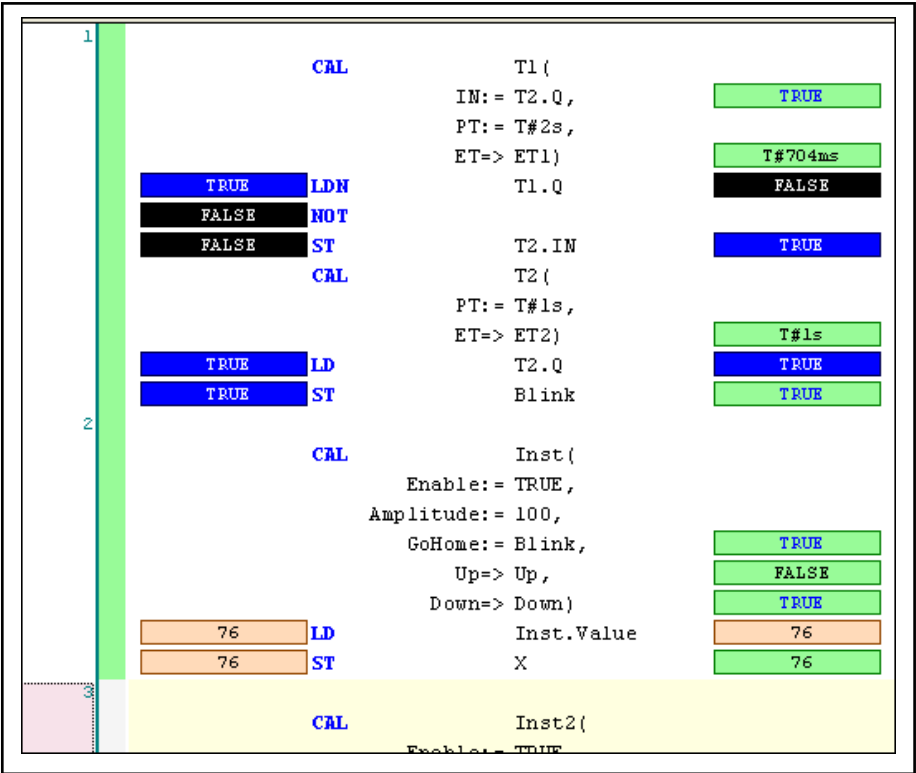


Fig. 12-7:

12.7 Determining the current processing position with the call stack

Use the call stack to determine the current position of the program sequence. This function is very useful when stepping programs.

Prerequisite: The application is in online mode. The program is stopped at a breakpoint or you execute it step-by-step.

1. Open the call stack with **Debug ► Call stack**

The call stack opens. The list shows the current position with the complete call path.

The call stack is also available in offline mode and normal online mode (without using debugging functions). In this case, it contains the last displayed position during a step-by-step execution, but it is displayed in "gray".

Also refer to

- [chapter "Command 'Call list'" on page 627](#)
- [chapter 12.3 "Stepping a program" on page 181](#)

12.8 Checking task list

The tab **Task list** of the device editor provides an overview on the tasks that process the individual inputs and outputs of the I/O mapping of your application and their priority. Check here whether an unintentional overwriting of values is caused. This can result in undefined values.

1. Generate for the application code: For this purpose, select **Build ► Generate code** for example.
2. Double-click on the device object in the Project Explorer to open the device editor. Select the **Task list** tab.

The inputs and outputs of your application and the assignment of the tasks and their priorities are displayed. For details, refer to the description of the **Task list** tab.

Also refer to

- [chapter ""Task list" tab" on page 510](#)
- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- ["General information on I/O mapping \(Mapping\)" on page 74](#)

13 Application at runtime

If the application is running on the PLC, there are IndraLogic functionalities to monitor and change variable values as well as to record and save its characteristic curve.

It is also possible to retrieve certain information from the PLC, to see the PLC logbook, to display a core dump and to see the time response of the task.

Observe to restrict the access on the running application via online commands from the engineering system in case of critical machine states.

13.1 Monitoring values

In runtime mode, the current variable values of a programming object can be monitored at different places in a project. This is called "monitoring":

- Online view of the programming editor of an object: "Inline monitoring"
- Online view of the declaration editor of an object
- Object-independent, configurable watch lists

Set the `{attribute 'monitoring'...}` pragma to monitor the results from function calls and the current variable values in PROPERTY type objects.



More options for recording current variable values:

- Read and save "recipes"
- Record values on a timeline for displaying the history immediately or later: Trace functionalities

Also refer to

- [chapter "Attribute 'monitoring'" on page 394](#)
- [chapter 13.1.2 "Using watch lists" on page 196](#)
- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 13.3 "Data recording with "Trace"" on page 200](#)

13.1.1 Calling monitoring in programming object

The "Monitoring" function allows to present the current variable values of a POU during online mode.

Prerequisites:

- The IndraLogic option "Enable Inline monitoring" is enabled.
For the option setting, go to the IndraWorks menu and select **Tools ► Options ► IndraLogic 2G**, category **Text editor - Monitoring** **Enable Inline monitoring**.
- The application was downloaded to the control and the control is running.
- The application is logged in (online)
- The POU is open in the editor

Setting the representation format and starting a POU monitoring

1. To set the representation format to decimal values, select **Debug ► Display mode ► Decimal**.
2. Download an application to the control and start it.
3. Log into the application (online mode).

4. Double-click into the Project Explorer to open the POU.
→ The editor opens.
Actual values of the variables are displayed and refreshed continuously in the declaration and implementation.

Monitoring in the declaration editor

The current variable value is shown in the **Value** column (3). The value entered into the **Prepared value** (4) column can be written or forced. A red forced symbol **F** precedes the forced values.

PlcProg					
IndraMotionMlc1.Application.PlcProg					
Expression ①	Type ②	Value ③	Prepared value ④	Address ⑤	Comment ⑥
iVar	INT	F 2411		%MW9	
bVar	BOOL	TRUE	FALSE	%QX0.3	
myStruct	TestStruct				Check address defined in "TestStru
nTest1	INT	2411		%MW0	
nTest2	INT	0		%MW2	
aVar	ARRAY [0..3] OF INT			%MW5	
fbinst	FB1				instance of function block FB1
fbin	INT	0			
fbout	INT	0			
fhvar	INT	n			

- ① Expression
- ② Data type
- ③ Value
- ④ Prepared value
- ⑤ Address
- ⑥ Comment

Fig. 13-1:

The actual value of an expression ① is displayed in the “Value” ③ column.
You can write and force a value in the “Prepared value” ④ column. During the forcing, the actual value is decorated with a red symbol (**F**).
The expression of an interface reference can be expanded. If the interface points to a global instance, then this global instance is displayed as the first entry below the reference. Afterwards, if the interface reference changes, then the displayed reference is collapsed.

Monitoring in the implementation
Inline monitoring

"Inline monitoring" is the display of the current variable value in the implementation.

Depending on the implementation language, the following displays are possible in the implementation part:

- Variables have a window with the current value displayed after their name: `nResult2` 17.
If you have prepared values for variables for forcing or writing, then they are displayed in angle brackets in the inline monitoring view after the current value.
After forcing, the respective values are identified by the **F** symbol.
- Network editors and the CFC editor:
Connecting lines (signal lines) are displayed in color according to their actual Boolean value: Blue means `TRUE`, black means `FALSE`.
- LD editor:
The contact and coil elements are also marked.
For contacts and coils, a prepared value (`TRUE` or `FALSE`) is shown in a small view next to the element.

- SFC editor:
Transitions with the value `TRUE` are displayed in color according to their actual Boolean value: Blue means `TRUE`, black means `FALSE`.
Active steps are displayed in blue.
Forced transition values are displayed in red in the implementation.
- IL tabular editor:
Current values are displayed in a separate column.

Example:

Monitoring in the ST editor

```

1 iVar[13945] := iVar[44] + 1; (* counter *)
2 bVar[TRUE] := TRUE;
3 myStruct.nTest1[13945] := iVar[139 < 39];
4 aVar[2][13945] := iVar[13945];
5 erg[0] := fbinst.fbout[0]; RETURN

```

Fig. 13-2:

Example:

Monitoring in the LD editor

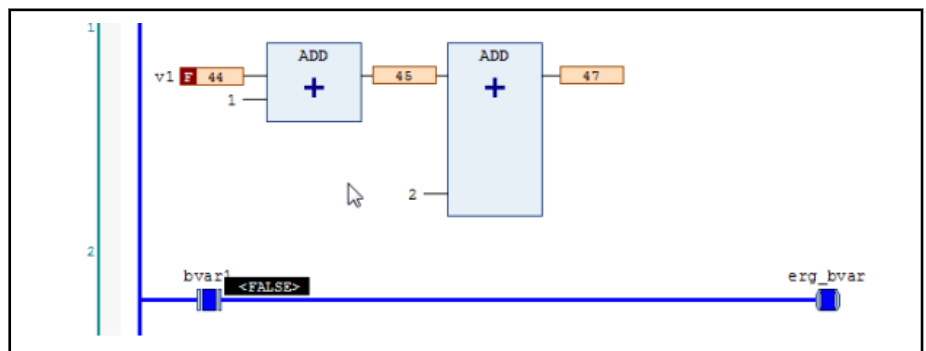


Fig. 13-3:

Example:

Monitoring in the SFC editor

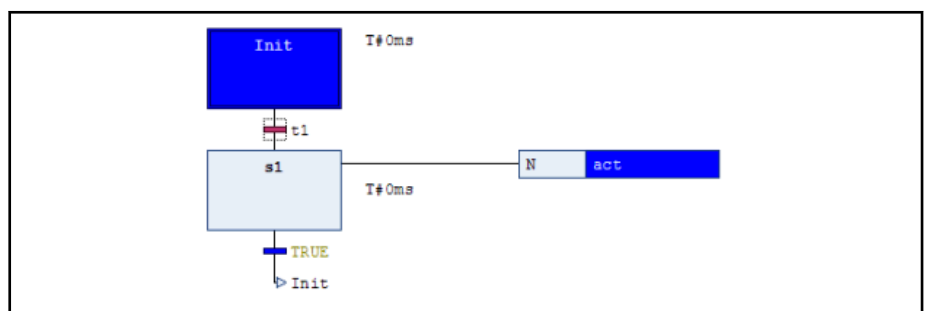


Fig. 13-4:



Disable the "Inline monitoring" function here: **Tools** ► **Options** ► **IndraLogic 2G**, category **Text editor / Monitoring**.

Also refer to

- [chapter "ST editor in online mode" on page 226](#)
- [chapter "FBD/LD/IL editor in online mode" on page 258](#)
- [chapter "SFC editor in online mode" on page 236](#)
- [chapter "CFC editor in online mode" on page 273](#)

Partial monitoring of an array

An expanded array can show the actual values of up to 1,000 elements. This can be very complex for big arrays.

In addition, an array can contain more than 1,000 elements and thus exceed the maximum amount of representable elements. It is useful to limit the range of displayed elements as described in the following section.

Limiting monitoring ranges

Prerequisite: An application is running. It contains a multidimensional array variable with more than 1000 elements.

Example: `arrBig : ARRAY [0..100, -9..10, -19..20] OF INT;`

1. Click in the field of the data type column in the variable `arrBig`.
→ The "Monitoring range" dialog opens.
2. In "Start", enter the value `[1, -9, -19]`.
3. In "Ende", specify the value `[1, 10, 20]`.
→ The actual values of 800 array elements are displayed in the declaration editor. The range is limited to the elements of the index `[1, <i>, <j>]` with `i` from -9 to 10 and `j` from -19 to 20.

Also refer to:

- [chapter 9.2.1 "Using the declaration editor" on page 85](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter 18.3.5 "Dialog "Monitoring range"" on page 704](#)

Monitoring function blocks

When you double-click the editor view of a function block in online mode, a dialog opens where you can choose between viewing the basic implementation or a specific instance.

If you select the basic implementation, then the code is displayed in the editor without current values. Now set a breakpoint in the basic implementation. If the execution halts there, then the current values of the instance that is processed first in the program flow are displayed. Now you can step successively through all instances.

If you select one of the instances, then the editor opens with the code of the function block instance. The current values are displayed in the declaration and, if applicable, in the implementation, and are updated continuously.

Also refer to:

- [chapter ""Function block" object \(FUNCTION_BLOCK\)" on page 523](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)

Monitoring properties

You can monitor variables in a property object  by setting a breakpoint in the function during online mode. When halted there, the current values are displayed.

In addition to your own values, the values of the variables of the superordinate instance are displayed automatically. In the declaration part of the property, the `THIS` pointer, which points to the superordinate instance, appears in the first line with the current data type information and values.

Example

```

FUNCTION_BLOCK FB_BaseAlfa
VAR
    iBaseLocal : INT;
    sBaseLocal : STRING;
END_VAR
iBaseLocal := iBaseLocal + 1;
sBaseLocal := 'Testing Text';

FB_BaseAlfa.PorpBeta.Get
iBaseLocal := iBaseLocal + 1;
IF iBaseLocal > 0 THEN
    PropBeta := TRUE;
END_IF

FB_BaseAlfa.PorpBeta.Set
IF PropBeta = TRUE THEN
    iBaseLocal := 0;
    sBaseLocal := 'Tested IF';
END_IF

PROGRAM PLC_PRG
VAR
    fb_BaseAlfa : FB_BaseAlfa;
END_VAR

fb_BaseAlfa();

IF fb_BaseAlfa.PorpBeta = TRUE THEN
    xResult := TRUE;
END_IF
IF xReset THEN
    fb_BaseAlfa.PorpBeta := TRUE;
    xReset := FALSE;
END_IF

```

Also refer to [chapter "Object 'Property'" on page 537](#)

Monitoring property accesses in the parent programming object

You can monitor the values of subordinate properties  in a function block or program in addition to the variable values.

To do this, add either the pragma {attribute 'monitoring' = 'variable'} or {attribute 'monitoring' = 'call'} to the subordinate property object in the declaration.

If you open the superordinate program instance or function block instance at runtime, then the current property values are displayed in the editor in addition to the current variable values.

Also refer to:

- [chapter "Object 'Property'" on page 537](#)
- [chapter "Attribute 'monitoring'" on page 394](#)

Monitoring methods

You can monitor variables in a method object  by setting a breakpoint in the method during online mode. When halted there, the current values are displayed.

In addition to your own values, the values of the variables of the superordinate instance are displayed automatically. In the declaration part of the method, the `THIS` pointer, which points to the superordinate instance, appears in the first line with the current data type information and values.

Example

```

FUNCTION_BLOCK FB_BaseAlfa
VAR
    iBaseLocal : INT;
    sBaseLocal : STRING;

```

Application at runtime

```

END_VAR
iBaseLocal := iBaseLocal + 1;
sBaseLocal := 'Testing Text';

METHOD MethBaseAlfa : BOOL // Method of FB_BaseAlfa
VAR_INPUT
END_VAR
VAR
    iMethLocal : INT;
END_VAR
iMethLocal := iMethLocal + 1;

PROGRAM PLC_PRG
VAR
    fb_BaseAlfa : FB_BaseAlfa;
END_VAR
fb_BaseAlfa();
fb_BaseAlfa.MethBaseAlfa();

```

Also refer to:

- [chapter "Object 'Property'" on page 537](#)
- [chapter ""Method" object " on page 532](#)

Monitoring functions

You can monitor variables in a function object by setting a breakpoint in the function during online mode. When halted there, the current values are displayed.

Monitoring the return value of a function call

In the ST editor of a POU, the current return value is displayed as inline monitoring at the position of the POU where a function is called.

The following conditions must be fulfilled:

- The return value is a type that can be interpreted as a 4-byte numeric value. Example: INT, SINT, or LINT.
- The pragma {attribute 'monitoring' := 'call'} is inserted into the function.

Also refer to

- [chapter ""Function" object \(FUNCTION\)" on page 525](#)
- [chapter "Attribute 'monitoring'" on page 394](#)

13.1.2 Using watch lists

What is a watch list?

A watch list is a user-defined list of project variables summarized in one view to monitor their values.

In online mode, variable values can be written and forced into a watch list.

Monitoring, writing and forcing are to be handled as in online mode of the declaration editor.

You can customize the format of the representation of floating-point values in the options for monitoring.

In one project, there are four watch lists `Watch list <n>` in the menu under **Debug ► Monitoring**.



If the expression is an interface reference, it can be expanded. In case the interface points to a global instance, the global instance is shown as first entry below the reference. If the interface reference changes, the displayed reference is collapsed.

See:

- [chapter 17.1.1 "Declaration Editor" on page 223](#)
- [chapter 18.3.5 "Dialog "Monitoring range"" on page 704](#)

**Creating and editing a watch list
(offline or online mode)**

Prerequisite: A project is either opened in online or offline mode. The project includes an application with declared variables to be added to one of the four possible watch lists.

1. Select **Debug ► Monitoring ► Watch list <n>**.

The `Watch list <n>` view opens with a blank table row.

2. Double-click on the field in the **Expression** column and enter a variable to be monitored either manually or using the input assistant.

Syntax:

`<DeviceName>.<ApplicationName>.<ObjectName>.<VariableName>`

Example:

`IndraMotionMlc1.Application.PlcProg.ivar`

When entering the name of a structured variable, the individual components are displayed automatically in other lines in online mode.

3. Define all variables to be monitored with this list. Change the order via drag&drop.

The **Execution point**, **Type**, **Address**, **Comment** fields are filled in automatically according to the variables declaration.

The symbol in front of the expression indicates the variable type:

-  Input variable
-  Output variable
-  "normal" variable



In online mode, create and edit watch lists using the context menu item **Add to watch list**.

Also refer to

- [chapter "Command 'Monitoring- 'Monitoring list <n>' on page 628](#)

**Adding variables by selecting the
"Add watch" command (online
mode)**

Prerequisite: A project is open at runtime. It includes an application with declared variables to be added to a watch list.

1. Open the watch list via **Debug ► Monitoring ► Watch list <n>**.
2. Place the cursor on a variable in the declaration or implementation section of a POU and select **Add watch** from the context menu.

An entry for the selected variable is added to the list.

3. Other variables can be added like this or by entering them directly into the list in the **Expression** field as described above ("Creating and editing...").

The watch lists are updated immediately.



If a watch list is not open when clicking on **Add watch list** for a variable, it is added automatically to **Watch list 1**.



Variable values can also be written and forced in watch lists. In online mode, the column **Prepared value** is displayed.

Also refer to

- [chapter "'Add to watch list" command" on page 699](#)

- [chapter 13.1.1 "Calling monitoring in programming object" on page 191](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

13.2 Changing values with recipes

Use recipes to change or read recipes values for a specific set of variables (recipe definition) on the control at the same time.

Define the basic settings for recipes, such as storage location and format, in the **Recipe manager** object. Insert one or more recipe definitions below this object. A recipe definition comprises one or more recipes for the contained variable. The recipe consists of specific variable values.

Recipes can be saved to a file or directly written from files to the control.

Recipes can be loaded via the IndraLogic programming interface, the visualization element or the application program.



Recipe use on remote devices

The variable values of recipes are automatically transmitted to or from another control if they are data source variables and a data source exchange is configured.

Reading/writing is done synchronously. IndraLogic updates all variables contained in a recipe at the same point.

After reading or writing, use the call (`g_RecipeManager.LastError = 0`) to check whether the transfer was successful.

Also refer to

- [chapter 18.1.21 ""Recipe manager" object" on page 549](#)
- [chapter 18.1.22 ""Recipe definition" object" on page 552](#)

Handling recipes in the IndraLogic user interface

The IndraLogic development interface provides commands for generating recipes as well as reading/writing in online mode.

Also refer to

- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

Using recipes in applications

At runtime, recipes can be used in the user program and visualization elements.

The methods of the function block `RecipeManCommands` are used in the user program from the library `RecipeManagement` (**Add library ► Internal**). In the visualization, recipes are used via the input configuration (internal command) of visualization elements.



During the initialization process, the recipe management reads the values of the variables that are defined in the recipe definition. The operation takes place at the end of the initialization phase of the application. At this point, all initial values of the application variables are set. This procedure is used to correctly initialize missing values from recipe files.

Also refer to

- [chapter 19.2 ""Recipe management" library" on page 743](#)
- [chapter "Input configuration" on page 800](#)

Creating recipes

1. Select the **Application** object in the Project Explorer.
2. Select **Add ► Recipe manager**.

IndraLogic adds the recipe manager to the Project Explorer.

3. Select the **Recipe manager** object in the Project Explorer.
4. Select **Add ► Recipe definition**.

IndraLogic adds the recipe definition below the recipe manager.

5. To open the editor of the recipe definition, double-click on the object.
6. Double-click on the blank field below **Variable** in the editor. Specify the name of a variable that you will define a recipe. The input assistant can be used: Button .
7. Select **Recipes ► Add a new recipe** and enter a name for the new recipe.
A column with the recipe name is displayed in the editor.
8. Enter the variable value for this recipe in this field.
9. Insert additional fields if required.
10. Select a variable value of the recipe and click on **Recipes ► Save recipe**.
Select storage location and file name.

IndraLogic saves the recipe in the format defined in the recipe manager.

Also refer to

- [chapter 18.1.21 ""Recipe manager" object" on page 549](#)
- [chapter 18.1.22 ""Recipe definition" object" on page 552](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

Loading a recipe from a file

Prerequisite: A recipe manager is available in the application. In a recipe definition, there is a **myRec** recipe with variable values. A `myRec.txt` recipe file is located on the file system and contains the entries for this recipe.

Example of the recipe file:

```
PlcProg.bVar:=0
PlcProg.iVar:=2
PlcProg.dwVar:=35232
PlcProg.stVar:='first'
PlcProg.wstVar:="123443245"
```

1. Double-click on the **Recipe definition** object in the Project Explorer to open the tabular editor for the definition of the individual recipes.
There is a `myRec` column with the current values for this recipe.
2. Edit the `myRec.txt` file in an external text editor and replace the variable values with other values to be loaded to the recipe definition in IndraLogic. Save the file.
3. Click on the **myRec** column in the recipe definition and select **Load recipe** from the context menu.
A note is displayed that an online change is required upon the next login. An online change is required when the current values of the recipe variables are changed by loading the recipe.
4. Click on **Yes** to close the dialog (continue). Select the `myRec.txt` file from the file explorer for loading.
The recipe values in the recipe definition are updated according to the values read in the file.



If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definition are not loaded during the update and remain unchanged on the control and in the project.

For values of the data type `REAL/LREAL`, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

Also refer to

- [chapter ""Load Recipe" command" on page 681](#)
- [chapter ""Load and write recipe" command" on page 682](#)

Recipe management on the control Memory consumption

If the option **Recipe management in the PLC** is disabled, the recipe manager and recipe definitions do not use any memory on the control anymore.

If this option is enabled, code is generated for the recipe manager and all recipe definitions and this code is stored on the control. The amount of memory consumed primarily depends on the number of recipes and their variables as well as on the data type of the variables. Whether the fields of the recipe definition are filled is important as well. The memory consumption of recipes cannot be calculated. It has to be determined empirically when required. The following table merely provides some guiding principles.

	Code size (bytes)	Data size (bytes)	Total (bytes)
Recipe definition with 100 INT variables	194406	79400	267352
Recipe definition with 200 INT variables	238318	121284	459344
Recipe definition with 300 INT variables	282230	163084	543856
Recipe definition with 100 BOOL variables	192742	69884	343168
Recipe definition with 200 BOOL variables	235446	101568	436872
Recipe definition with 300 BOOL variables	278146	133284	510072
Recipe definition with 100 string variables	203278	870084	1154000
Recipe definition with 200 string variables	255570	1709784	2973296
Recipe definition with 300 string variables	307886	2549484	2964112

Tab. 13-1: *Memory consumption for IndraLogic 3.5.6 with the control in simulation mode*

13.3 Data recording with "Trace"

A **Trace** allows to record the value history of variables on the control similar to a digital sampling oscilloscope.

Use a trace to follow the value curve of variables on the control similarly to a digital sampling oscilloscope.

When the application is running, all statements are executed first within the task cycle. Then, data recording starts and the values are stored with time stamp.

These time stamps are relative and refer to the start time of the data recording.

The data result in a time-discrete signal and IndraLogic displays its characteristics in the trace editor.

A "sample" (data record) is composed of the value and the time stamp. The runtime system writes the samples to a buffer of definable size.

IndraLogic requests data, saves it in the trace editor buffer and displays it in the trace diagram as a function over time. The value history of the configured variables can be continuously monitored, as IndraLogic displays the latest data.

You can trigger the data recording. When the trigger is initiated, the application saves the data from the time of the trigger and IndraLogic displays the data at the time of the trigger.

Configuring a trace

The configuration and the display of a trace are possible in the project using the trace editor.

Add the "Trace" object to the application object in the Project Explorer.

The trace object always contains a purely application-specific trace configuration. The trace configuration is downloaded to the control and is running parallelly to the application.



A continuous data logging with "Trace" can significantly increase the cycle time of IEC tasks.



A data logging with "Trace" continuous running after logging out of the device.

Runtime system component "CmpTraceMgr"

The device description for a runtime system with trace manager includes the `tracemanager` entry in the `TargetSettings` section.

Example

Device description

```
<ts:section xmlns:ts="http://www.3s-software.com/schemas/TargetSettings-0.1.xsd" name="trace">
  <ts:setting name="tracemanager" type="boolean" access="visible">
    <!-- With this option, the new tracemanager is activated in IL 2G.
         With this option, the CmpTraceMgr component is needed in the runtime system! -->
    <ts:value>1</ts:value>
  </ts:setting>
</ts:section>
```

Then, IndraLogic transfers only the trace configuration when loading the application to the control. When the trace is started, the application interprets the configuration, executes the data recording and buffers the data sets on the control.

The `CmpTraceMgr` runtime system component provides an extended functionality in contrast to a data recording with IEC code.

Thus, the following values can be recorded:

- Parameters of the control (e.g., the temperature curve of a CPU or battery)

- Device signals, e.g. the current characteristics of a drive
- System variables of another runtime system component

Parameters, such as IEC variables, can be configured in the **Trace configuration** dialog under **Variable settings**.

Also refer to

- [chapter 13.3.4 "Accessing all traces of the control" on page 207](#)
- ["Subdialog "Variable settings" on page 740](#)

Data recording after triggering

To monitor data that depends on an event or a condition, start the data recording with regard to a trigger.

At runtime, the application checks whether the event has occurred or the condition is fulfilled and then it buffers the data accordingly.

The trace configuration allows to initiate a trigger via:

- a trigger variable that maps the event
- a condition as expression
- a combination of trigger variable and condition

Saving samples to file

The samples on the file system of the development system can be saved as file.

The file can include the trace configuration additionally.

File extension	File type	Description
*.trace:	Trace file	Contains the samples and the trace configuration in XML format. Execute the Load trace command to load the file offline to the trace editor and analyze the samples without operating a control.
*.txt	Text file	Contains the samples in ASCII format. The file can be processed further using an external tool.
*.trace.csv	Trace dump file	File in CSV format with the trace configuration and optional samples. Create the file via Export symbolic trace config . This file can be transferred to the control and loaded to the application. Then, execute the Load trace command in IndraLogic to display them in the trace editor. Select Trace ► Save Trace and then the *.trace.csv file format alternatively. The file can be transferred to a control and loaded and analyzed with an HMI.
.traceconfig	Symbolic trace configuration	Contains the trace configuration in CSV format. Create the file via Export symbolic trace config . The CmpTraceMgr runtime system component can read the file.

Tab. 13-2: Possible file formats

Also refer to

- [chapter 18.1.27 ""Trace" object" on page 568](#)
- [chapter ""Save trace" command" on page 695](#)
- [chapter ""Load trace" command " on page 692](#)
- [chapter ""Export symbolic trace config" command" on page 691](#)
- [chapter 18.1.24 ""Symbol configuration" object" on page 553](#)

13.3.1 Getting started with trace

PlcProg

```
PROGRAM PlcProg
VAR
    rSin : REAL;
    rVar : REAL;
END_VAR
rVar := rVar + 0.1;
rSin := 30 * SIN(rVar);
```

Prerequisite: The application is running with the `PlcProg` program on the control.

1. In the Project Explorer, select the application and add a new trace object via **Add**.

The respective trace editor opens and the commands from the **Trace** menu are available.

2. Select **Trace ► Configuration**.

The **Trace configuration** dialog opens.

3. Select a task to run the trace functionality. This is normally the same task that is running in PlcProg (`PlcTask`).
4. Click on **Add variable** to add an entry to the tree view of the trace configuration and assign an IEC variable (e.g., `PlcProg.rSin`).
5. Log in via the context menu item of the application **Login** and start the application (context menu item of the application **Start**).
6. Select **Trace ► Download trace**.

IndraLogic loads the trace configuration to the control. The application starts recording data and transmits that data to IndraLogic. They are displayed in the trace diagram as graph. Commands are provided for navigating through the samples and controlling the data recording.

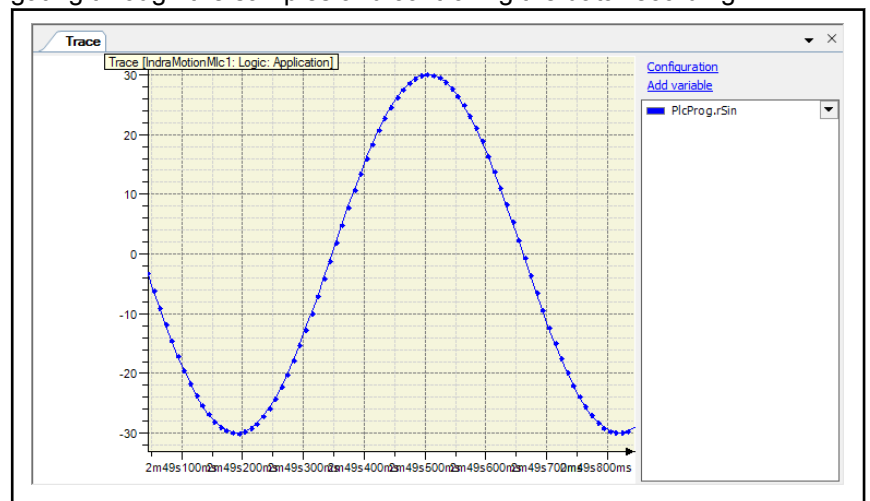


Fig. 13-5: Record the sinusoidal data of the IEC variable "PlcProg.rSin"

Also refer to

- [chapter 18.1.27 ""Trace" object" on page 568](#)

13.3.2 Creating the trace configuration

Specify at least one task and one variable for a complete trace configuration.

To trigger data recording, enable the trigger option and select a trigger variable or specify a recording condition.

Refer to

- [chapter "'Advanced trace settings" dialog" on page 736](#)
- [chapter "'Trace configuration' dialog" on page 737](#)

Assigning task

Data recording is executed during runtime in this task. Usually, the same task is selected in which it is written on the variable.

1. Double-click on the trace object.
The trace editor opens and the commands of the **Trace** menu are available.
2. Select **Trace ► Configuration**.
The **Trace configuration** dialog opens. In the tree view of the **trace configuration**, the top entry is selected, and to the right, the **Record settings** dialog is available.
3. Click in the **Task** input field to .
The selection list opens with all tasks available application-wide.
4. Select a task for the trace.

Also refer to

- [chapter "'Trace configuration' dialog" on page 737](#)

Configuring a trace variable

1. Double-click on the trace object.
The trace editor opens. The commands of the **Trace** menu are available.
2. Select **Add variable**.
The **Trace configuration** dialog opens. The subdialog **Variable settings** is displayed on the right.
3. Click on  in the input field of the **Variable** setting and select a trace variable in the "Input wizard" dialog.
The variable is configured for data recording. The trace record tree and the display tree were extended by the variable.
4. Select the **Add variable** link.
The trace record tree and the display tree receive a new variable. The settings of the variables are available on the right.
5. Select a trace variable.
6. Exit the dialog with **OK**.
The variables are trace variables and are displayed in the trace variable list.

Deleting a trace variable

1. Double-click on the trace object.

2. Click on a variable in the trace record tree.
The variable is selected.
3. Click on the **Delete variable** link or press <Del>.
4. Close the dialog with **OK**.
The variable is removed from the trace variable list.

Tracing a parameter

1. Double-click on the trace object.
2. Select **Add variable**.
The **Trace configuration** dialog opens. The subdialog **Variable settings** is displayed on the right.
3. Click on  to the right of the **Variable** setting.
4. Select the **Parameter** option in the drop-down list.
5. Click on  and select a parameter from the **Input assistant** dialog.
6. Configure how the parameter is to be displayed.
7. Close the dialog with **OK**.
The parameter is now displayed in the trace variable list.

Configuring a trigger

1. Double-click on the trace object.
The trace editor opens with the commands available in the **Trace** menu.
2. Select **Trace ► Configuration**.
The dialog **Record settings** is displayed. To the right, the subdialog **Record settings** is available.
3. Select the option **Enable trigger**.
4. Select the task in which the trend record is to be executed.
5. Select a variable from the **Trigger variable** field.
6. Close the dialog with **OK**.
The data recording will be triggered.

The trigger time is displayed as a black line in the diagram in runtime mode.

1. Download the application and start it.
2. Click **Trace ► Download trace**.
Loading the trace configuration. After triggering, the runtime system saves the value graph of the trace variables. The data is displayed in the trace editor. The trigger time is displayed as a black line in the diagram.

Configuring the display of the time axis

1. Double-click on the trace object.
2. Click on the **Configuration** link above the configuration tree.
The **Trace configuration** dialog opens.
3. Select **Time axis** in the display tree under **Display ► Diagrams**.
The display settings of the time axis are shown on the right.

4. Edit the default settings and click on the **Preview** link.
The changes are visualized in the coordinate system preview.
5. Select **Y-axis** in the display tree. The **Y-axis** item is contained under every configured diagram. Therefore, the display of the value axis is set for each diagram separately.
The subdialog **Display settings** of the selected axis is displayed on the right.
6. Change the setting
The changes are applied to the coordinate system preview.
7. Click **OK** to close the **Trace configuration** dialog.
The display changes are visible in the affected diagrams.

Configuring the display of the trace variable

1. Double-click on the trace object.
2. Click on the **Configuration** link.
The **Trace configuration** dialog opens.
3. Select a variable under **Trace record**.
The subdialog **Variable settings** of the selected variable is displayed on the right.
4. Change a setting, for example the **Line type**.
5. Close the dialog with **OK**.
The display changes are visible in the affected diagrams.

Configuring the buffer for data on the runtime system

1. Double-click on a trace object.
2. Select **Trace ► Configuration**.
The **Trace configuration** dialog opens. The subdialog **Record settings** is displayed on the right.
3. Click on **Advanced**.
The **Advanced trace settings** dialog opens.
4. Change the default settings in **Measure in each nth cycle** or **Size of runtime buffer size in samples**.
5. Close the dialog with **OK**.
The buffer settings are reconfigured. It is applied after the trace configuration is loaded to the RTS the next time.

Editing the trace configuration at runtime

Prerequisite: The application is running on the control and a trace configuration is loaded.

1. Double-click on a variable in the trace record tree.
The **Trace configuration** dialog opens.
2. Change the color.
The variable is displayed in the new color in the affected diagrams without interrupting the execution of the application.



When changing essential settings, for example a trace variable, download the trace configuration to the control again.

Also refer to

- [chapter ""Advanced trace settings" dialog" on page 736](#)
- [chapter ""Trace configuration" dialog" on page 737](#)

13.3.3 Controlling data recording

Use menu commands to control the data recording.

Prerequisite: The application is loaded on the runtime system and a trace is configured.

Menu commands for "Trace"

- [chapter ""Download trace" command" on page 690](#)
- [chapter ""Start trace" command" on page 696](#)
- [chapter ""Stop trace" command" on page 696](#)
- [chapter ""Reset trigger" command" on page 695](#)

13.3.4 Accessing all traces of the control

Use menu commands to access the `CmpTraceManager` runtime system component.

Prerequisite: The application is in online mode.

Use the following trace commands to access the traces on the control `CmpTraceMgr`:

- [chapter "Command 'Online list'" on page 694](#)
- [chapter "Command 'Upload trace'" on page 697](#)

13.3.5 Navigating in the trace data

Use menu commands to navigate in the data of the trace diagram.

Prerequisite: The application is in online mode.

Menu commands "Trace"

- [chapter ""Cursor" command" on page 689](#)
- [chapter ""Zoom with mouse" command" on page 692](#)
- [chapter ""Reset to default view" command" on page 695](#)
- [chapter ""AutoFit" command" on page 689](#)
- [chapter ""Compress" command" on page 689](#)
- [chapter ""Stretch" command" on page 696](#)
- [chapter "Command 'Convert to multi-channel'" on page 692](#)
- [chapter "Command 'Convert to single-channel'" on page 693](#)

13.3.6 Managing trace

Use menu commands to load and save traces in various formats.

Menu commands "Trace"

- [chapter ""Save trace" command" on page 695](#)
- [chapter ""Load trace" command" on page 692](#)
- [chapter ""Export symbolic trace config" command" on page 691](#)

13.3.7 Showing statistics

Function: IndraLogic evaluates and the recorded data statistically, displays it and allows to store it into the clipboard.

Call: Install the command via **Tools ▶ Customize - Trace ▶ Statistics**

Also refer to

- [chapter ""Statistics" command" on page 697](#)

13.4 Monitoring tasks

In online mode, some statistical values of the tasks in the runtime system can be displayed. This information is very useful when testing cycle times or solving problems with the time response.

1. Switch to online mode.
2. Select the **Task configuration** in the Project Explorer and double-click to open it.

The task configuration opens in the editor.

3. Select the **Monitoring** tab.

Also refer to

- [chapter ""Monitoring" tab" on page 564](#)

13.5 Reading PLC logbook

IndraLogic allows to display the results and error messages logged in the control.

Reading logbook Prerequisite: The control is running.

1. Select the control in the Project Explorer.
2. Select **Open** from the context menu.

The device editor opens.

3. Select the **Log** tab.

4. Click on  to refresh the view.

A connection is established to the control.

All log information of the control is displayed.

5. Click on  to delete the current list.

6. To filter the view, click on the respective category (e.g. "Information").

7. Save the log entries. Click on  and select a file name.

Also refer to

- [chapter ""Log" tab" on page 495](#)

13.6 Error analysis using core dump

A core dump is a dump of application data.

If there is an exception on the control, the control automatically stores a core dump.

The core dump is saved in a file <ApplicationName>.core in the application directory.

In online mode, a core dump can also be generated explicitly if the application is currently stopped at a breakpoint or if an exception occurred.

In this case, IndraLogic stores the core dump file in the project directory and not on the control (e.g. C:\ProgramData\IW-Projects*\Project\IndraLogic\IndraLogic.IndraMotionMlc1.Application.*.core).

In offline mode, the core dump can be loaded from the control to the project. An online view of the application is displayed with data and values at the time of the exception.

In addition, a project view of the control log file can be called at the time of the core dump generation.



The runtime system has to support this functionality completely to ensure a correct display of variable values in the core dump view.

In the online view of the application, which IndraLogic created while loading the core dump to the project, menu commands are displayed as available, which are not applied in this status. When selecting one of these commands, a message is displayed.

CAUTION

Restriction (CDS-37572):

Exception handling in the runtime system may overwrite parts of the variable values in the call stack. Therefore, the original values are lost and important information may no longer be displayed for variables in functions and methods while monitoring.



To create a project archive: If a core dump explicitly created with **Create core dump**, this option is available automatically in the archive configuration.

If the core dump was copied from the PLC, it can only be added to a project archive via **Additional files**.

Warning:

A project archive with a core dump has to include the download information. Otherwise, IndraLogic cannot use the core dump.

Loading core dump and device log into the project for analysis

Prerequisite: A project is open with the application that caused an exception on the control. The runtime system can create core dumps. The application in the project is in offline mode.

1. To load the core dump from the PLC, select **Debug ► Core dump ► Load core dump**.

IndraLogic shows an online view of the application. Variable values and call stack from the time of error can be seen (see below). This affects the values of the I/O variables in the mapping dialog of the device configurator and the task configuration as well. The status bar displays "Core dump loaded".

IndraLogic copies the core dump file with the name <Project-Name>.<DeviceName>.<ApplicationName>.<Application-Guid>.core to the local project directory (e.g. \...\IndraLogic.IndraMotionMlc1.Application.*.core).

2. Click on **Debug ► Core dump ► Load device log from core dump** to load the device log at the time of error into the project.
The log view opens (as in online mode of the device editor) with the events at the time of the core dump.
3. After the core dump analysis, click on **Debug ► Core dump ► Close core dump**.
IndraLogic closes the core dump view of the application. The project returns to the views of the normal offline mode.

Creating core dump of the running application manually

Prerequisite: In IndraLogic, one application is in online mode. The runtime system supports the core dump functionality.

1. Select **Debug ► Core dump ► Create core dump**.

IndraLogic starts generating a new core dump. A progress bar is displayed in the status line and a button to cancel the process.

IndraLogic creates a core dump file `<ProjectName>.<DeviceName>.<ApplicationName>.<ApplicationGuid>.core` in the local project directory (e.g. `\\...\IndraLogic.IndraMotionMlc1.Application.*.core`).

Also refer to

- [chapter "Command 'Create core dump'" on page 634](#)
- [chapter "Command 'Load core dump'" on page 635](#)
- [chapter "Command 'Close core dump'" on page 636](#)
- [chapter "Command 'Load device logbook from core dump'" on page 635](#)

13.7 PLC shell to query information

The "PLC shell" in IndraLogic is a text-based control monitor on a tab of the device editor.

Commands to query certain information can be entered and actions, such as starting, stopping or loading applications, can be executed. A description on the syntax of the possible commands can be directly retrieved from the PLC shell.

The possible commands depend on the manufacturer. The number of commands can be extended via the so-called `PLC shell command handler`. It can either be extended in a C-component in the runtime system or as function block in an IEC library or a project.

The response from the PLC is displayed on the tab as character string in a result window.

Querying information on the application on the control

Prerequisite: Your project is connected to the control.

Example of a control on which an application `App1` is running.

1. Double-click on the object `<Control>` in the Project Explorer to open the device editor and go to the **PLC shell** tab.

The tab is displayed with an empty display window. There is an input line for a command below

2. Click on .

The **Enter standard command** dialog is displayed with a list of commands.

3. Select **?** and click on **Execute**.

The dialog closes and a list of the available commands with possible parameters is displayed. It shows the respective syntax of the command input.

4. Click again on  and select **pid**.

Supplement the input line as follows: `pid Appl`. Press enter.

The following is displayed in the output window (GUIDs only as examples):

```
pid Appl
Project identification
Application: Appl
Code GUID:0x08a893c0
Data GUID:0x762d0e90
```

5. Click on  in the command line.

The `pid Appl` command is included into the history of the entered commands.

Also refer to

- [chapter "PLC shell" tab" on page 502](#)

13.8 PLC operation control via system variables

CAUTION

PLC operation control via system variables

You are responsible for runtime system services being enabled under safe application conditions and disabled only under critical conditions.

At runtime, the state of an application or system can become sensitive and disruptive actions can endanger the entire machine or system.

However, in this state, certain commands can be suppressed and thus dangerous actions prevented.

The **PlcOperationControl** function block and the **Component manager** library are provided for this purpose.

Examples of IndraLogic commands whose execution can be suppressed:

- **Online change, Download**
- **Enable breakpoint**
- **Reset application, Stop application**
- **Transfer data**
- **Force values, Write values**

To ensure that there is always a backup solution, **Reset origin** and **Delete** commands cannot be suppressed.

IndraLogic provides a message if a currently disabled runtime system service is required when the application is running. Then, respond with an appropriate countermeasure.

Function block "PlcOperationControl" for operation control

This function block is used to enable and disable operations.

Name	Data type	Initial value	Description
xDisableAll	BOOL	FALSE	TRUE: All operations are suppressed
xDisableApplicationBreakpoint	BOOL	FALSE	TRUE: Setting a breakpoint is suppressed
xDisableApplicationDownload	BOOL	FALSE	TRUE: Download is suppressed
xDisableApplicationForce	BOOL	FALSE	TRUE: Forcing variables is suppressed
xDisableApplicationOnline-Change	BOOL	FALSE	TRUE: Online change is suppressed
xDisableApplicationReset	BOOL	FALSE	TRUE: Application reset (not "Reset origin!") is suppressed
xDisableApplicationStop	BOOL	FALSE	TRUE: Application stop is suppressed
xDisableApplicationWrite	BOOL	FALSE	TRUE: Writing variables is suppressed Can also be prevented by PLCHandler/IecVarAccess.
xDisableFileTransferDownload	BOOL	FALSE	TRUE: File transfer (download) is suppressed
xDisableFileTransferUpload	BOOL	FALSE	TRUE: File transfer (upload) is suppressed

Tab. 13-3: Property (PROPERTY)

Implementing operation control**Prerequisite**

- Compiler version from 3.4.3.0
- In the device description, the PLC operation control is enabled via system variables.

Example**Declare an instance of the function block**

```
VAR
  PlcOpCtrl_Inst : PlcOperationControl;
END_VAR
```

Suppress a command by assigning the respective TRUE property.

Example "Stop application"

```
PlcOpCtrl_Inst.xDisableApplicationStop := TRUE;
```

Also refer to

- [chapter "Command 'Online change'" on page 619](#)
- [chapter "Command 'Force values'" on page 631](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

14 Updating an application on the PLC

IndraLogic provides two options to transfer a modified application to the control: "Download" and "Online change".

A **download** compiles the application again. The syntax is checked and the application code is created and loaded to the control. This stops the running program. A download is the recommended data transfer method, as a defined initial state is always created due to the program stop and the re-initialization.

In the case of an **online change**, only the modified parts are reloaded to the control. The running program is not stopped. Use an online change only for minor application changes. In the case of extensive changes, the program response cannot be safely predicted. Please read the notes in the description of the **Online change** command.

Also refer to

- [chapter 14.1 "Executing an online change" on page 213](#)
- [chapter "Command 'Online change'" on page 619](#)
- [chapter 14.2 "Executing a download" on page 214](#)
- [chapter "Command 'Load'" on page 618](#)

14.1 Executing an online change

IndraLogic automatically provides an "online" change when logging in with an application that is already present on the control, but has been changed since the last download in the programming system.

In this case, only the modified parts are reloaded to the control. A running program on the control is not stopped during the online change.



For the online change, observe the notes in the description of the **Online change** command.

Also refer to

- [chapter "Command 'Online change'" on page 619](#)
- [chapter "Menu 'Login'" on page 614](#)
- [chapter 11.2 "Generating application code" on page 175](#)

Online change when logging in

Prerequisite: The connection settings of the control are set correctly. The applications in the project and on the control are identical. The project on the control is running. The application is logged out.

1. Change your application.
2. Select **Debug ► Login**

A dialog is displayed showing that the application has been changed since the last download.

3. Click on **Details...**
4. Check the details in the **Application information** tab.

If no code was generated since the last change, the command **Application is not up to date. Generate code now?** is displayed at the bottom edge of the dialog. In this case, click on this command.

A comparison view of the objects is shown. Objects highlighted in red are different.

5. Close the dialog.
6. Select the option **Login with online change** and click on **OK**.

The change is loaded to the control. In this case, the running program on the control is not stopped. The application is logged in.

Online change in the logged-in state (online mode)

Prerequisite: The connection settings of the control are set correctly. The applications in the project and on the control are identical. The project on the control is running. The application is logged in.

1. Select an object in the Project Explorer. It is recommended to select a POU or a GVL.
2. Select **Edit object (offline)** from the context menu.
The object opens in the editor.

3. Change the object. For example, declare a new variable or change a value assignment here.

4. Select **Online ► Online change**.

A query is displayed whether you really want to execute the online change.

5. Select **Yes** to confirm the dialog.
The change is loaded to the control.

14.2 Executing a download

The active application is compiled due to an application download. The syntax is checked and the application code is created and loaded to the control. A program running on the control is stopped during download.



Observe the notes in the description of the command **Debug ► Load** while downloading.

Also refer to

- [chapter "Command 'Load'" on page 618](#)
- [chapter "Menu 'Login'" on page 614](#)
- [chapter 11.2 "Generating application code" on page 175](#)

Downloading when logging in

Prerequisite: The connection settings of the control are set correctly. The applications in the project and on the control are identical. The project on the control is running. The application is logged out.

1. Change your application.
2. Select **Debug ► Login**

A dialog is displayed showing that the application has been changed since the last download.

3. Select the option **Login with download** and click on **OK**.

The running program on the control is stopped and the change is loaded to the control. The application is logged in.

Downloading when logged in (online mode)

Prerequisite: The connection settings of the control are set correctly. The applications in the project and on the control are identical. The project on the control is running. The application is logged in.

1. Select an object in the Project Explorer. It is recommended to select a POU or a GVL.
2. Select **Edit object (offline)** from the context menu
The object opens in the editor.
3. Change the object. For example, declare a new variable or change a value assignment here.
4. Select **Debug ► Load**.
A query is displayed whether you really want to download.
5. Select **Yes** to confirm the dialog.
The running program on the control is stopped and the change is loaded to the control.

15 Copying files to/from PLCs

With IndraWorks/IndraLogic, files can be copied between the local file system and the control.

Prerequisite: The connection to the control device is configured in the Project Explorer of your project. The device is running.

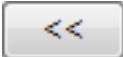
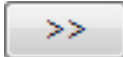
1. Open the context menu **<Control> ► Device data ► Explorer** in the Project Explorer.
2. Go to the left section of the dialog (**PC**) and set the path in the local file system (where files are copied to and from) at **Address**. Example: D:\FileTransferWithPLC. If necessary, create a new directory via .

The files and directories are shown like in a file manager. Press  to update the display.

3. The required directory for the file transfer is analogously set in the right section of the dialog **Control**.

IndraLogic shows the files present on the control.

4. Select the required files from the file system tree for the file transfer. Several items can be selected at once. To transfer all files in the directory, select the directory.

5. Click on  or  between the dialog sections.

IndraLogic copies the selected files immediately to the other file system. If a file is not yet available in the target directory, it is created. If it is already available and not write-protected, it is overwritten. A corresponding message is then displayed.

Examples: *Also refer to:*

- RIL_ProfinetIO.library/ IL_PNIODDeviceBackup
- RIL_ProfinetIO.library/ IL_PNIOIoLinkBackupDevice
- RIL_EthernetIP.library/ IL_ENIPAdapterBackup

16 Creating and managing libraries

Library repository Libraries and their respective metadata are stored in the library repository on the engineering system. Implement any installed the libraries into your project using the library manager. The libraries are installed under version control to easily update the libraries.

You can create and edit more repositories in addition to the preinstalled System repository.

Refer to [chapter ""Library repository" command" on page 636](#)

Library manager The library manager manages the libraries implemented into your project. It displays the libraries as per the respective library types and properties. Use the library manager to implement or remove further libraries or to edit their properties.

The library manager can be located either in the project folder "General module folder" or in the "Application" node of the control. Thus, a project can have one library manager per application and also one cross-application library manager in the "General module folder". The library function blocks of the linked libraries in the "General module folder" in the Project Explorer can be called regardless of the application. The library function blocks of the linked libraries in the control node "Application" can be called in the respective application code only. Furthermore, placeholder libraries respond differently when downloading depending on their positions.

Libraries that are linked to a specific version in the project also have a placeholder for that version (placeholder library). You can define special placeholder resolutions. You can also use the placeholder resolution that is defined for a device in the device description or that is stored in the library repository for a library. The library manager notifies about the actual placeholder resolution and shows the version that is used when downloading an application (effective version).

If a library manager is linked cross-application in the project folder "General module folder", then you have global access to its contents. If placeholder libraries are linked, then only the placeholder resolutions in the device description or library repository are checked.

A library manager is normally in the "Application" node of the control. Then, only the application code below calls the library function blocks. Moreover, the special placeholder resolutions are checked first for placeholder libraries. Only after that are the placeholder resolutions checked that are in the device description or that originate from the library repository.

Refer to [chapter 18.1.14 "Object 'Library manager'" on page 516](#)

16.1 Information for library developers



To avoid consistency problems and to adequately support the user, adhere to certain rules when creating, referencing, encrypting, protecting and documenting libraries.

The following description provides an overview on the library development options. For a more detailed description of these topics, refer to the guidelines on library development "LibDevSummary".

General information

- Categories for libraries can be defined. The libraries are then displayed in the library repository below these categories.

- Define a namespace for a library to ensure unambiguous access to the integrated objects. For a unique access, add the namespace in front of the function block name:

`<namespace>.<variable name>`

Example: `AC.Module`

- To open the POU's of unencrypted libraries (*.library), double-click on the respective entry in the library manager.
- The following library types can be created:
 - *.library: Implementation library
 - *.compiled-library: Protected library: The source code can no longer be accessed
 - *._Itfs.library: Interface library
 - *._Cnt.library: Container library
- External libraries can be integrated into the application. External libraries are programmed outside of IndraLogic in a different programming language, for example C.

Library protection

- Source code protection:
When a library is prepared in "compiled library" format, the source code of the library modules is no longer visible after the library is integrated into a project.
- Signing:
Use a private key and a respective token to sign libraries. Go to the "Sign" tab and sign in the "Library info" object. Add the "Library info" object in the "General module folder" in the Project Explorer.

Library versions

- Multiple library versions can be simultaneously installed on the system.
- Multiple library versions can be simultaneously integrated into the project (not recommended).

Referenced libraries

- A library can be integrated into other libraries (referenced libraries). The nesting depth is arbitrary.
- It can be defined whether referenced libraries should be visible in the library manager.
- Referenced libraries can be integrated using library placeholders. This avoids problems that can occur due to version dependencies or the necessity to use vendor-specific libraries.



Using parameter lists is only permitted if the library is in the top position of the hierarchy.

16.2 Adding a library to the application

The following instruction describes how to integrate the `Util` library into your application.

1. Select the library manager and double-click to open it.
The library manager opens in the editor.

2. Select **Add library** in the library manager.
The **Add library** dialog opens.
3. Browse for the library by entering the string "util" in the input field above.
The `Util` library is displayed in the library view.
4. Select the `Util` library and close the dialog with **OK**.
The `Util` library is added to the library manager.

Also refer to

- [chapter ""Add library" command" on page 672](#)
- [chapter 16.3 "Adding a library to the repository" on page 221](#)

16.3 Adding a library to the repository

The following instruction describes how to install a library in the library repository.

1. Select **Tools ► Libraries ► Library repository**.
The **Library repository** dialog opens.
2. Click on **Install**.
3. Select the library to be installed. A file filter can be set.
Click on **Open**.
The library is added to the repository. The library can now be added to the library manager.

Also refer to

- [chapter 16.2 "Adding a library to the application" on page 220](#)

16.4 Exporting library file

A library can be exported from the library manager of a project or from the library repository and it can be saved as file on the hard disk.

1. Select **Tools ► Libraries ► Library repository**.
The **Library repository** dialog opens.
2. Select a library version in the window of the installed libraries.
3. Click on the "Export" button.
The "Export library" dialog opens.
4. If the selected library is not only integrated as compiled library, but also in the "Source format" in the project, both file types are included in the drop-down list in "File type". Otherwise, the filter automatically shows the present type: `Libraries (*.library)` or `Compiled libraries (*.compiled-library)`.
5. Select the file type and the storage location and click on "Save".

Also refer to:

- [chapter 18.1.14 "Object 'Library manager'" on page 516](#)
- [chapter ""Library repository" command" on page 636](#)

17 Reference "Programming"

17.1 Programming languages and editors

Program a POU in the editor for the implementation language selected when creating the POU. IndraLogic provides a text editor for ST and graphical editors for SFC, FBD/LD/IL and CFC.

To open the editor, double-click on the POU in the Project Explorer below **Application** or in the "General module folder".

Each programming editor consists of two subwindows:

- In the upper part, use the "declaration editor" to make declarations. Depending on the setting, declarations can either be made as text or table.
- Insert the implementation code of the respective language in the lower part.

Representation and response of each editor can be configured project-comprehensively in the respective tab of the IndraLogic options.

17.1.1 Declaration Editor

The declaration editor is used for declaring variables in the variable lists and function blocks.

If the declaration editor is used with an implementation language editor, it opens in a view above the implementation language editor.

The declaration editor provides two possible views: textual  and tabular .

In the dialog in **Tools ► IndraLogic 2G ► Options ► Declaration editor**, define whether only the textual view or only the tabular view is available, or whether it can be toggled between both views using the buttons on the right side of the editor view.

A rectangle selection is possible in the textual view of the declaration editor. You can find the key combination for the rectangle selection in the help page for the ST editor.

Also refer to

- [chapter 9.2 "Declaring variables " on page 83](#)
- [chapter 17.1 "Programming languages and editors" on page 223](#)
- [chapter ""Options" – "Declaration editor" dialog" on page 722](#)
- [chapter "ST editor" on page 225](#)

Declaration editor in online mode

The tabular view of the editor is shown in online mode.

The header always contains the current object path:

`<DeviceName>.<ApplicationName>.<ObjectName>.`

In addition to the columns visible in offline mode, the table also contains the columns **Value** and **Prepared value**.

- The **"Value"** column displays the actual value on the control; i.e. provides monitoring functionality. If the expression is an array, you can define the range of the array indices to be monitored. Double-click in the "Data type" column to open the "Monitoring area" dialog box and enter the "Start index" and the "End index" of the required range. Use a scroll bar to define a range of 1000 elements.

- The **"Prepared value"** column contains the value prepared for forcing or writing. Double-click on a field in the "Prepared value" column to directly enter a value for writing or forcing. In case of enumerations, a combo box opens to select a value from. In case of a Boolean variable, toggle the prepared value with <Enter> or the <space> bar. If an expression (variable) is of a structured data type, e.g. the instance of a function block or an array variable, a plus or a minus is placed in front.

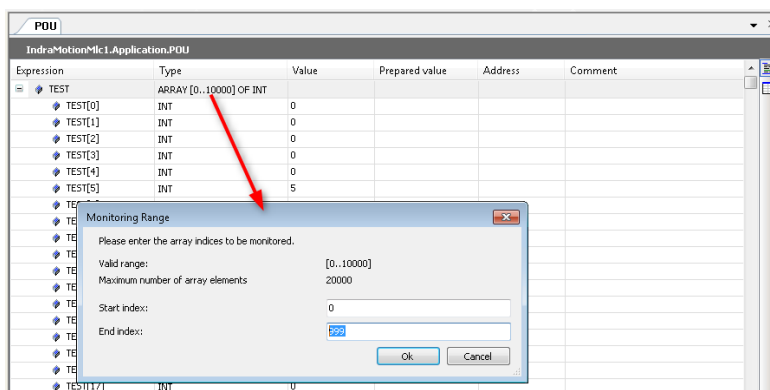
You can customize the format of the representation of floating-point values in the options for monitoring, see [chapter 18.3.5 "Dialog "Monitoring range""](#) on [page 704](#).



For arrays, the monitoring area is set to 1,000 array indices (0-999) by default.

The area can be extended up to 20,000 array elements if required.

To open the dialog to set the monitoring area, double-click on the column "Data type"



Also refer to

- [chapter 18.3.3 "Prepare value"](#) on [page 701](#)
- ["Forcing in the declaration section"](#) on [page 183](#)

17.1.2 Common functionalities in graphical editors

The implementation section of the graphical editors for FBD, LD, CFC and SFC contains a toolbar in the lower right corner:

	Return to normal editing mode: The mouse pointer changes back to the shape of the default arrow. Element can be selected and edited in the editor view.
	Panning tool: The mouse pointer changes to the shape of two crossed arrows. Click and drag anywhere in the editor view to move the visible area of the FBD/LD/IL editor or also pivot a CFC chart within the window.
	Magnification tool: A magnified window opens in the lower right corner of the editor view and the mouse pointer changes to the shape of a cross. When moving the cursor over your diagram, the magnification tool shows the area of the diagram under the cross at 100% magnification. Note: Click into the view and the magnification tool closes and the part of the diagram with the tool is displayed at 100% magnification. To retain the set zoom factor, use the default arrow for returning to the default editing mode.
	Zooming tool: The button opens a submenu to select a zoom factor. Select (. . .) to open the Zoom dialog to enter another value. The current zoom factor is always shown to the left of the symbol.

Zooming with the scroll wheel: Press <Ctrl>, move the scroll wheel and the zoom factor changes in steps of 10%.

By default, every graphical editor has its own toolbox (**View ► Other windows ► Tools**) on the right of the editor window.

The toolbox contains elements to drag to insertion points in the editor view.

IndraLogic highlights the insertion points with gray position flags in the shape of diamonds triangles, or arrows. These flags are green when the cursor moves over them. Release the mouse to insert the element at the selected position.

It is also possible to use the mouse for moving elements in the editor.

You can drag function block declarations in the FBD, LD, and CFC graphical editors to the editor view. To do this, select the full declaration (variable name and data type) and drag it to a suitable position in the editor view. In the ladder diagram, you can also drag Boolean declarations to the editor and insert them as contacts.

Also refer to

- [chapter "SFC \(sequential function chart\) editor" on page 236](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "CFC editor" on page 270](#)

17.1.3 Structured Text and Extended Structured Text (ExST)

ST editor

The ST editor is a text editor and is used for the implementation of code in Structured Text (ST) and Extended Structured Text (ExST).

The line numbering is located on the left-hand side of the editor.

The input help [F2] and the functionality "List components" provide support when entering programming elements.

If the cursor is over a variable, IndraLogic displays a tooltip with information on how to declare variables.

The box selection can be made with the following key combinations:

- [Ctrl] + [Right arrow]: The selected area is extended one position to the right.
- [Ctrl] + [Left arrow]: The selected area is extended one position to the left.
- [Ctrl] + [Up arrow]: The selected area is extended one position up.
- [Ctrl] + [Down arrow]: The selected area is extended one position down.

The behavior (for example parenthesizing, mouse actions, tabs) and appearance of the editor are configured in the IndraLogic options: **Tools ► Options ► IndraLogic 2G** in the category **Text editor**.

IndraLogic can detect syntax errors already when entered into the editor and then output a message as tooltip. Enable the setting "Underline errors in the editor" in the IndraLogic options. The setting is in the menu: **Tools ► Options ► IndraLogic 2G**, category "Intelligent coding".

Also refer to

- [chapter "Programming in Structured Text \(ST\)" on page 101](#)
- [chapter "ST expressions" on page 226](#)
- [chapter "ST - Comments" on page 235](#)

- [chapter 17.1.3 "Structured Text and Extended Structured Text \(ExST\)" on page 225](#)
- [chapter "'Options" dialog - Text editor" on page 731](#)

ST editor in online mode

In online mode, IndraLogic displays the variables and expressions in the ST editor.

Writing and forcing variables and expressions as well as debugging functions (breakpoints, single step execution) are also possible.

When using assignments as expressions in ST programming, no further breakpoint positions are created within a line.

Also refer to

- [chapter 13.1.1 "Calling monitoring in programming object" on page 191](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter 13.1.2 "Using watch lists" on page 196](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 12 "Testing and debugging" on page 179](#)
- [chapter 12.3 "Stepping a program" on page 181](#)
- [chapter 12.6 "Sequence control" on page 186](#)
- [chapter 12.7 "Determining the current processing position with the call stack" on page 189](#)

ST expressions

An expression is a construct that returns a value after its evaluation.

Expressions are composed of operators and operands.

In Extended Structured Text (ExST), assignments can also be used as expressions.

An operand can be a constant, a variable, a function call or another expression.

Example:

Examples

2014	(* Constant *)
ivar	(* Variable *)
fct(a,b)	(* Function call *)
(x*y)/z	(* Expression *)
real_var2 := int.var;	(* in ExST: Assignment *) *)

Also refer to

- [chapter "Programming in Structured Text \(ST\)" on page 101](#)

Evaluation of expressions

The evaluation of an expression is carried out by processing the operators according to specific rules for the order of operation (priority of tasks).

IndraLogic processes the operator with the strongest binding first. Operators with equal priority are processed from left to right.

Operation	Symbol	Binding strength
Bracketing	(Expression)	Highest priority
Function call	Function name (parameter list) All operators with syntax: <Operator> ()	
Exponentiation	EXPT	
Negation Complementation	- NOT	
Multiply Divide Modulo	* / MOD	
Add Subtract	+ -	
Compare	<,>,<=,>=	
Equality Inequality	= <>	
Bool AND	AND AND_THEN	
Bool XOR	XOR	
Bool OR	OR OR_ELSE	Lowest priority

Also refer to

- [chapter 17.3 "Operators" on page 294](#)

Assignments

ST assignment operator

Syntax:

<Operand> := <Expression>

This assignment operator executes the same function as the `MOVE` operator.

Also refer to

- [chapter 17.3.6 "Operator "MOVE"" on page 301](#)

ST assignment operator for outputs

The assignment operator `=>` assigns the output of a function, a function block and a method to a variable. The place on the right of the operator can also be empty.

Syntax: <output> => <variable>

Example:

```
FBcomp_Output1 => bVar1;
FBcomp_Output2 => ;
```

FBcom_Output1 and FB_Output2 are outputs of a function block. The FBcom_Output1 value is assigned to the variable bVar1.

ExST assignment 'S='

When the operand of the Set assignment switches to **TRUE**, then **TRUE** is assigned to the variable to the left of the operator. The variable is set.

<variable name> S= <operand name> ;

The variables and the operand have the data type **BOOL**.

Example

```
PROGRAM PLC_PRG
VAR
    xOperand: BOOL := FALSE;
    xSetVariable: BOOL := FALSE;
END_VAR

xSetVariable S= xOperand;
```

When the operand **xOperand** switches from **FALSE** to **TRUE**, then **TRUE** is also assigned to the variable **xSetVariable**. But then the variable keeps this state, even if the operand continues to change its state.

Multiple assignments

In the case of multiple assignments within a code line, the individual assignments are not processed from right to left, but all assignments refer to the operands at the end of the code line.

Example

```
FUNCTION funCompute : BOOL
VAR_INPUT
    xIn : BOOL;
END_VAR
IF xIn = TRUE THEN
    funCompute := TRUE;
    RETURN;
END_IF

PROGRAM PLC_PRG
VAR
    xSetVariable: BOOL;
    xResetVariable: BOOL := TRUE;
    xVar: BOOL;
END_VAR
xSetVariable S= xResetVariable R= funCompute(xIn := xVar);
```

The variable **xResetVariable** gets the **R=** assignment of the return value of **funCompute**.

The variable **xSetVariable** gets the **S=** assignment of the return value of **funCompute**, but not from **xResetVariable**.

Also refer to

- [chapter 9.3.3 "Structured Text \(ST\) and Extended Structured Text \(ExST\)" on page 101](#)
- [chapter "ExST assignment 'R='" on page 228](#)

ExST assignment 'R='

When the operand of the Reset assignment switches to **TRUE**, then **FALSE** is assigned to the variable to the left of the operator. The variable is reset.

<variable name> R= <operand name> ;

Example

```
VAR
    xOperand: BOOL := FALSE;
```

```

    xResetVariable: BOOL := TRUE;
END_VAR

xResetVariable R= xOperand;

```

Multiple assignments

When the operand `xOperand` switches from FALSE to TRUE, then FALSE is also assigned to the variable `xResetVariable`. But then the variable keeps its state, even if the operand continues to change its state.

In the case of multiple assignments within a code line, the individual assignments are not processed from right to left, but all assignments refer to the operands at the end of the code line.

Example

```

FUNCTION funCompute : BOOL
VAR_INPUT
    xIn : BOOL;
END_VAR
IF xIn = TRUE THEN
    funCompute := TRUE;
    RETURN;
END_IF

PROGRAM PLC_PRG
VAR
    xSetVariable: BOOL;
    xResetVariable: BOOL := TRUE;
    xVar: BOOL;
END_VAR
xSetVariable S= xResetVariable R= funCompute(xIn := xVar);

```

The variable `xResetVariable` gets the R= assignment of the return value of `funCompute`.

The variable `xSetVariable` gets the S= assignment of the return value of `funCompute`, but not from `xResetVariable`.

Also refer to

- [chapter 9.3.3 "Structured Text \(ST\) and Extended Structured Text \(ExST\)" on page 101](#)
- [chapter "ExST assignment 'S='" on page 228](#)

ExST assignment "S=" as expression

In ExST, as an extension to the IEC 61131-3 standard, IndraLogic permits the use of assignments as expressions.

Example

<code>int_var1:= int_var2:= int_var3 + 9;</code>	(* int_var1 and int_var2 receive the value of int_var3 + 9 *)
<code>real_var1:= real_var2:= int_var;</code>	(* real_var1 and real_var2 obtain the value of int_var *)
<code>int_var:= real_var1:= int_var;</code>	(* incorrect assignment, data types do not match! *)
<pre> IF b:= (i = 1) THEN i:= i + 1; END_IF </pre>	

Also refer to

- [chapter 9.3.3 "Structured Text \(ST\) and Extended Structured Text \(ExST\)" on page 101](#)

Assignment operator 'REF'

The operator generates a reference (pointer) to a value.

Syntax: REF=<Variable>

Example:

```
A : REFERENCE TO DUT;
B : DUT;
C : DUT;

A REF= B; // corresponds to A := ADR(B);
A := C; // corresponds to A^ := C;
```

Also refer to [chapter 17.5.13 "Data type 'REFERENCE'"](#) on page 361

Statements

ST statement "IF"

The IF statement is used for checking a condition and, depending on this condition, for executing the subsequent statements.

A condition is coded as an expression that returns a Boolean value. If the expression returns TRUE, then the condition is fulfilled and the corresponding statements are executed after THEN. If the expression returns FALSE, then the following conditions, which are identified with ELSIF, are evaluated. If an ELSIF condition returns TRUE, then the statements are executed after the corresponding THEN. If all conditions return FALSE, then the statements are executed after ELSE.

Therefore, at most one branch of the IF statement is executed. ELSIF branches and the ELSE branch are optional.

Syntax:

```
IF <condition> THEN
  <statements>
( ELSIF <condition> THEN
  <statements> ) *
( ELSE
  <statements> ) ?
END IF;
// ( ... ) * None, once or several times
// ( ... ) ? Optional
```

Example

```
PROGRAM PLC_PRG
VAR
  iTemp: INT;
  xHeatingOn: BOOL;
  xOpenWindow: BOOL;
END_VAR

IF iTemp < 17 THEN
  xHeatingOn := TRUE;
ELSIF iTemp > 25 THEN
  xOpenWindow := TRUE;
ELSE xHeatingOn := FALSE;
END_IF;
```

The program is run as follows at runtime:

- For the evaluation of the expression `iTemp < 17 = TRUE`, the subsequent statement is executed and the heating is switched on (`xHeatingOn := TRUE`).
- For the evaluation of the expression `iTemp < 17 = FALSE`, the subsequent ELSIF condition `iTemp > 25` is evaluated.

If the condition is true `iTemp > 25`, the statements are executed under `ELSIF` and the window is open (`xOpenWindow := TRUE`).

- If all conditions are `FALSE`, then the statement is executed in `ELSE` and the heating is switched off (`xHeatingOn := FALSE`).

Also refer to

- [chapter "ST statement "CASE"" on page 232](#)

ST statement "FOR"

The `FOR` loop is used to execute statements with a certain number of repetitions.

Syntax:

```
FOR <counter> := <start value> TO <end value> {BY <increment> } DO
  <instructions>
END_FOR;
```

The section inside the curly parentheses {} is optional.

IndraLogic executes the `<instructions>` as long as the `<counter>` is not greater - or in case of a negative increment increment not smaller - than the `<end value>`. This is checked before executing the `<instructions>`.

Every time the statements `<statements>` are executed, the counter `<counter>` is automatically increased by the increment `<increment>`. The increment `<increment>` can have any integral value. If no increment is specified, the standard increment is 1.

Example

```
FOR iCounter := 1 TO 5 BY 1 DO
  iVar1 := iVar1*2;
END_FOR;
Erg := iVar1;
```

If `iVar1` was preassigned with 1, `iVar1` is 32 after the `FOR` loop.

CAUTION

Endless loop

The end value `<end value>` may not attain the same value as the upper limit of the data type of the counter.

If the end value of the counter is equal to the upper limit of the data type of the counter, an infinite loop results.

For example, an infinite loop results in the above example if `iCounter` is of the data type `SINT` and the `<end value>` equals 127, since the data type `SINT` has the upper limit 127.

The `EXIT` statement is used to end the loop irrespective of other abort conditions.

As an extension to the IEC 61131-3 standard, use the `CONTINUE` statement within the `FOR` loop.

Also refer to

- [chapter 17.5.2 "Integer data types" on page 352](#)
- [chapter "ST statement "EXIT"" on page 235](#)
- [chapter "ST statement "IF"" on page 230](#)
- [chapter "ST statement "REPEAT"" on page 233](#)

- [chapter "ST statement "WHILE"" on page 232](#)
- [chapter "EXST statement "CONTINUE"" on page 235](#)

ST statement "CASE"

. The CASE statement is used to summarize several conditional statements with the same conditional variables in a construct.

Syntax:

```

CASE <Var1> OF
<value1>:<instruction1>
<value2>:<instruction2>
<value3, value4, value5>:<instruction3>
<value6 ... value10>:<instruction4>
...
<value n>:<instruction n>

{ELSE <ELSE-instruction>}

END_CASE;
```

The section inside the curly parentheses {} is optional.

Processing scheme of a CASE statement

- If the variable <Var1> is <<value i>, the statement <statement i> is executed .
- If the variable <Var1> does not have any of the specified values, the <ELSE instruction> is executed.
- If the same statement is to be executed for several values of the variable, write these values one behind the other, separated by commas.

Example

```

CASE iVar OF
1, 5: bVar1 := TRUE;
      bVar3 := FALSE;

2: bVar2 := FALSE;
   bVar3 := TRUE;

10..20: bVar1 := TRUE;
        bVar3:= TRUE;
ELSE
      bVar1 := NOT bVar1;
      bVar2 := bVar1 OR bVar2;
END_CASE;
```

Also refer to

- [chapter "ST statement "IF"" on page 230](#)

ST statement "WHILE"

The WHILE loop is used like the FOR loop to execute statements several times until the abort condition. The abort condition of a WHILE loop is a Boolean expression.

Syntax:

```

WHILE <boolean expression> DO
  <instructions>
END_WHILE;
```

IndraLogic repeatedly executes the <statements> as long as the <BooleanExpression> returns TRUE.

If the Boolean expression is already FALSE at the first evaluation, IndraLogic never executes the statements.

If the Boolean expression never assumes `FALSE`, the statements are repeated endlessly resulting in a runtime error.

Example

```
WHILE iCounter <> 0 DO
  Var1 := Var1*2
  iCounter := iCounter-1;
END_WHILE;
```



Ensure programmatically that no infinite loops are caused.

In a certain way, the `WHILE` and `REPEAT` loops are more powerful than the `FOR` loop, since the number of executions of the loop do not have to be known before its execution. In some cases, it is thus only possible to work with these two types of loop. However, if the number of loop executions is known, a `FOR` loop is to be preferred to avoid infinite loops.

The `EXIT` statement is used to end the loop irrespective of other abort conditions.

As an extension to the IEC 61131-3 standard, use the `CONTINUE` statement within the `WHILE` loop.

Also refer to

- [chapter "ST statement "EXIT"" on page 235](#)
- [chapter "ST statement "IF"" on page 230](#)
- [chapter "ST statement "FOR"" on page 231](#)
- [chapter "ST statement "REPEAT"" on page 233](#)
- [chapter "EXST statement "CONTINUE"" on page 235](#)

ST statement "REPEAT"

The `REPEAT` loop is used like the `WHILE` loop, but IndraLogic only checks the abort condition after the execution of the loop. Thus, the `REPEAT` loop is executed at least once, regardless of the abort condition.

Syntax:

```
REPEAT
  <instructions>
UNTIL <boolean expression>
END_REPEAT;
```

IndraLogic executes the `<Statements>` until the `<BooleanExpression>` returns `TRUE`.

If the Boolean expression is already `TRUE` at the first evaluation, IndraLogic executes the statements once.

If the Boolean expression never assumes `TRUE`, the statements are repeated endlessly resulting a runtime error caused by the program.

Example

```
REPEAT
  Var1 := Var1*2;
  iCounter := iCounter-1;
UNTIL
  iCounter = 0
END_REPEAT;
```

In a certain way, the `WHILE` and `REPEAT` loops are more powerful than the `FOR` loop, since the number of executions of the loop do not have to be

known before its execution. In some cases, only working with these two kinds of loop is possible. However, if the number of loop executions is known, a `FOR` loop is to be preferred to avoid infinite loops.

The `EXIT` statement is used to end the loop irrespective of other abort conditions.

As an extension to the IEC 61131-3 standard, use the `CONTINUE` statement within the `WHILE` loop.

Also refer to

- [chapter "ST statement "EXIT"" on page 235](#)
- [chapter "ST statement "FOR"" on page 231](#)
- [chapter "ST statement "IF"" on page 230](#)
- [chapter "ST statement "WHILE"" on page 232](#)
- [chapter "EXST statement "CONTINUE"" on page 235](#)

ST statement "RETURN"

Use the `RETURN` statement in order to exit from a function block. Perform this with regard to a condition for example.

Example

```
IF xIsDone = TRUE THEN
    RETURN;
END_IF;

iCounter := iCounter + 1;
```

If the value of `xIsDone` is equal to `TRUE`, then the function block is exited immediately and the statement `iCounter := iCounter + 1;` is not executed.

Refer to [chapter "ST statement "IF"" on page 230](#)

ST statement "JMP"

The `JMP` statement is used to execute an unconditional jump to a program line that is marked by a jump label.

Syntax:

```
<label>: <instructions>

JMP <label>;
```

The jump label `<label>` is any unique identifier to place it at the beginning of a program line.

When reaching the `JMP` statement, a return to the program line takes place with the `<Label>`.

Example

```
iVar1 := 0;

_label1: iVar1 := iVar1+1;

(*instructions*)

IF (iVar1 < 10) THEN
    JMP _label1;
END_IF;
```



Ensure programmatically that no infinite loops are caused. A jump can be linked to a condition for example.

ST statement "EXIT"

The **EXIT** statement is used in a **FOR**, **WHILE** or **REPEAT** loop to end the loop regardless of other abort conditions.

Also refer to

- [chapter "ST statement "FOR"" on page 231](#)
- [chapter "ST statement "WHILE"" on page 232](#)
- [chapter "ST statement "REPEAT"" on page 233](#)

EXST statement "CONTINUE"

CONTINUE is an statement of the Extended Structured Text (ExST).

The statement is used inside **FOR**, **WHILE** and **REPEAT** loops to jump to the beginning of the next execution of the loop.

Example

```
// SUM= 15 IF FLAG= FALSE
// SUM= 9 IF FLAG= TRUE
SUM:= 0;
FOR I:= 1 TO 3 DO
  FOR J:= 1 TO 2 DO
    SUM:= SUM + 1;
    IF FLAG THEN
      CONTINUE;
    END_IF;
    SUM:= SUM + 1;
  END_FOR;
  SUM:= SUM + 1;
END_FOR;
```

Also refer to

- [chapter "ST statement "FOR"" on page 231](#)
- [chapter "ST statement "WHILE"" on page 232](#)
- [chapter "ST statement "REPEAT"" on page 233](#)

ST - Calling function blocks

Syntax

```
<FB-instance>(<FB input variable>:=<value or address> | <other FB input variables>);
```

Example

```
TMR:TON;

TMR (IN:=%OX5, PT:=T#300ms);
varA:=TMR.Q;
```

The timer function block **TON** is instantiated in **TMR:TON** and called with assignments for the parameters **IN** and **PT**.

The output **Q** is addressed with **TMR.Q** and assigned to the variable **varA**.

Also refer to

- [chapter ""Function block" object \(FUNCTION_BLOCK\)" on page 523](#)

ST - Comments

Comment	Description	Example
Single-line	Starts with // and ends at the end of the line.	// This is a comment
Multi-line	Starts with (* and ends with *).	(* This is a multi-line comment *)
Nested	Starts with (* and ends with *). There may be further (*. . .*) comments inside this comment.	(* a:=inst.out; (* 1. comment *) b:=b+1; (* 2. comment *) *)

17.1.4 Sequential function chart (SFC)

SFC (sequential function chart) editor

The SFC editor is a graphical editor. A new SFC-POU includes an init step and a subsequent transition.

In the SFC editor, individual elements can be inserted into the diagram using commands of the "SFC" menu, the context menu and the "Tools" view.

When inserting via a menu item, the elements that can be inserted at the currently selected position can be selected.

To parallelly insert branchings for multiple actions or transitions, select these actions and transitions by multiple selections.

SFC elements can also be positioned in the diagram via drag&drop from the "Tools" view.

When dragging an element to the editor, IndraLogic marks all possible positions for insertion with grey rectangles.

When moving the mouse via a possible position for insertion, the color of the rectangle becomes green. Release the mouse button to insert the object at this location.

If a branching is inserted via drag&drop, select the beginning and the end of the branching using the mouse. To highlight the beginning of the branching, release the mouse on the insertion position. The color of the rectangle becomes red.

Click on the second insertion position and specify the end of the branching. IndraLogic inserts a branching around the objects between the highlighting of beginning and of the end.

To copy step and transition elements calling action or transition objects, two different duplication modes can be set. Either the references are also copied or the referenced objects are "embedded" and duplicated when copying.

Define appearance and response of the editor in the IndraLogic options in the **SFC editor** category.

Also refer to

- [chapter 17.1.2 "Common functionalities in graphical editors" on page 224](#)
- [chapter 17.1.4 "Sequential function chart \(SFC\)" on page 236](#)
- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)
- [chapter ""Options" - "SFC editor" dialog" on page 728](#)

SFC editor in online mode

In the SFC editor, display the variables and expressions used on the control at runtime if required.

Variables and expressions can also be written and forced.

A debugging functionality (breakpoints, step-by-step execution, etc.) is not yet available.

Set the online representation of the SFC elements and attributes in the options of the SFC editor.

If SFC flags were declared explicitly, they are displayed in the declaration section in online mode. They are not displayed in offline mode.



Please also note the processing sequence of the elements in an SFC diagram.

In online mode, IndraLogic displays the active steps in blue.

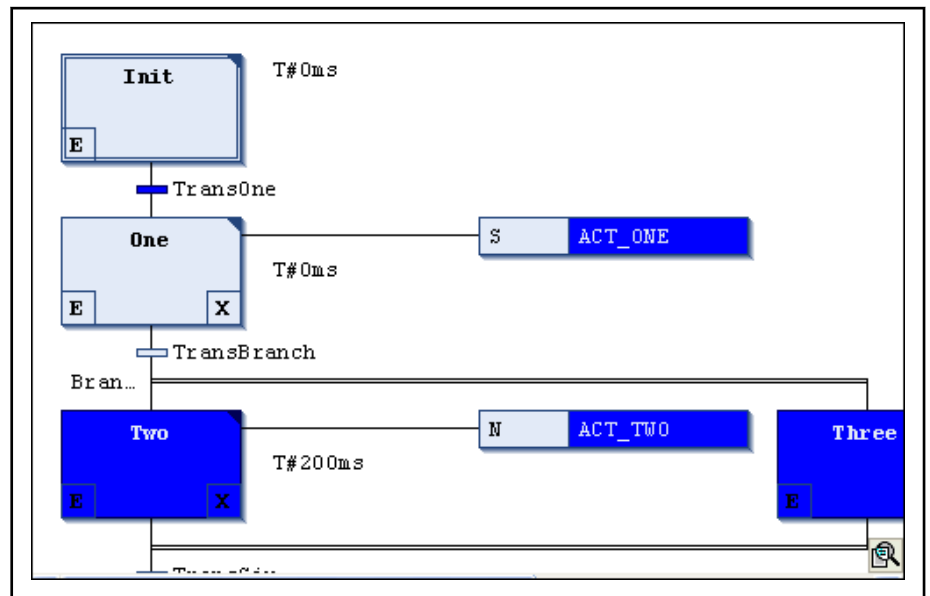


Fig. 17-1:

Also refer to

- [chapter "Implicit variables" on page 240](#)
- [chapter 9.2.1 "Using the declaration editor" on page 85](#)
- [chapter "Processing sequence in SFC" on page 237](#)
- [chapter "'Options' - 'SFC editor' dialog" on page 728](#)

Processing sequence in SFC

Basic response of the elements

- **Active step:** An active step includes actions currently executed. In online mode, IndraLogic displays the active steps in blue.
- **Initial step:** In the first cycle after calling a POU in SFC, the initial step is activated automatically and the step action is executed.
- **IEC actions** IndraLogic executes IEC actions at least twice. The first time after the step has been enabled, the second time but only in the next cycle after the step has been disabled.
- **Alternative branchings:** If the step preceding the branching is active, IndraLogic evaluates the first transition of each of the alternative branches from left to right.

IndraLogic enables the subsequent step in the first branch with a transition providing `TRUE`.

- **Parallel branchings:** If the step before the branch (horizontal double line) is active and the transition before the branch is `TRUE`, IndraLogic activates the first steps in every branch. Subsequently, the branches are processed simultaneously. The step following the end of the branching (after the horizontal double line) is enabled if all the final steps in the branches are active and the transition after the double line returns `TRUE`.

Processing order:

1. Resetting IEC actions

IndraLogic resets the internal action check flags of the action qualifiers (N, R, S, L, D, P, SD, DS, SL). These flags control IEC actions. However, flags are not reset when they are called within actions!

2. Executing exit actions

IndraLogic verifies whether all steps fulfill the condition for executing the exit action for each step. The order of verification follows the layout in the SFC diagram, from top to bottom and from left to right.

IndraLogic executes an exit action when the step is disabled (after any entry actions and step actions have been executed in the preceding cycle and the condition for the subsequent step becomes `TRUE`).

3. Executing entry actions

IndraLogic verifies whether all steps fulfill the condition for executing the entry action for each step.

The order of verification follows the layout in the SFC diagram, from top to bottom and from left to right.

If the conditions are fulfilled, IndraLogic executes the entry actions.

IndraLogic executes an entry action as soon as the processing reached the transition preceding the step and this transition becomes `TRUE` and, the step is enabled.

4. Time check / Execute step actions

IndraLogic executes the following for all steps in the order arranged in the SFC diagram:

- IndraLogic copies the time elapsed for the active step to the corresponding implicit step variable `<StepName>.t` (not yet implemented)
- If a timeout occurs, IndraLogic sets the respective error flags (not yet implemented)
- For non-IEC steps, IndraLogic executes the step action.

5. Executing IEC actions

IndraLogic executes the IEC actions in alphabetical order. This is carried out in two cycles by the list of actions. In the first run, IndraLogic executes the IEC actions for all steps disabled in the preceding cycle. In the second cycle, the IEC actions for all active steps are executed.

6. Transition check, enabling the following steps

The following transitions are analyzed: If a step is active in the current cycle and the subsequent transition is `TRUE` and any defined minimum time of the step has elapsed, the subsequent step is activated.



Please note when implementing actions:

An action can be executed multiple times within the same cycle if used in multiple SFC diagrams. Example: An SFC contains two IEC actions A and B, both programmed in the SFC language and both calling an IEC action C. In this case, C would be called twice.

If the same IEC action is used at the same time in different levels of an SFC diagram, this can lead to unpredictable results when processing. Thus, IndraLogic outputs a corresponding error message. This error message can be displayed for projects created in an earlier version of the development system.



Note that implicit variables can be used to monitor the processing status of steps and actions or to check the processing.

Also refer to

- [chapter "Implicit variables" on page 240](#)
- [chapter "Qualifiers for SFC actions" on page 239](#)

Qualifiers for SFC actions

You assign qualifiers to IEC steps. Qualifiers describe how a step action is processed.

Qualifiers are processed by the `SFCActionControl` function block in `IecSfc.library`. The library is automatically integrated into the project.

N	Non-stored	The action is active as long as the step is active.
R	overriding Reset	The action is disabled.
S	Set (Stored)	IndraLogic executes this action as soon as the step is active. The action is executed even if the step was already disabled until it obtains a reset.
L	Time Limited	IndraLogic executes this action as soon as the step is active. The action is executed until the step becomes inactive or the specified time elapsed.
D	Time Delayed	IndraLogic begins executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until the step is disabled.
P	Pulse	IndraLogic executes this action twice: One time when the step is activated and one time when the step is deactivated.
SD	Stored and time delayed	IndraLogic begins executing the action only after the given delay time has elapsed following step activation. The action is executed until it is reset.
DS	Delayed and stored	IndraLogic begins executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until it is reset.
SL	Stored and time limited	IndraLogic executes this action as soon as the step is enabled. It is executed until the given time period has expired or it is reset.

Tab. 17-1: Available qualifiers

Enter the times for the L, D, SD, DS and SL qualifiers in the format of a TIME constant.



If an IEC action is disabled, it is executed a second time. This means that IndraLogic executes this kind of action at least twice. This also applies to actions with the P qualifier.

Also refer to

- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)

Implicit variables

Every SFC object supplies implicit variables for you to monitor the status of steps and IEC actions at runtime.

These implicit variables are declared automatically for each step and each IEC action by the IndraLogic.

The implicit variables are structural instances of the type `SFCStepType` for steps and type `SFCActionType` for actions.

The variables have the same names as their elements, for example "step1" variable name for "step1" step name.

The structural components describe the status of a step or action or the current elapsed time in an active step.



In the element properties, define whether IndraLogic should export a symbol definition for this flag to the symbol configuration.

Also refer to

- [chapter "SFC element properties" on page 252](#)

Step and action status

The syntax for the implicitly executed variable declaration:

```
<StepName>:SFCStepType;  
_  
_<actionname>:SFCActionType;
```

Step	
<StepName>.x	Shows the activation state in the current cycle. If <StepName>.x = TRUE, IndraLogic processes the step in the current cycle.
<StepName>._x	Shows the activation state for the next cycle. If <StepName>._x = TRUE and <StepName>.x = FALSE, IndraLogic executes the step in the next cycle, i.e. <StepName>._x is copied to <StepName>.x at the beginning of a cycle.
<StepName>.t	The t flag returns the current time period that has elapsed since the step was enabled. This applies only to steps irrespective of whether a minimum time has been defined in the step properties. See also <code>SFCError</code> SFC flag.
<StepName>._t	For internal use only
IEC action	
_<ActionName>.x	TRUE when the action is executed.
_<ActionName>._x	TRUE when the action is active.

Tab. 17-2: The following implicit variables are available for the step status or the IEC action status:



Use the above variables to force a specific status to a step (activate a step). However, note that this causes an uncontrolled status of the complete SFC!

Also refer to

- [chapter "SFC flags" on page 241](#)

Access to implicit variables

Syntax for the access:

Assign implicit variable directly in the POU:

`<VariableName>:=<StepName>.<ImplicitVariable>` or

`<VariableName>:=_<ActionName>.<ImplicitVariable>`

Example:

```
status:=step1._x;
```

From another function block with POU name:

`<VariableName>:=<POU-Name>.<StepName>.<ImplicitVariable>`

or

`<VariableName>:=<POU-Name>._<ActionName>.<ImplicitVariable>`

Example:

```
status:=SFC_prog.step1._x;
```

Symbol generation

In the element properties of a step or action, define whether IndraLogic should add a symbol definition for the step flag or the action flag.

In the **Properties** view, select the necessary access rights in the **Symbol** column.

Also refer to

- [chapter "SFC element properties" on page 252](#)

SFC flags

SFC flags are implicitly generated variables with predefined names. They can be used for controlling the processing of an SFC diagram. Use these flags to display timeouts or reset SFCs for example. In addition, enable the jog mode specifically to enable transitions. Declare and enable these variables to access.

Implicit creation of SFC flags

IndraLogic declares SFC flags automatically when enabling the respective options. Set this option in the **SFC settings** tab of the "Properties" dialog for each POU, or in the **SFC** project settings dialog for each SFC-POU in the project.



The SFC settings for the SFC flags of individual POUs are only applied if the option **Use defaults** is not selected.

Select this option and the settings defined in the project settings apply.



SFC flags declared in the SFC settings dialog are only visible in the online view of the SFC function block!

Also refer to

- ["Variables \("SFC flags"\)" on page 712](#)

Explicit creation of SFC flags

Manual declaration, which was necessary in IndraLogic 1x, is now only required to enable write access from another function block.

In this case, note the following:

When declaring the flag in a Global Variable List, disable its setting **Declare** in the SFC settings dialog.

If not, a local SFC flag is implicitly declared. IndraLogic uses it instead of the global variable!

Use case for SFCError

Example:

An SFC block named `sfc1` with an `s1` step was created. You defined time-outs in the step properties.

To select the times to be displayed, go to **Tools ▶ Options ▶ IndraLogic2G ▶ SFC editor "View"**

Time monitoring is enabled by forcing the variables `SFCEnableLimit`.

If the `s1` step remains longer active than its time properties have permitted (timeout), IndraLogic sets the `SFCError` flag to permit access by the application.

To permit access, declare and enable the SFC flag in the SFC settings. If it is only declared, the SFC flag is only displayed in the online view of `sfc1` in the declaration section, but it has no function.

☐	SFCReset	☐
☒	SFCError	☒
☒	SFCEnableLimit	☒
☒	SFCErrorStep	☒
☐	SFCErrorPOU	☐

Fig. 17-2:

Now, the SFC flag can be addressed within the POU, for example in an action (2) or outside of the function block (1).

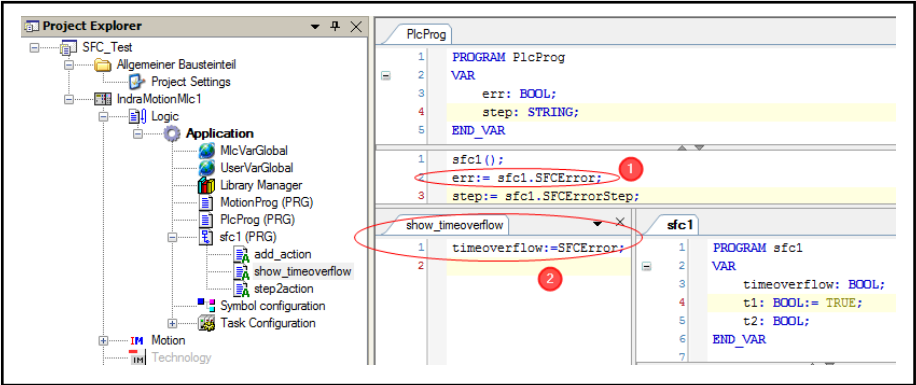


Fig. 17-3:

Online view of the SFC function block `sfc1`.

Minimum step time of the step `s1`: `T#0s`, maximum step time of the step `s1`: `T#5s`.

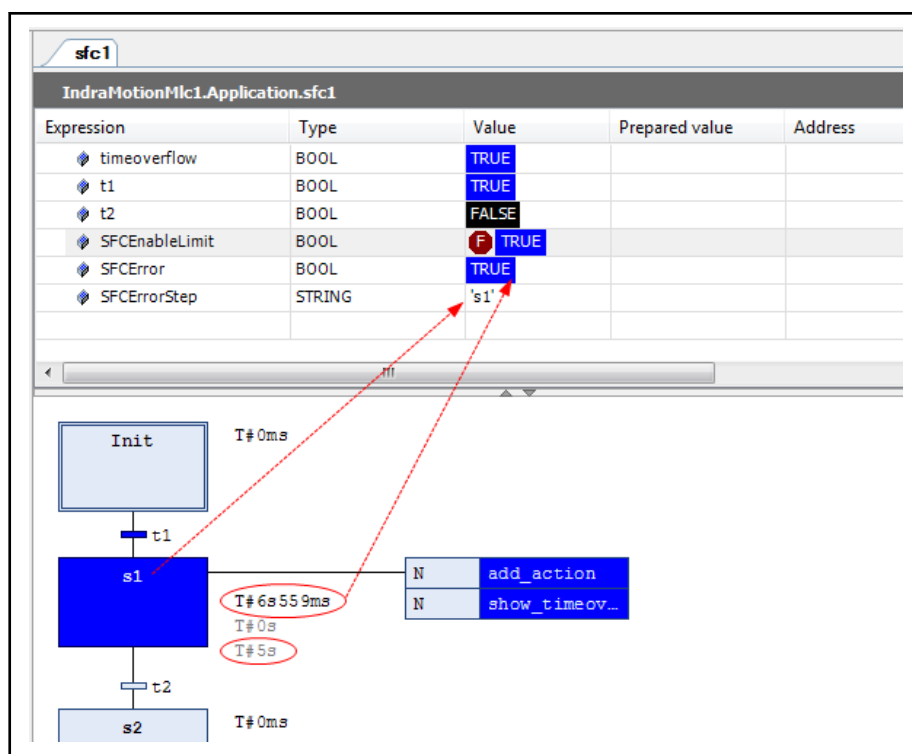


Fig. 17-4:

SFCError becomes TRUE as soon as a timeout occurs within step s1.

Access to the flags

Syntax for the access:

Assign the flag directly in the POU: <VariableName>:=<SFC Flag>

Example:

Example

```
checkerror:=SFCError;
```

From another function block with POU name: <VariableName>:=<POU Name>.<SFC Flag>

Example:

Example:

```
checkerror:=SFC_prog.SFCError;
```

If write access from another block is required, also declare the SFC flag explicitly as a VAR_INPUT variable in the SFC function block or globally in a GVL.

Example

Local declaration:

```
PROGRAM SFC_prog
VAR_INPUT
    SFCinit:BOOL;
END_VAR
```

or

Reference "Programming"

Global declaration in a Global Variable List:

```
VAR_GLOBAL
    SFCinit:BOOL;
END_VAR
```

Write access to SFCinit in SFC_prog

```
PROGRAM PlcProg
VAR
    setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;
```

SFC flags

Name	Data type	Description
SFCInit	Bool	TRUE: IndraLogic resets the procedure to the initial step. The other SFC flags are also reset (initialized). As long as the variable is TRUE, the initial step remains set (active), but it is not executed. Only when SFCInit is set to FALSE again, the function block continues to be processed normally.
SFCReset	Bool	Similar to SFCInit. In contrast, IndraLogic processes the initialization after the initial step. For example, in the initial step, reset the SFCReset flag can be reset immediately to FALSE.
SFCError	Bool	It becomes TRUE if a timeout occurred in an SFC diagram. If there is another timeout in the program after the first one, the second one is no longer registered if the variable SFCError was not reset before. The declaration of SFCError is required for the functioning of the other flag variables to check the time sequence (SFCErrorStep, SFCErrorPOU, SFCQuitError).
SFCEnableLimit	Bool	It is used for switching on (TRUE) or off (FALSE) the check for timeouts in the steps by SFCError. If this variable (SFC settings) is declared and enabled, set it to TRUE for the SFCError to operate. If not, the timeouts are ignored. This can be useful during commissioning or manual operation for example. If the variable is not declared, SFCError operated automatically. The prerequisite is the declaration of SFCError.
SFCErrorStep	String	It stores the name of the step that caused a timeout registered by SFCError. The prerequisite is the declaration of SFCError.
SFCErrorPOU	String	In case of a timeout, it stores the name of the function block in which a timeout occurred and registered by SFCError. The prerequisite is the declaration of SFCError.
SFCQuitError	Bool	As long as this Boolean variable is TRUE, IndraLogic stops the processing of the SFC diagram and any timeout in the SFCError variable is reset. If the variable is set to FALSE again, all of the previous time periods in the active steps are reset. The prerequisite is the declaration of SFCError.
SFCPause	Bool	As long as this variable is TRUE, IndraLogic stops the processing of the SFC diagram.
SFCTrans	Bool	TRUE if a transition is active.
SFCCurrent-Step	String	Shows the name of the active step, regardless of the time monitoring. In case of a parallel branch, the name of the step of the outermost branch at the right is always saved.
SFCTip, SFCTipMode	Bool	Allow the "jog mode" in an SFC function block. If the jog mode is enabled by SFCTipMode=TRUE SFCTip has to be set to TRUE to switch to the next step. As long as SFCTipMode is set to FALSE, transitions can also be used to switch to the next step.

Elements

SFC elements "steps" and "transition"

Symbol "Step" , Symbol "Transition" 

IndraLogic always inserts steps and transitions as a combination.

When entering a steps without transition or a transition without step, a compilation error results.

To modify it, double-click on the name.

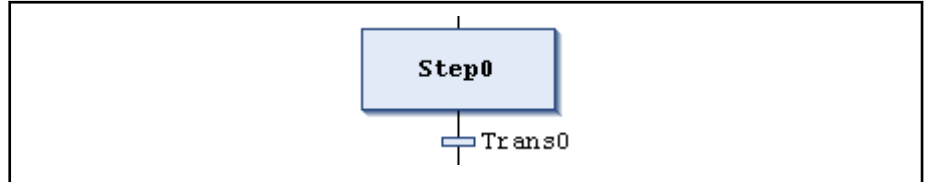


Fig. 17-5:



Step names have to be unique within the valid range of the "father" function block.

Consider this especially when using actions that were also programmed in SFC.

Please note that a step can be converted into an initial step. Click on **Init step** or set the respective property in the SFC properties.

All steps are defined by the step properties. They can be displayed and edited in the **Properties** view, depending on the set options.

Add the actions to be executed to the step when the step is active. There are "IEC actions" and "step actions". For details, refer to the chapter on the SFC element "Action".

A transition has to include the condition when the subsequent step should be active as soon as the condition becomes TRUE. Therefore, a transition condition has to be TRUE or FALSE.

It can be defined in two ways:

- (1) Direct ("Inline condition"): They replace the default transition name by a Boolean variable, a Boolean address, a Boolean constant or an statement with Boolean result.

No programs, function blocks or assignments may be specified here.

- (2) Using separate transition object or property object ("Multi-use condition"):

They replace the default transition name by the name of a transition object or property object (, ).

Create these objects via **Add**. This allows "multiple use" of transitions

Like an "inline condition", the object can contain a Boolean variable, a Boolean address, a Boolean constant, or an statement with a Boolean result. In addition, it can also contain multiple statements with any code.

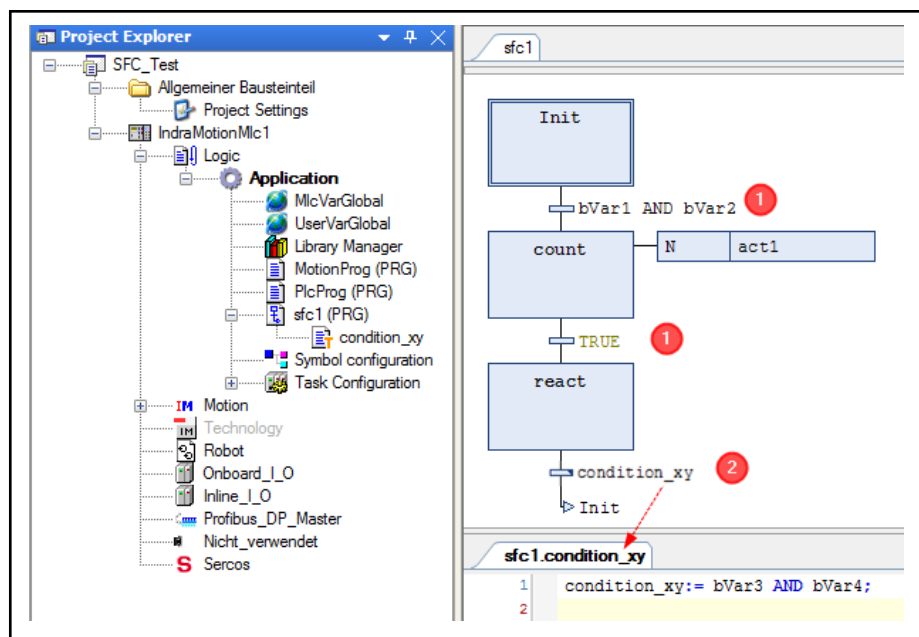


Fig. 17-6:



The user is responsible for assigning the required expression to a transition variable if the transition includes multiple statements!

Transitions referencing a transition or a property object are marked with a small triangle in the upper right corner of the transition box.

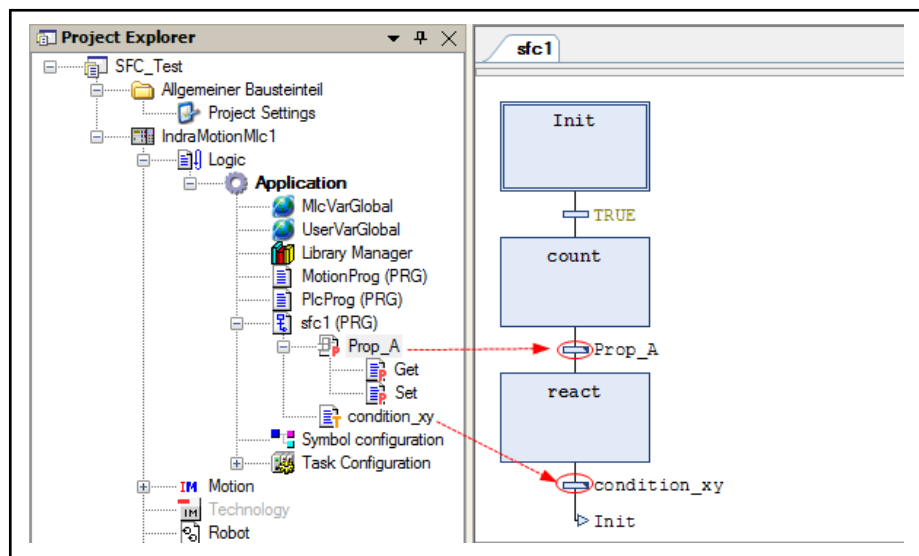


Fig. 17-7:

As opposed to IndraLogic 1x, IndraLogic 2G now handles a transition condition like a method call.

Input acc. to the following syntax:

<TransitionName>:=<TransitionCondition>

(Example trans1:= a=100)

or simply

<TransitionCondition>

(Example $a=100$)

For an example (condition_xy), refer to the figure above.

Also refer to

- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)
- [chapter ""Insert step-transition" command" on page 642](#)
- [chapter "SFC element "Action"" on page 247](#)
- [chapter ""Init step" command" on page 642](#)
- [chapter "SFC element properties" on page 252](#)
- [chapter 9.18.4 "Method call" on page 150](#)

SFC element "Action"

Symbol



An action includes one or more statements in one of the valid implementation languages. One action can be assigned to one step.

Actions that you use in SFC steps must be created as POU in the project.

For the variables used in the actions, refer to the declaration section of the "father" POU (in the example sfc1).

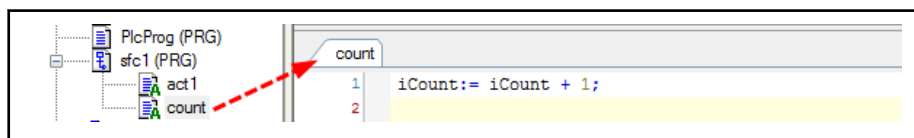


Fig. 17-8:

Exception: In the case of IEC actions, which you add to a step as action association, you can also specify a Boolean variable instead of an action object. The value of these variables is switched between FALSE and TRUE each time the "action" is executed.



Step names have to be unique within the valid range of the "father" function block.

An action described in SFC may not contain a step with the same name as the step to which the action is assigned.

IEC actions

There are "IEC actions" and "step actions":

The IEC actions comply with the IEC1131-3 standard. They are executed according to the qualifier.

IEC actions are executed two times: first when the step is activated and second when the step is deactivated. When assigning several actions to one step, the action list is processed from top to bottom.

Each action box includes the qualifier in the first column and the action name in the second column. Both can be edited directly.

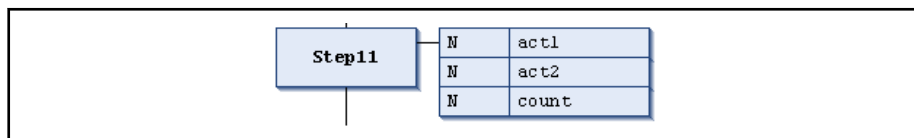


Fig. 17-9:



If the same global Boolean variable is associated as an IEC action in different SFC blocks, then this can result in undesired overwriting effects.

As opposed to step actions, different qualifiers can be used for IEC actions. In addition, each IEC action is provided with a control flag. This directs IndraLogic to execute an action only once at a time, even if the action is called by another step at the same time. This cannot be guaranteed for step actions.

To assign IEC actions to steps, select **SFC ► Insert action association**.

Also refer to

- [chapter ""Insert action association" command" on page 645](#)
- [chapter "Qualifiers for SFC actions" on page 239](#)

Step actions *Use these step actions to extend the IEC standard.*

- **Entry action:**

IndraLogic executes this action after the step has been activated and before the main action is executed.

Reference a new action, or an action created below the SFC object, from a step using the **Entry action** element property (sCount_entry). A new action can also be added to the step using the "Add entry action" command. The entry action is indicated by an "E" in the lower left corner of the step box.

- **Main action:**

IndraLogic executes this action when the step is active and any entry actions have already been processed. However, as opposed to IEC actions (see above), these step actions are not executed a second time when the step is deactivated. Qualifiers may nor be used here as well.

Add an already existing action to a step with the **Main action** (sCount_active) element property. Double-click on the step element to create and add a new action. The main action is indicated by a filled triangle in the right upper corner of the step box.

- **Exit action:**

IndraLogic executes this action once when the step is disabled. Please note, however, that an exit action is not executed in the same cycle, but at the beginning of the next cycle!

Reference a new action, or an action created below the SFC object, from a step using the **Output action** element property (sCount_exit). Add new action to the step using the "Insert exit action" command. It is marked with an "X" in the lower right of the step box.

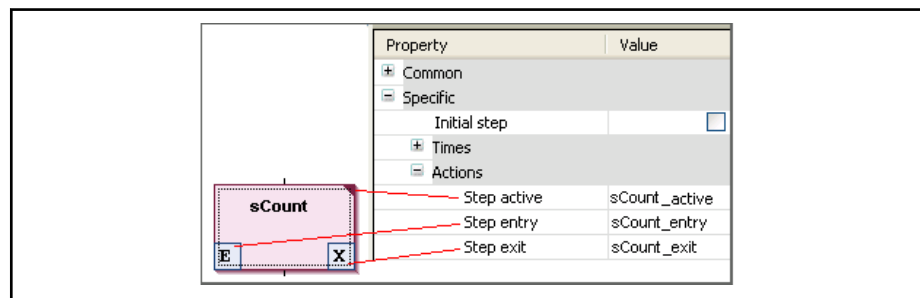


Fig. 17-10:

Difference between IEC action and step action

Also refer to

- [chapter "SFC element properties" on page 252](#)

The main difference between step actions and IEC actions with a qualifier "N" is that an IEC action is always executed two times: when the step is activated and when the step is deactivated. Refer to the following example:

Example:

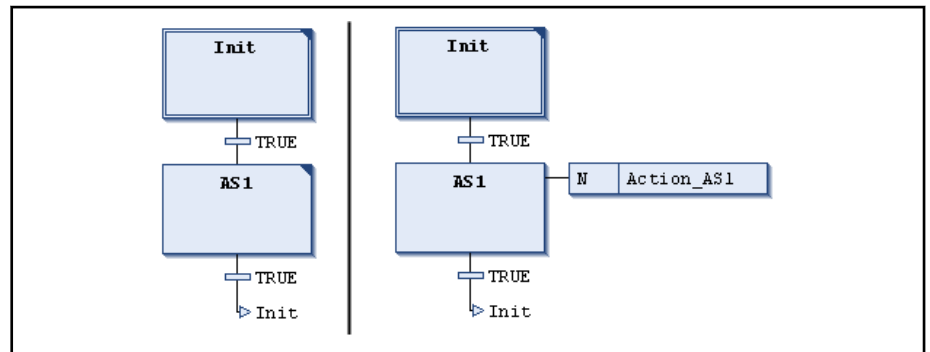


Fig. 17-11:

You have attached the `Action_AS1` action to the `AS1` step as a step action (left) and as an IEC action with qualifier `N` (right).

As two transitions are activated in each case, the time to reach the initial step again is two PLC cycles.

It is assumed that the `iCounter` counter variable was initialized at 0 and then incremented in the `Action_AS1` action.

After the `Init` step has been enabled again, `iCounter` is 1 in the example on the left.

In the example on the right, 2 is provided, as the IEC action is executed a second time due to the deactivation of `AS1`.

Another difference is that the step actions can be virtually "embedded". In this case, they can only be called by the respective step. If this step is copied, IndraLogic automatically creates new action objects and copies the respective implementation code. Define whether a step action is "embedded", either when the first action is inserted into the step, or later in the "Duplicate when copying" element property. Generally, this response can also be preset in the SFC options.

Finally, for IEC actions a Boolean variable can be specified instead of an action object. This is not possible for step actions.

SFC element "Branching"

Symbol

Use branchings to program parallel or alternative sequences in the sequential function chart.

For alternative branchings, IndraLogic processes only one of the branches at a time, depending on the preceding transition condition. Parallel branchings are executed simultaneously.

Also refer to

- [chapter "Processing sequence in SFC" on page 237](#)
- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)

- [chapter ""Insert branch right" command" on page 644](#)

Parallel branching

For parallel branching, the branches have to start and end with steps. Parallel branches can contain other branchings.

The horizontal line in front of and behind the branch is a double line.

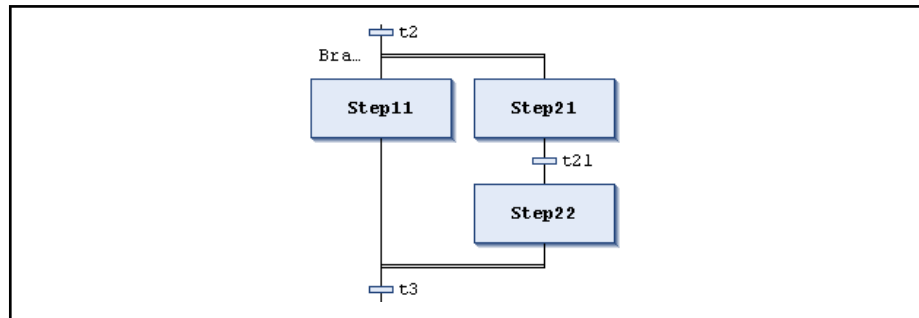


Fig. 17-12:

Processing in online mode:

If the preceding transition (t2 in the example below) is **TRUE**, the first steps in all parallel branches become active (Step11 and Step21).

IndraLogic processes the individual branches at the same time and the subsequent transition is evaluated afterwards (t3).

The "Branch<n>" jump marker is added automatically to the horizontal line that indicates the beginning of a branching. This label can be defined as jump target.

Please note that a parallel branching can be converted into an alternative branching with **Alternative**.

Also refer to

- [chapter ""Alternative" command" on page 643](#)

Alternative branch

The horizontal lines in front of and behind the branch are single lines.

In an alternative branching, the branches have to start and end with transitions. The branches can contain other branchings.

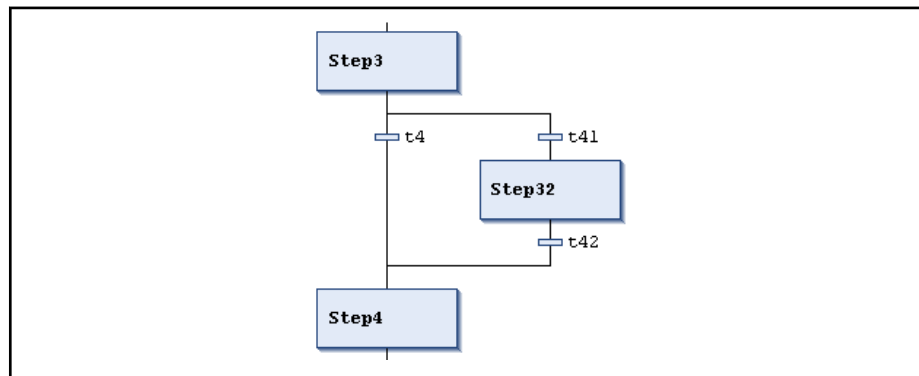


Fig. 17-13:

If the step preceding the branching is active, IndraLogic evaluates each first transition of the alternative branches from left to right. For the first transition returning **TRUE**, the associated branch opens, thus activating the step following the transition.

Please note that an alternative branching can be converted into a parallel branching with **Parallel**.

Also refer to

- [chapter "Parallel" command" on page 643](#)

SFC element "Jump"

Symbol 

Use a jump to define which actions in a step should be executed next as soon as the transition preceding the jump is **TRUE**. Jumps can be necessary, since the execution lines cannot cross each other or lead upwards.

Excluding the required jump at the end of a diagram, jumps can generally be inserted only at the end of a branching.

The jump target is defined by the added text string, which can be edited directly. The jump target can be a step name or the marker of a parallel branching.

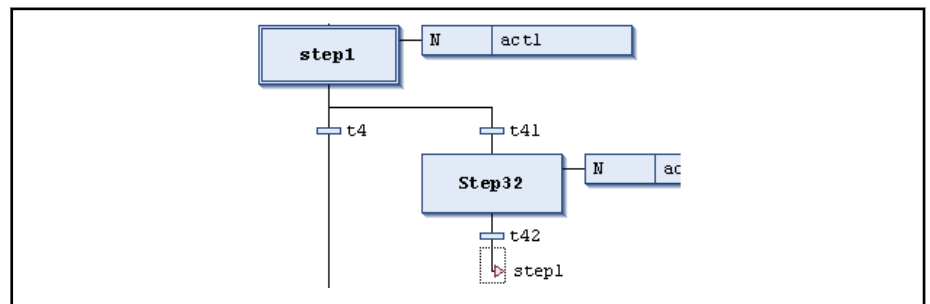


Fig. 17-14:

Also refer to

- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)
- [chapter "Insert jump" command" on page 646](#)

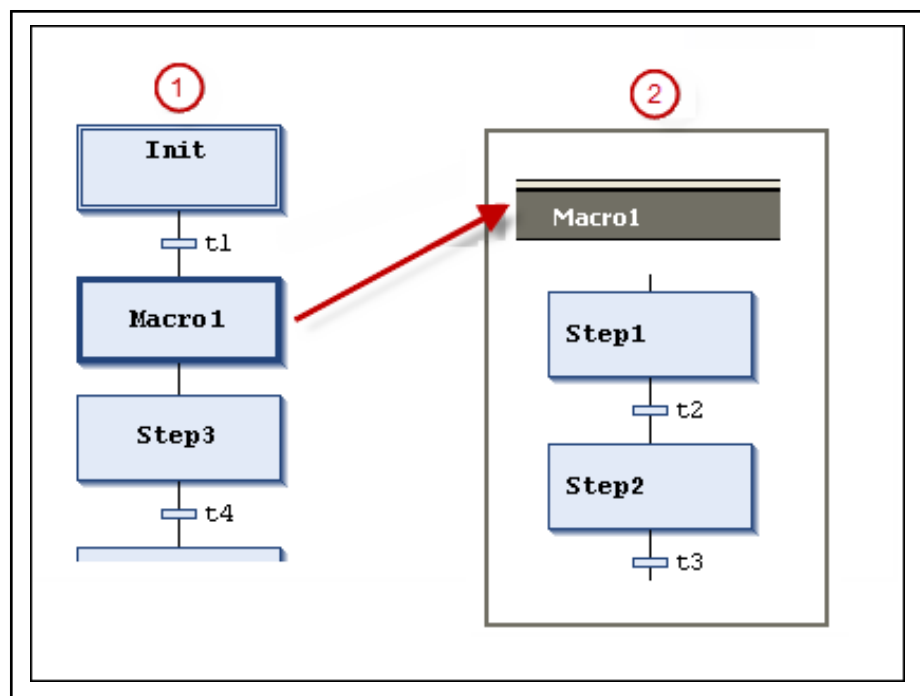
SFC element "Macro"

Symbol 

A macro contains a part of the SFC diagram that is not displayed in detail in the main view of the editor.

The processing flow is not affected by the use of macros. They are only used to hide certain parts of the diagram, e.g. to improve clarity.

To open the macro editor, double-click on the macro box or select **SFC ► Zoom into macro**. It can be programmed like in the main view of the SFC editor. To close the macro editor, click on **SFC ► Zoom out of macro**.



- ① Main view in the SFC editor
 ② View in the macro editor for Macro1

Fig. 17-15:

Macros can also contain macros. The header of the macro always shows the path of the macro currently open in the diagram.

Example:

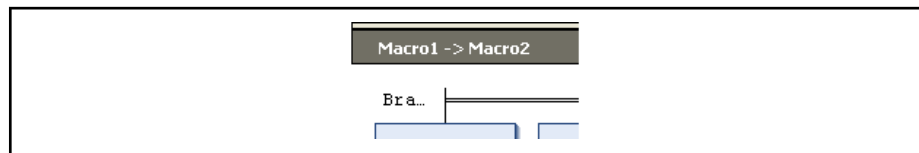


Fig. 17-16:

Also refer to

- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)
- [chapter ""Zoom into macro" command" on page 647](#)
- [chapter ""Zoom out of macro" command" on page 648](#)

SFC element properties

Edit the properties of an SFC element in the **Properties** view.

Open this view with **View ► Element properties**. The selected element determines which properties are displayed.



The properties displayed in the SFC diagram next to the element depend on the settings in the **View** tab of the SFC editor options.

General information

Property	Value description
Name	Element name, by default "<Element><consecutive number>" , e.g. step name "Step0", "Step1", branch name "Branch0" etc.
Comment	Element comment (text), e.g. "Counter reset". To insert line breaks press <Ctrl>+<Enter>.
Symbol	For each SFC element, IndraLogic creates an implicit variable with the same name as the element. The configuration determines whether this flag variable should be exported to the symbol configuration and which access rights should be applied to the control for the symbol. • No access: The symbol is exported to the symbol configuration, but cannot be accessed from the control. • Read: The symbol is exported to the symbol configuration and can be read on the control. • Write: The symbol is exported to the symbol configuration and can be written on the control. • Read/write: Read/write combination. • No selection: No symbol is exported to the symbol configuration.

Specifically

Property	Value description
Initial step	 : This option is only enabled for the defined initial step. By default, this is the first step in an SFC diagram. Note: If this property has been enabled for another step, disable it in the previous step to prevent compilation errors.
Duplicate when copying	This option is available for steps containing a step action (entry action, main action or exit action) and for transitions linked to a transition object. : When copying the step/the transition, a new object is created for the called actions/transitions. It includes a copy of the implementation code of the copied object. : When copying the step or the transition, the link to the called object is retained for the corresponding actions/transitions. No new objects result. Source or copies of the step or the transition call the same action/transition.
Times • Minimum active • Maximum active	Minimum time that the step is active, even if the subsequent transition is TRUE. Maximum time that the step may be active. If this time is exceeded, IndraLogic sets the implicit variable <code>SFCError</code> to TRUE. Times according to IEC syntax (for example <code>t#8s</code>) or the TIME variable; default: <code>t#0s</code>.
Actions • Entry action • Step action • Exit action	• Entry action: IndraLogic executes this action after the step has been enabled. • Step action: IndraLogic executes this action when the step is active and any entry actions have already been processed. • Exit action: IndraLogic executes this action in the subsequent cycle when the step is deactivated. Note: Note the processing sequence.



When using the respective implicit SFC variables and flags, information on the status of a step or an action or about timeouts is provided.

Also refer to

- [chapter ""Options" - "SFC editor" dialog" on page 728](#)
- [chapter "Implicit variables" on page 240](#)

17.1.5 Function block plan/ladder diagram/statement list (FBD/LD/IL)

FBD/LD/IL Editor

The FBD/LD/IL editor is a combined editor of the programming languages FBD, LD and IL.



IL can be enabled if required. Go to **Tools** ▶ **Options** ▶ **IndraLogic 2G** ▶ **FBD, LD and IL** ▶ **IL** ▶ **View** ▶ **Enable IL**

There is a common set of commands and elements and IndraLogic automatically converts the three programming languages into one another internally.

The code in the implementation section is structured in all three languages using networks.

The menu **FBD/LD/IL** contains command when working in the editor.

Using the command, it can be switched between the three editor views at any time in offline and in online mode.

The response of the FBD/LD/IL editor is defined by the settings in **Tools** ▶ **Options** ▶ **IndraLogic 2G**, category **FBD, LD and IL**.



There are some special elements that IndraLogic cannot convert and thus only displays them in the appropriate language.

There are also constructs that are not clearly convertible between IL and FBD and are therefore "normalized", i.e. nullified when converted back to FBD. This applies to the: Negation and explicit/implicit assignments for function block inputs and function block outputs.

A successful conversion between the languages requires syntactically correct code. Otherwise, parts of the implementation might be lost.

Also refer to

- [chapter 9.3.1 "FBD/LD/IL" on page 89 \(Programming\)](#)
- [chapter 18.2.9 "Menu 'FBD/LD/IL'" on page 661 \(Commands\)](#)
- [chapter ""Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter 17.1.2 "Common functionalities in graphical editors" on page 224](#)

FBD editor und LD editor

Inserting and arranging elements

The cursor can also be used to drag elements from the view **Tools** (toolbox) into the implementation section of the editor. Alternatively, use the commands of the context menu or the **FBD/LD/IL** menu.

Settings for the display and interface are defined in the IndraLogic options, category **FBD/LD/IL**.

When dragging an element over a network in the editor, all possible insertion positions are displayed with gray diamond-shaped, triangular or arrow-shaped position marks. As soon as the cursor is located over one of these marks, the mark turns green. If the mouse button is now released, IndraLogic inserts the element at this position.

Example:

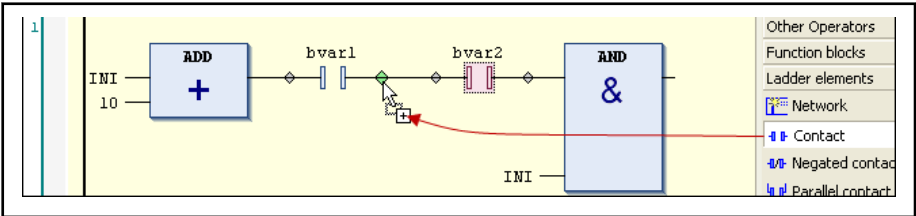


Fig. 17-17:

When dragging a function block or an operator from the toolbox or a network at the left-hand side of the network onto one of the two arrows, the following occurs: IndraLogic automatically creates a new network and inserts the element there.

To replace an element, drag a suitable other element onto its position. Elements that can be replaced by the new element are marked in the editor with text fields, e.g. **Replace**, **Attach input** by IndraLogic.

Use the common commands in the menu **Edit** for cutting, copying, pasting and deleting elements. Copying also works with drag&drop, keeping the <Ctrl> key pressed.

 The operators with EN/ENO functionality can only be inserted into the FBD and IL editors.

Selection of elements

To select a function block or a connecting line in the editor, click on it to set the focus. Multiple selection is possible with the <Ctrl> key pressed. A selected element is shaded red.

Tooltip

If the cursor points to certain elements, for example to a variable or to an input, a tooltip is displayed showing information on this element.

In case of elements underlined with a wavy red line, the tooltip shows the pre-compile error message of the error that occurs with this element.

Navigation in the editor

Using the keys and commands described below, the focus can be set to a different cursor position in the editor. The change between the positions is also a network-comprehensive.	
<->	Change to the neighboring cursor position, along the signal flow, i.e. from left to right and vice versa.
<->	

<↑> <↓>	Change to the next cursor position above or below the current position if this neighboring position belongs to the same logical group. All connections of a function block form a logical group for example. If such a logical group does not exist: Change to the first cursor position in the next higher or lower neighboring element. In case of parallel-connected elements, navigation takes place along the first branch.
<Ctrl> + <Pos 1>	Change to the first network. It will be selected.
<Ctrl> + <End>	Change to the last network. This will be selected.
<Image ↑>	Scroll upwards by one page; the highest network on this page is selected.
<Image ↓>	Scroll downwards by one page; the lowest network on this page is selected.
Go to... command	Change to a certain network.

Tab. 17-3: Navigating in the editor

Open a function block:

If a function block is inserted in the editor, open its implementation by a double-click or with the context menu command **Browse for symbol ► Go to definition**.

Also refer to

- [chapter 9.3.1 "FBD/LD/IL" on page 89 \(FBD, LP\)](#)
- [chapter "Programming in Function Block Diagram \(FBD\)" on page 92](#)
- [chapter "Programming in Ladder Diagram \(LD\)" on page 93](#)
- [chapter ""Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter "FBD/LD/IL editor in online mode" on page 258](#)
- [chapter ""Go to" command" on page 671](#)

IL editor**Inserting and arranging elements:**

Elements can be inserted using the commands of the menu **FBD/LD/IL** from the context menu. A new network can also be dragged from the toolbox into the implementation section of the editor.

Use the common commands in the menu **Edit** for cutting, copying, pasting and deleting elements. Copying also works with drag&drop, keeping the <Ctrl> key pressed.



Please note that operators with EN/ENO functionality can only be inserted into the FBD and IL editors.

Each program line is entered into a table line.

1st line: Network title
Prerequisite: The option is enabled in the IndraLogic options
2nd line: Network comment
Prerequisite: The option is enabled in the IndraLogic options

From the third line:		
Column	Content	Description
1	Operator	Contains the IL operator (LD, ST, CAL, AND, OR, etc.) or a function name. When calling a function block, additionally specify the corresponding parameters here; in the preceding field, enter := or =>.
2	Operand	Contains one operand or the name of a jump label. In case of several operands, enter them in several lines and insert a comma directly behind the individual operands. (See example below)
3	Address	Contains the address of the operand as defined in its declaration. Not editable Activate/deactivate the display via the option Display symbol address . Select Tools ► Options ► IndraLogic 2G and the General tab in the category FBD, LD and IL .
4	Symbol comment	Contains the comment specified for the operand in the declaration if required. Not editable Enable/disable the display via the option Display symbol comment when selecting Tools ► Options ► IndraLogic 2G and the General tab in the category FBD, LD and IL .
5	Operand comment	Comment for the current program line. Enable/disable the display via the option Operand comment when selecting Tools ► Options ► IndraLogic 2G and the General tab in the category FBD, LD and IL .

Tab. 17-4: Structure of a network in the IL editor:

Example:

Example

Spalte:				
①	②	③	④	⑤
CAL	tonInst1(			
	IN:= bVar,			
	PT:= t1,			
	ET=> tOut1)			
LD	tonInst1.Q		is TRUE, PT seconds after IN had a r...	
ST	tonInst2.IN		starts timer with rising edge, reset...	
CAL	tonInst2(			
	PT:= t2,			
	Q=> bReady,	\$QB1		for tonInst2
	ET=> tOut2)			

Fig. 17-18:

Key(s)/ command	Cursor movement
<↑> <↓>	Jumps to the field located above/below.
<Tab>	Jumps one field to the right within the line.
<Shift> >+ <Tab>	Jumps to the left to the preceding field within the line
<Space>	Opens the editing frame for the selected field. Alternatively, click on the field. If applicable, the button for the input assistant dialog is available.
<Ctrl>+<Enter>	Inserts a new line below the current line.
<Del>	Deletes the current line.
<Ctrl> + <Pos1>	Sets the focus to the beginning of the document and marks the first network.
<Ctrl> + <End>	Sets the focus to the end of the document and marks the last network.
<Page down>	Scrolls up by one page and marks the top rectangle.
<Page up>	Scrolls down by one page and marks the top rectangle.

Tab. 17-5: Navigating in the editor

Also refer to

- [chapter 9.3.1 "FBD/LD/IL" on page 89 \(IL\)](#)
- [chapter "Programming in Instruction List \(IL\)" on page 95](#)
- [chapter "Modifiers and operators in IL" on page 259](#)
- [chapter ""Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter "FBD/LD/IL editor in online mode" on page 258](#)

FBD/LD/IL editor in online mode

FBD/LD/IL editor in online mode

In online mode, the current value of each variable is displayed behind the variable in the editor. Writing/forcing and setting breakpoints is possible.

If the variable is presently forced, this is indicated directly in front of the forced value by **F**. If a value has been prepared for writing or forcing, this value is displayed directly behind the current value in square brackets <Value>.

Example:

Forced variable:

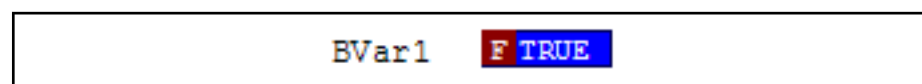


Fig. 17-19:

Prepared value

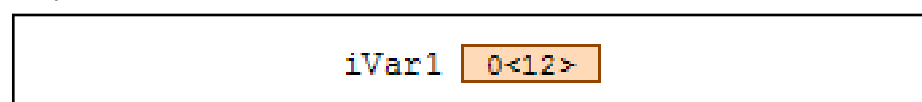


Fig. 17-20:

In the online view of a ladder diagram (LD), the connection lines are colored: Connections with the value `TRUE` are displayed as a thick blue line, connections with the value `FALSE` are displayed as a thick black line. In contrast, connections with an unknown or analog value are displayed normally (thin black line).



Note that values of the connections are calculated from the monitored variables. This is not a real sequence control.

Breakpoints

Possible positions for breakpoints are in principle the positions at which values of variables can change (statements), at which the program branches or at which another function block is called.

Possible breakpoint positions:

- On the complete network: Causes the breakpoint to be set at the first possible position in the network.
- On a function block if the function block includes an assignment. Not possible for operator function blocks, e.g. ADD, DIV.
- On assignments.
- At the end of the function block at the position of the return to the calling function block. In online mode, an empty network is automatically displayed. It is labeled with "RET" in place of a network number.



At present, a breakpoint cannot be set directly to the first function block in the network. However, if a breakpoint is set to the entire network, this breakpoint marking is transferred automatically to the first function block in online mode.



Breakpoints in methods:

IndraLogic automatically sets a breakpoint in all methods that can be called. Therefore, if a method managed by an interface is called, breakpoints are set in all methods that occur in function blocks that implement this interface as well as in all derived function blocks that use the method. If a method is called by a pointer to a function block, IndraLogic sets the breakpoints in the method of the function block and in all derived function blocks that use the method.

Also refer to

- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)

Modifiers and operators in IL

Reference "Programming"

Modifier	Combined with operator	Description
C	JMP, CAL, RET	The statement is only executed if the result of the previous expression is TRUE .
N	JMPC, CALC, RETC	The statement is only executed if the result of the previous expression is FALSE .
N	else	Negation of the operand (not of the accumulator).

Tab. 17-6: Modifiers

Operator	N	Meaning	Example
LD	N	Loads the (negated) value of the operand into the accumulator.	LD iVar
ST	N	Stores the (negated) content of the accumulator in the operand.	ST iErg
S		Sets the operand (type BOOL) to TRUE if the content of the accumulator is TRUE .	S bVar1
R		Sets the operand (type BOOL) to FALSE if the content of the accumulator is TRUE .	R bVar1
AND	N, (	Bit-by-bit AND of the accumulator value and (negated) operand	AND bVar2
OR	N, (	Bit-by-bit OR of the accumulator value and (negated) operand	OR xVar
XOR	N, (	Bit-by-bit exclusive OR of the accumulator value and (negated) operand	XOR N, (bVar1, bVar2)
NOT		Bit-wise negation of the accumulator value	
ADD	(	Addition of the accumulator value and the operand. The result is written to the accumulator.	ADD iVar1
SUB	(	Subtraction of the operand from the accumulator value. The result is written to the accumulator.	SUB iVar2
MUL	(	Multiplication of accumulator value and operand. The result is written to the accumulator.	MUL iVar2
DIV	(	Division of the accumulator value by the operand. The result is written to the accumulator.	DIV 44
GT	(	Checks whether the accumulator value is greater than the operand value. The result (BOOL) is written to the accumulator; >	GT 23
GE	(	Checks whether the accumulator value is greater than or equal to the operand value. The result (BOOL) is written to the accumulator.	GE iVar2
EQ	(	Checks whether the accumulator value is equal to the operand value. The result (BOOL) is written to the accumulator.	EQ iVar2
NE	(	Checks whether the accumulator value is not equal to the operand value. The result (BOOL) is written to the accumulator.	NE iVar1

Operator	N	Meaning	Example
LE	(	Checks whether the accumulator value is smaller than or equal to the operand value. The result (BOOL) is written to the accumulator.	LE 5
LT	(	Checks whether the accumulator value is smaller than the operand value. The result (BOOL) is written to the accumulator.	LT cVar1
JMP	CN	Unconditional (conditional) jump to the specified jump label	JMPN next
CAL	CN	(Conditional) call of a program or a function block (if the accumulator value is TRUE)	CAL prog1
RET		Exiting the function block and returning to the calling function block	RET
RET	C	If the accumulator value is TRUE: Exit the function block and return to the calling function block.	RETC
RET	CN	If the accumulator value is FALSE: Exit the function block and return to the calling function block.	RETCN
)		Evaluation of the postponed operation	

Tab. 17-7: Operators with the possible modifiers

Example:

1	AND	TRUE	load TRUE to accumulator
	ANDN	bVar1	execute AND with negated value of bVar1
	JMPC	m1	if accum. is TRUE, jump to label "m1"
	LDN	bVar2	store negated value of bVar2...
	ST	bRes	... in bRes
2	<u>m1:</u>		
	LD	bVar2	store value of bVar2...
	ST	bRes	... in bRes

Fig. 17-21:

Application	Description	Examples																				
Several operands for 1 operator	<i>Options</i> - Enter the operands into consecutive lines separated by commas in the second column. - Repeat the operator in consecutive rows.	Variant 1: <table><tr><td>LD</td><td>2</td></tr><tr><td>ADD</td><td>3,</td></tr><tr><td></td><td>4,</td></tr><tr><td></td><td>6</td></tr><tr><td>ST</td><td>iVAR</td></tr></table> Variant 2: <table><tr><td>LD</td><td>2</td></tr><tr><td>ADD</td><td>3</td></tr><tr><td>ADD</td><td>4</td></tr><tr><td>ADD</td><td>6</td></tr><tr><td>ST</td><td>iVAR</td></tr></table>	LD	2	ADD	3,		4,		6	ST	iVAR	LD	2	ADD	3	ADD	4	ADD	6	ST	iVAR
LD	2																					
ADD	3,																					
	4,																					
	6																					
ST	iVAR																					
LD	2																					
ADD	3																					
ADD	4																					
ADD	6																					
ST	iVAR																					
Complex operands	For a complex operand, enter the opening parenthesis (into the first column. Enter the closing parenthesis in the first column into a separate line following the operand entries of the following lines.	In each cycle, a string is rotated by one character: <table><tr><td>LD</td><td>stRotate</td></tr><tr><td>RIGHT(</td><td>stRotate</td></tr><tr><td>LEN</td><td></td></tr><tr><td>SUB</td><td>1</td></tr><tr><td>)</td><td></td></tr><tr><td>CONCAT (</td><td>stRotate</td></tr><tr><td>LEFT</td><td>1</td></tr><tr><td>)</td><td></td></tr><tr><td>ST</td><td>stRotate</td></tr></table>	LD	stRotate	RIGHT(	stRotate	LEN		SUB	1	)		CONCAT (	stRotate	LEFT	1	)		ST	stRotate		
LD	stRotate																					
RIGHT(	stRotate																					
LEN																						
SUB	1																					
)																						
CONCAT (	stRotate																					
LEFT	1																					
)																						
ST	stRotate																					
Function block call, program call	Column 1: Operator <code>CAL</code> or <code>CALC</code> Column 2: Name of the function block instance or the program and opening parenthesis (. If no parameters follow, the closing parenthesis) is entered. Following lines: Column 1: Parameter name followed by <code>:=</code> for the input parameter or <code>=></code> for the output parameter Column 2: Parameter value followed by a comma , if further parameters follow. The closing parenthesis) is entered after the last parameter. As a limitation according to the IEC standard, complex expressions cannot be used. Assign such constructs to the function block or the program before the call.	<table><tr><td><code>CAL</code></td><td><code>POUToCall(</code></td></tr><tr><td></td><td><code> iCounter:= 1,</code></td></tr><tr><td></td><td><code> iDecrement:= 1000,</code></td></tr><tr><td></td><td><code> wError=> wResult)</code></td></tr><tr><td><code>LD</code></td><td><code>POUToCall.bError</code></td></tr><tr><td><code>ST</code></td><td><code>bErr</code></td></tr></table>	<code>CAL</code>	<code>POUToCall(</code>		<code> iCounter:= 1,</code>		<code> iDecrement:= 1000,</code>		<code> wError=> wResult)</code>	<code>LD</code>	<code>POUToCall.bError</code>	<code>ST</code>	<code>bErr</code>								
<code>CAL</code>	<code>POUToCall(</code>																					
	<code> iCounter:= 1,</code>																					
	<code> iDecrement:= 1000,</code>																					
	<code> wError=> wResult)</code>																					
<code>LD</code>	<code>POUToCall.bError</code>																					
<code>ST</code>	<code>bErr</code>																					

Application	Description	Examples						
Function call	Line 1: Column 1: LD Column 2: Input variable Line 2: Column 1: Column2: Further input parameters separated by commas. IndraLogic writes the return value into the accumulator. Line 3: Column 1: ST Column 2: Variable to which the return value is written	<table><tr><td>LD</td><td>X7</td></tr><tr><td>GeomAverage</td><td>25</td></tr><tr><td>ST</td><td>Ave</td></tr></table>	LD	X7	GeomAverage	25	ST	Ave
LD	X7							
GeomAverage	25							
ST	Ave							
Action call	Like function block call or program call. The action name is attached to the name of the function block instance or the program.	<table><tr><td>CAL</td><td>POU1.ResetAction</td></tr></table>	CAL	POU1.ResetAction				
CAL	POU1.ResetAction							
Jump	Column 1: Operator JMP or JMPC. Column 2: Name of the jump label of the target network. In case of an unconditional jump, the preceding statement sequence has to end with one of the following commands: ST, STN, S, R, CAL, RET, JMP In case of a conditional jump, the execution of the jump depends on the loaded value.	<table><tr><td>LD</td><td>BVar1</td></tr><tr><td>JMPC</td><td>Label1</td></tr></table>	LD	BVar1	JMPC	Label1		
LD	BVar1							
JMPC	Label1							

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter 9.3.1 "FBD/LD/IL" on page 89 \(IL\)](#)
- [chapter "Programming in Instruction List \(IL\)" on page 95](#)

Elements

FBD/LD/IL element "Network"

Symbol: 

A network is the basic unit of an FBD or LD program. In the FBD/LD/IL editor, the networks are arranged in a list.

Each network is provided with a sequential network number on the left side and can include logical and arithmetic expressions, program/function/function block calls, jumps or return statements.

An IL program consists of at least one network. This network can include all IL statements of the program.

Each network can be provided with a title, comment or label. In the IndraLogic options (category **FBD, LD, and IL**), you can define whether network title, comment, , and separator between individual networks are displayed in the editor.

Click on the first line of the network to enter a network title. Click on the second line of the network to enter a network comment.

Also refer to

- [chapter ""Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter ""Insert network" command" on page 661](#)

FBD/LD/IL element "Function block"

Symbol: 

A function block and its call can represent additional functions, for example IEC function blocks, IEC functions, library function blocks, operators.

A function block can have any number of inputs and outputs.

If the function block also provides an image file, the function block symbol is displayed inside the function block. The prerequisite is that the option **Show function block symbol** is selected in the IndraLogic options, category **FBD, LD and IL**.

When changing the function block interfaces, update the function block parameters with **FBD/LD/IL ► Update parameters** without having to re-insert the function block.

Also refer to

- [chapter "Insert function blockcommand" on page 662](#)
- [chapter "Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter "Update parameters" command" on page 669](#)

FBD/LD/IL element "Assignment"

Symbol: 

The FBD editor shows a newly inserted assignment as a line with three following question marks. The LD editor shows a newly inserted assignment as a coil with three following question marks located above.

After insertion, replace the placeholder ??? by the variable name to which the signal coming from the left is to be assigned if required. The input assistant is available.



In IL an assignment is programmed via the operators **LD** and **ST**.

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "Insert assignment" command" on page 661](#)

FBD/LD/IL element "Function block with EN/ENO"

Symbol: 

The element is only available in the FBD and LD editors.

The function block generally corresponds to the FBD/LD/IL element **Function block**. However, this function block additionally contains an EN input and an ENO output. EN and ENO have the data type **BOOL**.

Function of the EN input and ENO output: If the input EN is **FALSE** when the function block is called, the operations defined in the function block are not executed. Otherwise, if EN is **TRUE**, these operations are executed. The ENO output has the same value as the EN input.

Also refer to

- [chapter "Insert function block with EN/ENO" command" on page 662](#)
- [chapter "FBD/LD/IL element "Function block"" on page 264](#)

FBD/LD/IL element "Input"

Symbol: 

The maximum number of inputs depends on the function block type.

A newly added input is assigned with ??? first. Replace the string ??? by a variable or a constant.

Also refer to

- [chapter ""Insert input" command" on page 663](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)

FBD/LD/IL element "Jump label"

The label is an optional identifier for a network in FBD and LD. It can be specified as target for the jump.

When inserting a jump label into a network, it is added as editable field **Label:** to the network.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Insert label" command" on page 663](#)

FBD/LD/IL element "Jump"

Symbol 

In FBD or LD, a jump is inserted either directly in front of an input, directly after an output or at the end of the network depending on the current cursor position.

Enter a jump label as jump target directly behind the jump element.

In IL, program a jump with the statement `JMP`.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Insert jump" command" on page 663](#)
- [chapter "FBD/LD/IL element "Jump label"" on page 265](#)

FBD/LD/IL element "Return"

This element immediately interrupts the execution of the function block if the input of the `RETURN` element becomes `TRUE`.

In an FBD or LD network, place the "Return" statement parallel to or after the preceding elements.

In IL, the `RET` statement is available for this purpose.

Also refer to

- [chapter ""Insert return" command" on page 663](#)
- [chapter "Modifiers and operators in IL" on page 259](#)

FBD/LD/IL element "Line branching"

Symbol: 

The element is available in the LD and FBD editor and represents an open line branching. A line branching splits the processing line into two subnetworks from the current cursor position onwards. These are executed successively from top to bottom. Each subnetwork can be branched further. As a result, multiple branchings are created within a network.

Each subnetwork is provided with a marker symbol (rectangle) at the branching point. This marker symbol can be selected to execute further commands.



The commands **Copy**, **Cut** and **Paste** are not available for subnetworks.

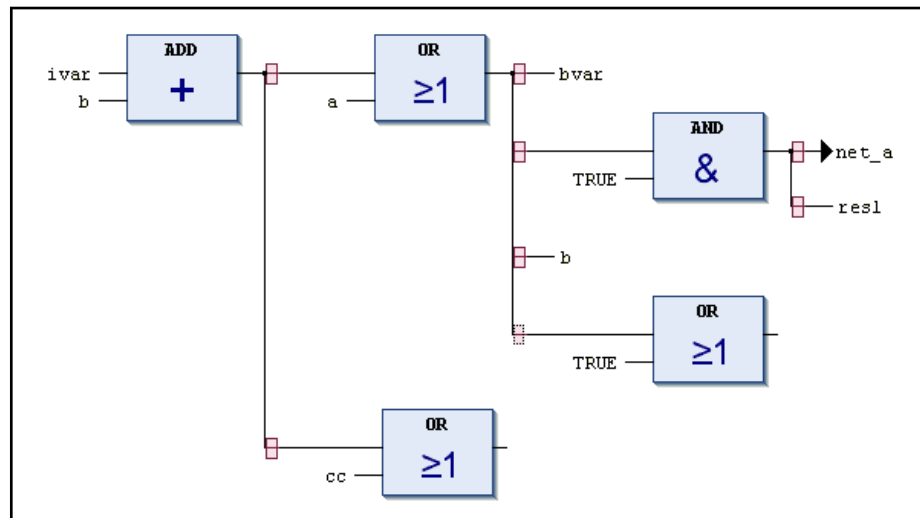


Fig. 17-22:

To delete a subnetwork, delete all elements of the network first and then the marker symbol of the subnetwork.

Also refer to

- [chapter "Insert line branching" command" on page 668](#)
- [chapter "Insert line branching above" command" on page 668](#)
- [chapter "Command "Insert line branching below" command" on page 669](#)

FBD/LD/IL element "Execute"

Symbol:

The element is a function block used to directly enter ST code into the FBD editor and in the LD editor.

Use the cursor to drag the **Execute** element from the **Tools** view into the implementation section of your POU.

Click on **Enter ST code here...** and an input field opens to enter multiple lines of ST code.

- [chapter "FBD/LD/IL Editor" on page 254](#)

LD element "Contact"

Symbol: , in the editor

The element is only available in the LD editor.

A contact passes on the signals **TRUE** (ON) or **FALSE** (OFF) from left to right until the signal finally reaches a coil in the right-hand part of the network. For this purpose, a Boolean variable with the signal is assigned to the contact. Replace the placeholder **???** above the contact with the name of a Boolean variable

Multiple contacts can be arranged in series and in parallel. In case of two parallel contacts, only one needs to obtain **TRUE** in order for ON to be transferred to the right. If contacts are connected in series, all have to obtain **TRUE** in

order for ON to be transferred to the right by the last contact in the series. Thus, electrical parallel connections and series connections can thus be programmed with LD.

A negated contact  forwards the signal `TRUE` if the variable value is `FALSE`. Negate an inserted contact using the command **FBD/LD/IL ► Negation** or insert a negated contact from the **Tools** view.

When placing the cursor on a contact with the left mouse button pressed and with a network selected, the button **Convert to coil** is displayed in the network. When pressing the cursor and moving it onto this button and releasing it, IndraLogic converts the contact into a coil.

Also refer to

- [chapter ""Insert contact" command" on page 664](#)
- [chapter ""Insert negated contact" command" on page 666](#)
- [chapter ""Insert contact \(right\)" command" on page 664](#)
- [chapter ""Insert contact in parallel \(above\)" command" on page 665](#)
- [chapter ""Insert contact in parallel \(below\)" command" on page 665](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)

LD element "Coil"

Symbol: , in the editor 

The element is only available in the LD editor.

A coil assumes the value provided from the left and saves it in the assigned Boolean variable.

Its input can be `TRUE` (ON) or `FALSE` (OFF).

Several coils in one network can only be arranged in parallel.

In a negated coil , the negated value of the incoming signal is stored in the Boolean variable assigned to the coil.

Set coil, reset coil

Symbol: , , in the editor: , 

Set coil: If the value `TRUE` reached a set coil, the coil retains the value `TRUE`. As long as the application is running, the value can no longer be overwritten at this location.

Reset coil: If `TRUE` reached a reset coil, the coil retains the value `FALSE`. As long as the application is running, the value can no longer be overwritten at this location.

Define an inserted coil as a set or reset coil using the command **FBD/LD/IL ► Set/reset** or insert it as an element **Set coil** and **Reset coil** from the **Tools** view.

Also refer to

- [chapter ""Insert coil" command" on page 664](#)
- [chapter ""Insert reset coil" command" on page 664](#)
- [chapter ""Negation" command" on page 667](#)
- [chapter ""Set/reset" command" on page 668](#)

LD element "Line branching start/end"

Symbol: 

The element is used for the closed line branching.

Also refer to

- [chapter "Closed line branching" on page 268](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Set line branching start point" command" on page 669](#)
- [chapter ""Set line branching end point" command" on page 669](#)

Closed line branching

A closed line branching is only available in LD and contains a starting point and an end point. It is used to implement a parallel analysis of logical elements.

Inserting a closed line branching

- Command **FBD/LD/IL ▶ Insert contact in parallel (below)**
- Command **FBD/LD/IL ▶ Insert contact in parallel (above)**
- Command **FBD/LD/IL ▶ Set branch start/end point**

Closed line branching at a contact

When highlighting one or multiple contacts and executing the command "Insert contact in parallel", a parallel branching is inserted with a single vertical line. For this type of branching, the signal flow passes through both branches. It is an OR construct of the two branches.

Closed line branching at a function block - OR evaluation, short-circuit evaluation

New: When highlighting a function block and executing the command "Insert contact in parallel", a parallel branching is inserted with a double vertical line. This indicates that a short-circuit evaluation ("Short Circuit Evaluation" - "SCE") is implemented. The short circuit evaluation enables the execution of a function block with a Boolean output to be bypassed if a certain condition is **TRUE**. The condition can be displayed in the LD editor by a branch connected in parallel to the function block branch. The short circuit condition is defined by one or more contacts in this branch, which are connected in parallel or in series.

Functional principle:

First of all, the branches without function block are observed. If IndraLogic determines the value **TRUE** for one of these branches, the function block in the parallel branch is not called. In this case, the value applied to the input of the function block is transmitted directly to the output. If IndraLogic determines **FALSE** for the short-circuit evaluation condition, the function block is called and the Boolean result of its processing is transferred. If all branches contain function blocks, they are analyzed from top to bottom and their outputs are logically ORed. If there are no branches with function blocks, normal OR operations are performed.

Example

The function block instance **x1** (TON) has a Boolean input and a Boolean output. The execution of **x1** is skipped if **TRUE** is determined for the condition in the parallel line branching. The condition value results from OR and AND operations that connect the contacts **cond1**, **cond2** and **cond3**.

x1 is executed if the condition value from the connection of the contacts **cond1**, **cond2** and **cond3** is **FALSE**.

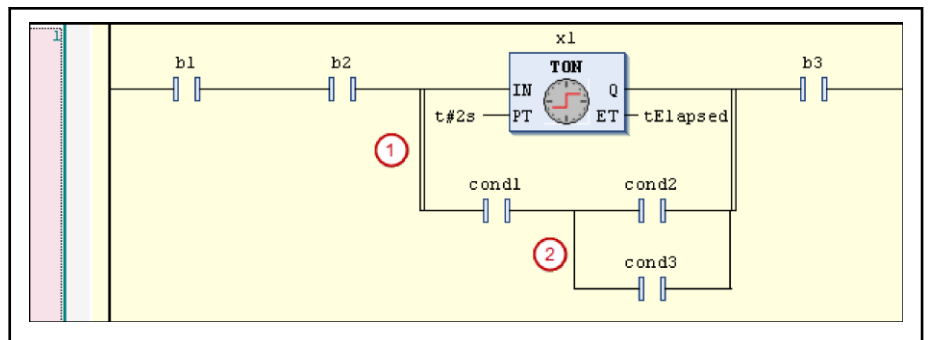


Fig. 17-23:

(1) uses double-drawn vertical connections to show a construct subject to a short circuit evaluation.

(2) uses single-drawn vertical connections to show an OR construct.

The preceding LD example is shown as ST code. P_IN and P_OUT are the Boolean values at the input (split point) and output (reunification point) of the parallel line branching.

```
P_IN := b1 AND b2;

IF ((P_IN AND cond1) AND (cond2 OR cond3)) THEN
  P_OUT := P_IN;
ELSE
  x1(IN := P_IN, PT := t#2s);
  tElapsed := x1.ET;
  P_OUT := x1.Q;
END_IF
bRes := P_OUT AND b3;
```

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Set line branching start point" command" on page 669](#)
- [chapter ""Set line branching end point" command" on page 669](#)
- [chapter ""Insert contact in parallel \(above\)" command" on page 665](#)
- [chapter ""Insert contact in parallel \(below\)" command" on page 665](#)

17.1.6 Continuous Function Chart (CFC) and page-oriented CFC

From an external point of view, a function block diagram consists of inputs and outputs, with data being processed between them. From an internal point of view, a function block diagram consists of POU's and their connections which represent data (signals) and act as assignment operators in ST. The overall behavior is composed of the behavior of the inserted POU's which call other POU's or library POU's.

Code in the "Continuous Function Chart (CFC)" implementation language mainly illustrates the data flow through the system. Therefore, a continuous function chart is also referred to as a "signal flow chart".

In the page-oriented CFC editor, you can wire POU's to each other and create well-structured function block diagrams distributed over multiple pages. The page-oriented editor behaves like the CFC editor, but provides the following functionality:

- Creating pages
- Setting the page size
- Copying and inserting pages in the page navigator

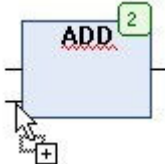
- Copying the implementation of a POU in the CFC implementation language and inserting into a page
- Well-structured and space-saving arranging of inputs, outputs, and connection marks in the border areas
- Connection over pages with connection marks

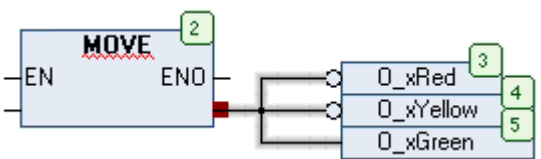
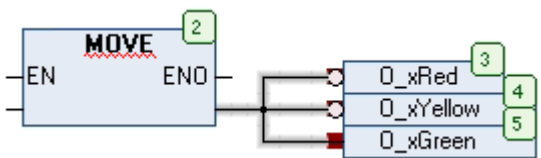
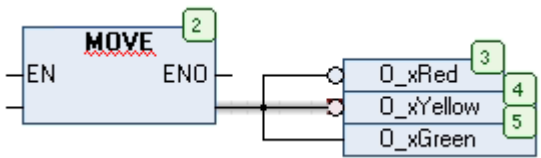
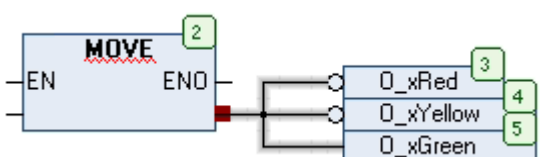
CFC editor

Configuring editors

Configure the appearance, behavior and printing for the entire project in the IndraWorks options under **Tools ► Options ► IndraLogic 2G**, category "CFC editor". For example, on the "View" tab, you can configure the color of the connecting lines depending on the data type.

Editing

Cursor symbol 	Prerequisite: The "pointer" is selected in the "ToolBox" view (to open the view, double-click into the workspace of the CFC editor). The symbol indicates that you can edit in the editor. Select elements or connections to move them or to execute commands.
Cursor symbol 	Prerequisite: An element is selected in the "ToolBox" view (to open the view, double-click into the workspace of the CFC editor). Clicking in the editor inserts the selected element. Also use drag&drop to drag an element into the editor.
Dragging a function block instance from the declaration and dropping it into the editor	Prerequisite: A line is selected in the declaration of the CFC. The instance is inserted as a POU with name, type, and all pins.
Dragging a variable from the declaration and dropping it into the editor at a function block pin	The variable is inserted as an input or output with a connection to the POU pin in focus. Tip: The cursor indicates when your focused location is valid for a variable: 
[Ctrl] + click in the programming area	Prerequisite: An element is selected in the "ToolBox" view (to open the view, double-click into the workspace of the CFC editor). As long as you hold down the [Ctrl] key, a selected element is created each time you click in the programming area.

[Ctrl] + [Right arrow]	Prerequisite: In the CFC program, exactly one output pin is selected for an element.  The selection is moved so that the input pin at the end of the connecting line is selected. In the case of multiple pins, all are selected. 
[Ctrl] + [Left arrow]	Prerequisite: In the CFC program, exactly one input pin is selected for an element.  The selection is moved so that the output pin at the beginning of the connecting line is selected. In the case of multiple pins, all are selected. 

Tab. 17-8: Functions in the CFC editor

Also refer to [chapter 17.1.2 "Common functionalities in graphical editors"](#) on page 224

Connecting

You can insert connecting lines between element connections. Connecting lines are inserted by means of auto-routing so that connecting lines are automatically optimal and as short as possible. The connecting lines are checked for collisions.

Dragging a pin and dropping it to another	A connecting line is inserted between the two element pins.
Command "Connect Selected Pins"	Prerequisite: Multiple pins are selected and the pins to be connected are labeled in red.
Move an inserted element so that it touches the pin of another element.	Prerequisite: The "Enable AutoConnect" option is selected. The touching pins are connected automatically.

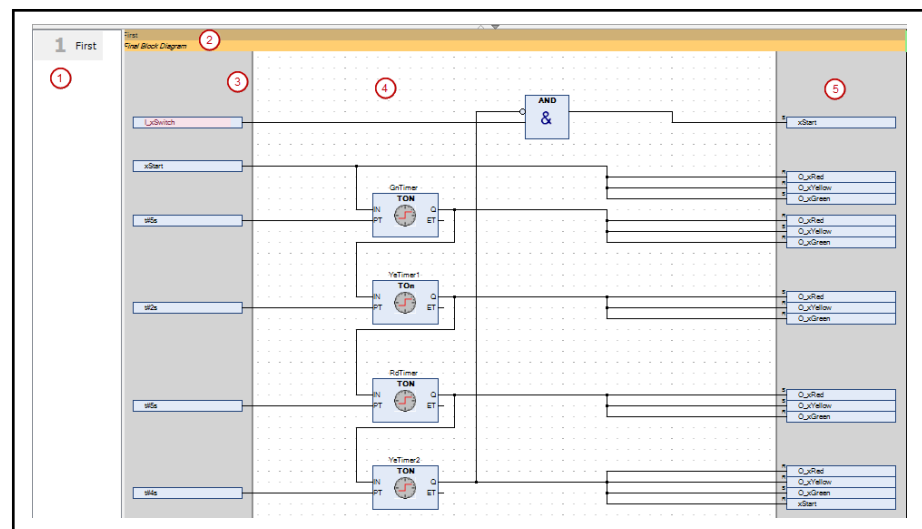
Tab. 17-9: Functions in the CFC editor

Commands when editing

Related documentation:

- chapter 17.1.2 "Common functionalities in graphical editors" on page 224
- chapter "Programming in the CFC editor" on page 97
- chapter 18.2.8 "Menu 'CFC'" on page 648

 POUs generated in the "Continuous Function Chart (CFC) - page-oriented" cannot be converted into "Continuous Function Chart (CFC)" POUs or back.



- ① Page navigator
- ② Page header with name and description
- ③ Left border area reserved for inputs and sink connection marks
- ④ Program area
- ⑤ Right border area reserved for outputs and source connection marks

Fig. 17-24:

Editing	A newly created and page-oriented object includes a page to create a program by default.

Use drag&drop to add additional pages to the page navigation. Double-click into the CFC editor to open the "ToolBox" view and drag the "Page" element from the view and drop it into the page navigation.

Existing pages can be multiplied. Select the page in the page navigation, then the context menu item **Copy** and then **Paste**.

Connections over multiple pages are established by means of the "Connection Mark - Source" and "Connection Mark - Sink" elements. When you drag a connecting line from an input pin or an output pin to the border area, a new connection mark is created automatically. The advantage is that the "List components" input assistance provides all previously defined connection mark sources.

If you have selected an element in the editor, then you can use the arrow keys to move the selection from one element to the next to navigate through the circuit. If you then select a connection mark and press another arrow key, even the corresponding connection mark of the next/previous page will be selected.

Transfer the networks from a CFC programming function block to the programming area of a page-oriented CFC using the context menu items "Copy" and "Paste" (from the clipboard). Also use drag&drop.

Execution order

The execution order is determined automatically according to the order of the pages as they are sorted in the page navigator of the editor. Within a page, a page-oriented CFC object behaves like a CFC object.

Also refer to:

- [chapter 17.1.2 "Common functionalities in graphical editors" on page 224](#)
- [chapter "Programming in the CFC editor" on page 97](#)
- [chapter 18.2.8 "Menu 'CFC'" on page 648](#)

CFC editor in online mode

Monitoring

In online mode, you can monitor and change variable values of the control. In addition, debugging features are provided such as breakpoints and stepping.

In online mode, the Inline monitoring function allows to display the current variable values in the declaration part and in the implementation part of the CFC editor. Inline monitoring of a function block is possible only when an instance of the function block is open. No values are displayed in the basic implementation view.

Monitoring a Boolean variable

The connections between Boolean variables are displayed in color according to their actual value: `TRUE` in blue and `FALSE` in black. The element pins are decorated with the actual value.

Example:

An application contains a CFC POU. An internal Boolean variable is switched there. With each cycle, the variable `xToggle` switches its state from `TRUE` to `FALSE`.

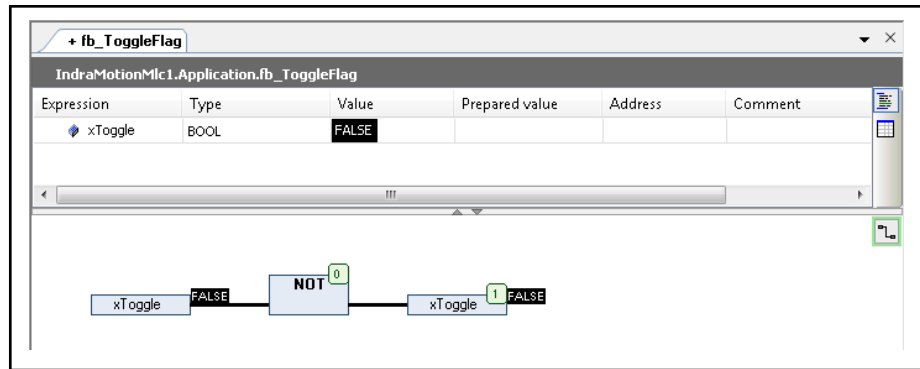


Fig. 17-25:

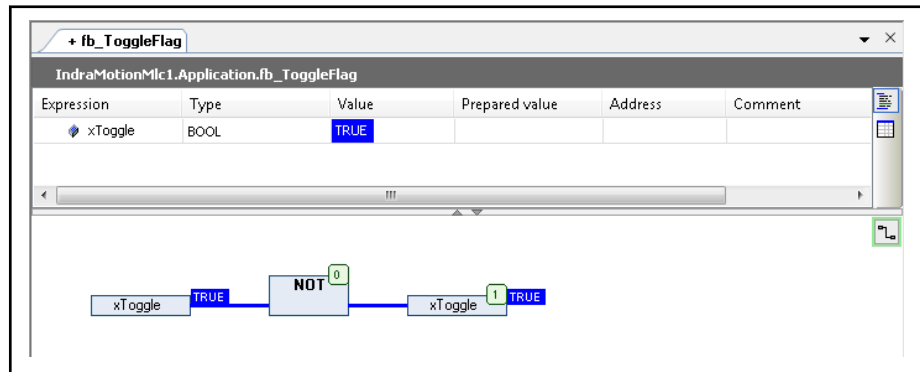


Fig. 17-26:

Monitoring a scalar variable

In the case of scalar variables, the element pins are decorated with the actual variable values.

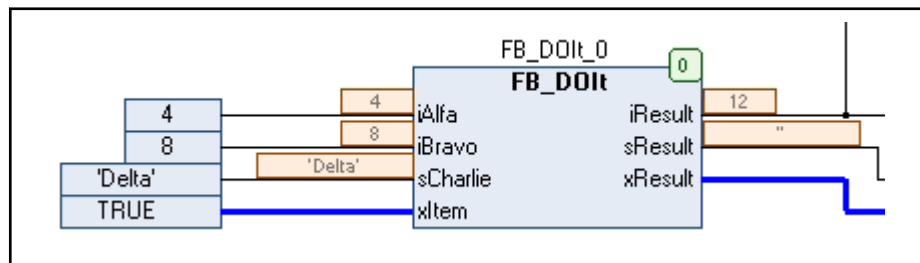


Fig. 17-27:

Forcing and writing variablesCFC editor

In online mode, a value can be prepared in the declaration editor to force or write a monitored variable.

Values can also be prepared in the implementation part if the IndraLogic Option "Prepare values in the implementation part" is selected.

For the option, go to the IndraWorks menu **Tools** ► **Options** ► **IndraLogic 2G** in the category **CFC editor**.

To prepare values, open the **Prepare value** dialog, double-click on an element or on the monitoring box next to the respective element or directly on the element.

No dialog is displayed for Boolean variables. Click into the "Prepared value" field to set the value. Each click toggles between **TRUE** and **FALSE**.

Prepared values are displayed in angle brackets. After executing writing or forcing, a red "F" is shown in the monitoring box.

Changing constant input parameters of function block instances

You can write input parameters of function block instances of type `VAR_INPUT CONSTANT` in online mode and modify the parameters in this way. To save these parameters after logout, click on "Save prepared parameters to project".

Prerequisite: A CFC editor is active. An instantiated function block has `VAR_INPUT CONSTANT` variables in its declaration.

1. In the editor, open the POU by calling the function block instance.
→ The declaration of `FB_DoIt` has been supplemented by the constant `MAXIMUM`.

Program:

```
FUNCTION_BLOCK FB_DoIt
VAR_INPUT
  iAlfa : INT;
  iBravo : INT;
  sCharlie : STRING := 'Charlie';
  xItem : BOOL;
  iDelta : INT;
END_VAR
VAR_INPUT CONSTANT
  MAXIMUM : INT := 12;
END_VAR
VAR_OUTPUT
  iResult : INT;
  sResult : STRING;
  xResult : BOOL;
```

The graphical representation of the function block instances includes the "Parameters" button:

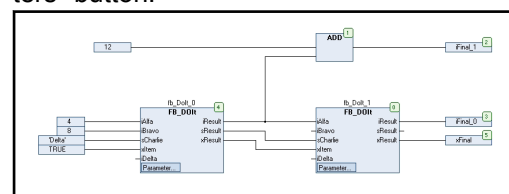


Fig. 17-28:

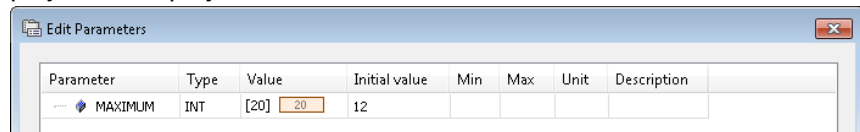
2. Log into the control
3. Click the "Parameters" button of the function block instance.
→ The "Edit parameters" dialog opens.
4. Click the "Value" column in an inline monitoring field of a parameter.
→ The "Prepare value" dialog opens.
5. Enter for example 20 into the "Prepare a new value for the next write or force operation" field.
6. Click "OK" to confirm the entry.

The prepared value is shown in angle brackets next to the current value (e.g. <20>)

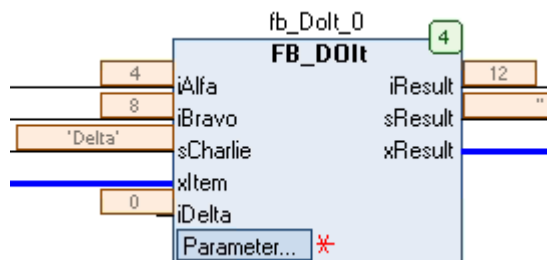
Edit Parameters							
Parameter	Type	Value	Initial value	Min	Max	Unit	Description
MAXIMUM	INT	12 <20>	12				

7. Write the prepared value via **Debug ► Write value**.

→ The prepared value is written. The parameter is changed and displayed in the project in brackets after the value.



The difference between both values is shown by a red cross next to the parameter field of the function block instance



8. Click on "Edit parameters" to close the dialog and log off.

9. Click on **CFC ► Save prepared parameters to project**.

The changed parameter value is applied to the project. The asterisk next to the parameter field disappears.

Breakpoint positions

Possible position of a breakpoint:

- Element "Output": Variables are described.
- Element "Box": POU's are called.
- Element "RETURN": The program flow is branched.
- Element "Selector": Structure elements are described.

Click "Debug" → "Toggle breakpoint" to set a new breakpoint or delete an existing breakpoint. A red circle in the block diagram represents an active breakpoint

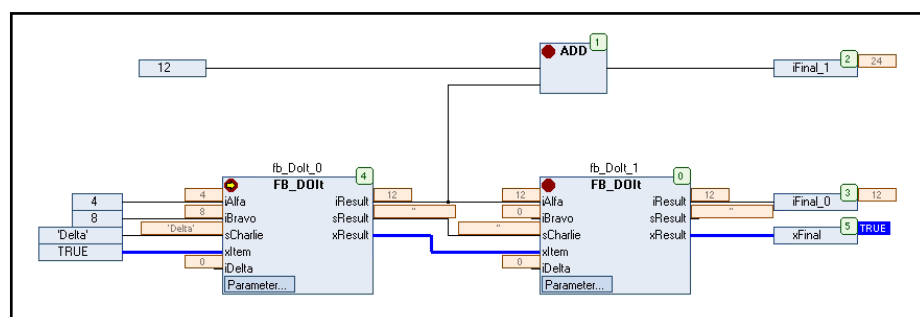


Fig. 17-29:



A breakpoint is automatically set in all methods that can be called.

Thus, the following applies: If a method is called that is defined via an interface, breakpoints are set in all methods of function blocks that implement this interface. This also applies to all derived function blocks defining the method.

Stepping into a POU You can process a POU in steps in debug mode. A called POU is supplemented internally by a `RETURN` at the beginning before the element with the number 0 and at the end after the last element. When stepping, these are started automatically.

Commands in online mode See:

- [chapter "'Force FB input' command" on page 658](#)
- [chapter "Command 'Prepare function block element for forcing'" on page 658](#)
- [chapter "'Edit parameters' command" on page 653](#)
- [chapter "'Save prepared parameters to project' command" on page 654](#)

Also refer to:

- [chapter 13.1 "Monitoring values" on page 191](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 12.3 "Stepping a program" on page 181](#)

Elements

CFC element "Page"

Symbol: 

The element inserts a new page into the editor. It is only available in the page-oriented CFC editor. The page number is specified automatically with regard to its position. The name and the description of the page can be entered into the orange-colored header. Adjust the page size with **Edit page size**.

Also refer to

- [chapter "'Edit page size' command" on page 648](#)

CFC element "Control point"

Symbol: 

Use this control point to fix points of a connection before adjusting the line management. Drag the element to the desired position on the connecting line. Connecting lines with control points are not automatically routed anymore.

Also refer to

- [chapter "Programming in the CFC editor" on page 97](#)
- [chapter "'Create control point' command" on page 657](#)
- [chapter "'Remove control point' command" on page 657](#)

CFC element "Input"

Symbol: 

IndraLogic inserts an input element with the Text "???" by default. Click on this field to directly edit it and to enter a constant value or a variable name. Click on  to also open the input assistant to select a variable.

CFC element "Output"

Symbol: 

IndraLogic inserts an output element with the Text "???" by default. Click on this field to directly edit it and to enter a constant value or a variable name.

Click on  to also open the input assistant to select a variable.

CFC element "Function block"

Symbol: 

Use the element to insert an operator, a function, a function block or a program. IndraLogic inserts the element with the name ??? by default. Click on this field to directly edit it and to enter a function block name. Click on  to also open the input assistant to select a function block.

For a function block, IndraLogic additionally displays an input field above the function block symbol (???). Replace this name with the name of the function block instance. When instantiating a function block with constant input parameters, the function block element in the lower left corner displays the field "Parameters...". Click on this field to edit the parameters.

To replace an existing function block, only replace the currently inserted identifier with the new name. Observe that IndraLogic adjusts the number of input pins and output pins according to the definition of the POU's. Thus, it is possible that existing assignments are deleted.

Also refer to

- [chapter ""Edit parameters" command" on page 653](#)

CFC element "Jump"

Symbol: 

Use the element to specify a position at which the program execution is continued. This target position has to be defined by a mark. Therefore, enter the name of the mark into the ??? input field. If the corresponding label has already been inserted, they can also be selected via the input assistant ().

Also refer to

- [chapter "CFC element "Label"" on page 278](#)

CFC element "Label"

Symbol: 

A marker defines a position to which the program execution jumps using a jump element.

In online mode, IndraLogic automatically inserts a RETURN label at the end of a CFC function block.

Also refer to

- [chapter "CFC element "Jumpon page 278](#)

CFC element "Return"

Symbol: 

Use the element to exit the function block.

Note that in online mode in the CFC editor, a "Return" element is automatically placed in front of the first line and after the last element. In step-by-step processing, IndraLogic automatically jumps to the "Return" element at the end before the function block is exited.

CFC element "Composer"

Symbol: 

The composer element is used to handle structural components. The individual components of a structure are provided as input. To do this, name the composer element as the respective structure (replace the ???).

The composer element is the counterpart of the selector element.

Also refer to

- [chapter "CFC element "Selector"" on page 279](#)

CFC element "Selector"

Symbol: 

The selector element is used to handle structural components. The individual components of a structure are provided as output. To do this, name the selector element as the respective structure (replace the ???).

The selector element is the counterpart of the composer element.

Also refer to

- [chapter "CFC element "Composer"" on page 279](#)

CFC element "Comment"

Symbol: 

This element is used to enter a comment into the CFC editor. Replace the placeholder text in the element with the comment text. A line break can be added with the shortcut <Ctrl> + <Enter>.

CFC element "Connection mark source/target"

Symbol:  , 

Between elements, connection marks can be used instead of a connecting line. This is useful to clearly present complex diagrams.

For a valid connection, connect an element **Connection mark - Source** to the output of an element and an element **Connection mark - Target** to the input of another element. Both marks have to have the same name. Upper and lower case letters not considered.

Notes on the name specification

- The default name for a connection mark is C-<no>. <no> is a consecutive number starting with 1.
- The default name can be renamed. Ensure that source marker and target marker have the same name.
- Changing the name of the source marker automatically changes the target name.
- Changing the name of the target label does not change the source name.



Note the **Connection mark** command to automatically convert an existing connection.

Also refer to

- [chapter ""Connection mark" command" on page 657](#)
- [chapter "CFC editor" on page 270](#)

CFC element "Function block input"

Symbol:

Depending on the function block type, more inputs can be added to an inserted function block element. Thus, select the function block and drag the function block input element to the function block body.



Press <Ctrl> to drag an input connection or output connection to another position of the function block.

Also refer to

- [chapter "CFC element "Function block output"" on page 280](#)

CFC element "Function block output"

Symbol:

Depending on the function block type, more outputs can be added to an inserted function block element. Thus, select the function block element and drag the function block output element to the function block body.



Press <Ctrl> to drag an input connection or output connection to another position of the function block.

Also refer to

- [chapter "CFC element "Function block input"" on page 280](#)

17.2 Variables

The scope of a variable defines how and where you can use a variable. You define the scope in the variable declaration.

17.2.1 Local Variables - VAR

Local variables are declared between the keywords `VAR` and `END_VAR` in the declaration part of programming objects.

There is only read-only access to local variables using the instance path.

Extend local variables with an attribute keyword.

Example

```
VAR
    iVar1 : INT;
END_VAR
```

Also refer to [chapter 9.16 "Data persistence" on page 137](#)

17.2.2 Input variables - VAR_INPUT

Input variables are used at the inputs of function blocks.

VAR_INPUT variables are declared between the keywords VAR_INPUT and END_VAR in the declaration section of programming objects.

Extend input variables with an attribute keyword.

Example

```
VAR_INPUT
  iIn1:INT; (* 1st input variable *)
END_VAR
```

Also refer to [chapter 9.16 "Data persistence" on page 137](#)

17.2.3 Output variables - VAR_OUTPUT

Output variables are used at the outputs of function blocks.

VAR_OUTPUT variables are declared between the keywords VAR_OUTPUT and END_VAR in the declaration section of programming objects. IndraLogic returns these variable values to the calling function block. The values can be queried and used again.

Extend output variables with an attribute keyword.

Example:

```
VAR_OUTPUT
  iOut1:INT; (*1st output variable *)
END_VAR
```

Also refer to [chapter 9.16 "Data persistence" on page 137](#)

Output variables in functions and methods

According to the IEC 61131-3 standard, functions and methods have additional outputs. Assign these additional outputs when calling the function as shown below.

Example:

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

17.2.4 Input and output variable - VAR_IN_OUT

A VAR_IN_OUT variable is an input/output variable, which is part of a POU interface and serves as a formal pass-by-reference parameter.

Syntax declaration

Program:

```
<keyword> <POU name>
VAR_IN_OUT
  <variable name> : <data type> ( := <initialization value> )? ;
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

You can declare an input/output variable in the VAR_IN_OUT declaration section in the POUs PRG, FUNCTION_BLOCK, METHOD, or FUNCTION. As an option, a constant of the declared data type can be assigned as an initialization value. The VAR_IN_OUT variable can be read and written.

Usage

- **Call:** When a POU is called, the formal VAR_IN_OUT variable receives the actual variable ("pass-by-reference variable") as argument. At run-time, no copies are generated when parameters are passed. Instead, the formal variable receives a reference to the actual variable passed remotely. The referential variables contain a memory address internally as a value to the actual value (pass as pointer, call-by reference). It is not possible to specify a constant (literal) or a bit variable directly as an argument.

- **Read/write access within the POU:** If the variable is written to within the POU, then this affects the passed variable. When the POU is exited, the performed changes are retained. This means that a POU uses its VAR_IN_OUT variables just like the calling POU uses its variables. Read access is always permitted.
- **Read/write access remotely:** VAR_IN_OUT variables do not have direct, remote read or written access via <function block instance name>.<variable name>. This works only for VAR_INPUT and VAR_OUTPUT variables.
- **Passing string variables:** If a string variable is passed as an argument, the actual variable and the formal variable should be of the same length. Otherwise, the passed string can be manipulated accidentally. This problem does not occur in the case of VAR_OUTPUT CONSTANT parameters.
- **Passing bit variables:** A bit variable cannot be passed directly to a VAR_IN_OUT variable because it needs an intermediate variable.



When a string is passed as a variable or a constant to a formal VAR_IN_OUT CONSTANT variable, the string is automatically passed completely. You do not have to check the string length.

Example:

Passing arrays

```

TYPE DUT_A :
STRUCT
  xA: BOOL;
  iB: INT;
END_STRUCT
END_TYPE

FUNCTION_BLOCK FB_SetArray
VAR_IN_OUT
  aData_A : ARRAY[0..1] OF DUT_A; // Formal variable
END_VAR
aData_A[0].xA := TRUE;
aData_A[0].iB := 100;

PROGRAM PLC_PRG
VAR
  fbSetA : FB_SetArray;
  aSpecialData : ARRAY[0..1] OF DUT_A; // Actual variable
END_VAR
fbSetA(aData_A := aSpecialData);

```

Expression	Type	Value	Prepared value	Address	Comment
fbSetA	FB_SetArray				
aSpecialData	ARRAY [0..1] OF DUT_A				Actual variable
aSpecialData[0]	DUT_A				
xA	BOOL	TRUE			
iB	INT	100			
aSpecialData[1]	DUT_A				
xA	BOOL	FALSE			
iB	INT	0			

Fig. 17-30:

Passing strings

```
{attribute 'qualified_only'}
```

```

VAR_GLOBAL
  g_sDEV_STATUS : STRING(25) := 'Device_A';
END_VAR

FUNCTION_BLOCK FB_SetStatus
  VAR_IN_OUT
    sDeviceStatus : STRING(25); // Formal parameter
  END_VAR
  sDeviceStatus := CONCAT(sDeviceStatus, ' Activ');

PROGRAM PLC_PRG
  VAR
    fbDoB : FB_SetStatus;
  END_VAR
  fbDoB(sDeviceStatus := GVL.g_sDEV_STATUS); //Call with actual parameter

```

The variable `sDeviceStatus` is part of the POU interface of `FB_B`. When calling `fbDoB`, first a device name is assigned to the string and then the string is manipulated.

Expression	Type	Value	Prepared value	Address	Comment
fbDoB	FB_SetStatus				Instance
sDeviceStatus	REFERENCE TO STRING(25)	'Device_A Activ Activ Acti'			Formal parameter

Fig. 17-31:

Passing bit variables

```

VAR_GLOBAL
  xBit0 AT %MX0.1 : BOOL;
  xTemp : BOOL;
END_VAR

FUNCTION_BLOCK FB_DoSomething
  VAR_INPUT
    xIn : BOOL;
  END_VAR
  VAR_IN_OUT
    xInOut : BOOL;
  END_VAR
  IF xIn THEN
    xInOut := TRUE;
  END_IF

PROGRAM PLC_PRG
  VAR
    xIn : BOOL;
    DoSomething_1 : FB_DoSomething;
    DoSomething_2 : FB_DoSomething;
  END_VAR

  // The following line of code causes a compiler error:
  // C0201: Typ 'BIT' is not equal to type 'BOOL' of VAR_IN_OUT 'xInOut'
  DoSomething_1(xIn := xIn, xInOut := xBit0);

  // Workaround
  xTemp := xBit0;
  DoSomething_2(xIn := xIn, xInOut := xTemp);
  xBit0 := xTemp;

```

The program calls the function block instances `DoSomething_1` and `DoSomething_2`. As a result of the direct assignment of the bit variable `xBit0` to the `VAR_IN_OUT` input, a compiler error is generated when the `DoSomething_1` instance is called. In contrast, calling the `DoSomething_2` instance with the assignment of an intermediate variable is correct code.

Transfer variable VAR_IN_OUT CONSTANT

A `VAR_IN_OUT` CONSTANT variable serves as a constant pass-by-reference parameter, to which a variable and a constant can be passed. The parameter can be read, but not written.

Syntax declaration

```

<keyword> <POU name>
VAR_IN_OUT CONSTANT
  <variable name> : <data type>; // formal parameter
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG

```

VAR_IN_OUT CONSTANT variables are declared without assigning an initialization value.

Usage:

- When calling the POU, a constant variable or a literal is passed. All data types are permitted for this. Consequently, write access is not permitted.
- Passing parameters of string constants: The string length of the constants can be any size, and the string length does not depend on the string length of the VAR_IN_OUT CONSTANT variables.



If the option "Replace constants" is selected in the "Compile options" category in the project settings, the parameter transfer of a constant with a basic data type or a constant variable with a basic data type generates a compiler error.

Example

Passing parameters of string constants and string variables

```

FUNCTION funManipulate : BOOL
VAR_IN_OUT
  sReadWrite : STRING(16); (* Can be read or written here in POU *)
  dwVarReadWrite : DWORD; (* Can be read or written here in POU *)
END_VAR
VAR_IN_OUT CONSTANT
  c_sReadOnly : STRING(16); (* Constant string variable can only be read here in POU *)
END_VAR

sReadWrite := 'String from POU';
dwVarReadWrite := STRING_TO_DWORD(c_sReadOnly);

PROGRAM PRG_A
VAR
  sVarFits : STRING(16);
  sValFits : STRING(16) := '1234567890123456';
  dwVar : DWORD;
END_VAR

// The following line of code causes the compiler error C0417:
// C0417: VAR_IN_OUT parameter 'sReadWrite' needs a variable with write access as input.
funManipulate(sReadWrite:='1234567890123456', c_sReadOnly:='1234567890123456', dwVarReadWrite := dwVar);

// Correct code
funManipulate(sReadWrite := sValFits, c_sReadOnly := '23', dwVarReadWrite := dwVar);
funManipulate(sReadWrite := sVarFits, c_sReadOnly := sValFits, dwVarReadWrite := dwVar);

```

In the code, strings are passed to the funManipulate function via different VAR_IN_OUT variables. When passing a literal, a compiler error is output to a VAR_IN_OUT variable. When passing a constant variable to a VAR_IN_OUT CONSTANT variable, correct code is generated even for passing string variables.

Declaration of the function block MB_LoadCamData

```

FUNCTION_BLOCK MB_LoadCamData EXTENDS IL_ExecuteDoneBase
VAR_INPUT
  Permanent: BOOL:= FALSE; (*Setting, if cam is written permanent or temporary*)
  CamTableID: MC_CAM_ID:= CAM_TABLE_1; (*Selection of camtable from drive or control*)

```

```

END_VAR

VAR_IN_OUT
  CamTable: MB_CAMTABLE; (*Destination of value, contains the values of cam table
                           after finishing without an error and the number of elements*)
END_VAR

VAR_IN_OUT CONSTANT
  Slave: AXIS_REF; (*Selection of reference*)
END_VAR

```

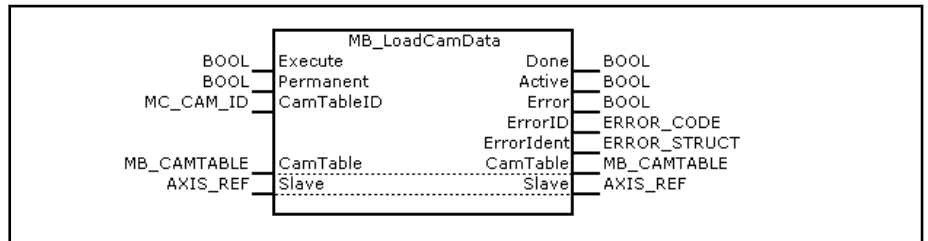


Fig. 17-32: Function block with VAR_IN_OUT "CamTable" and VAR_IN_OUT CONSTANT "slave"

"CamTable" example:

- Use the "CamTable" input of type MB_CAMTABLE to directly access (write or read-only) the memory range.

"Slave" example:

- The function block is provided with information on the desired control and on the axis controlled by this control via the input "Slave" of type AXIS_REF.

This information is used to address (write and/or read-only) the memory range and to access the corresponding data.

Also refer to:

- [chapter 9.2 "Declaring variables " on page 83](#)
- [chapter ""Project settings" - "Compiler options" dialog" on page 715](#)
- [chapter ""Function" object \(FUNCTION\)" on page 525](#)
- [chapter ""Function block" object \(FUNCTION_BLOCK\)" on page 523](#)
- [chapter ""Method" object " on page 532](#)
- [chapter ""Interface" object \(INTERFACE\)" on page 529](#)
- [chapter ""Interface method" object" on page 534](#)
- [chapter 17.2.10 "Constant variables - VAR CONSTANT" on page 287](#)

17.2.5 Global variables - VAR_GLOBAL

Global variables are "ordinary" variables, constants, external or remanent variables known in the entire project.

Declare global variables in Global Variables Lists or in the declaration section of programming objects between the keywords VAR_GLOBAL and END_VAR.

The system detects a global variable when a dot precedes the variable name (e.g. .iGlobVar1).



A locally declared variable in a function block with the same name as a global variable, has priority within the block.



For the compiler version 3.2.0.0 and later, IndraLogic always initializes global variables before initializing POU variables.

Example:

```
VAR_GLOBAL
    iVarGlob1:INT;
END_VAR
```

Also refer to

- [chapter 18.1.11 ""GVL" object - Global Variable List" on page 512](#)
- [chapter 17.3.67 "Operator - Global namespace" on page 337](#)

17.2.6 Temporary variable - VAR_TEMP

This functionality is an extension of the IEC 61131-3 standard.

Declare temporary variables locally between the keywords `VAR_TEMP` and `END_VAR`.

`VAR_TEMP` declarations can only be made in programs and function blocks.

IndraLogic initializes temporary variables each time the function block is called.

The application can access the temporary variables only in the implementation section of a program or a function block.

Example:

```
VAR_TEMP
    iVarTmp1:INT; (*1st temporary variable *)
END_VAR
```

17.2.7 Static variables - VAR_STAT

This function is an extension of the IEC 61131-3 standard.

Declare static variables locally between the keywords `VAR_STAT` and `END_VAR`.

IndraLogic initializes static variables upon the first call of each function block.

Static variables can only be accessed within the namespace. The variables are declared in this namespace (like static variables in C). But static variables retain their values when the application exits the function block. Use static variables as counters for function calls for example.

Extend static variables with an attribute keyword.

Example:

```
VAR_STAT
    iVarStat1:INT;
END_VAR
```

Also refer to [chapter 9.16 "Data persistence" on page 137](#)

17.2.8 External variables - VAR_EXTERNAL

External variables are global variables that are "imported" into a block.

Declare these variables between the keywords `VAR_EXTERNAL` and `END_VAR`. If the global variable does not exist, an error message is output.



It is not required in IndraLogic to declare variables as external. The keyword exists for maintaining the compliance with IEC 61131-3.

Syntax *Program:*

```
<POU keyword> <POU name>
VAR_EXTERNAL
  <variable name> : <data type>;
END_VAR
```

Initialization is not permitted.

Example

```
FUNCTION_BLOCK FB_DoSomething
VAR_EXTERNAL
  iVarExt1 : INT; (* 1st external variable *)
END_VAR
```

Also refer to:

- [chapter 18.1.11 ""GVL" object - Global Variable List" on page 512](#)

17.2.9 Instance variables - VAR_INST

IndraLogic does not save a VAR_INST method variable in a method stack, but in the stack of the function block instance.

This means that the VAR_INST variable functions like other variables of the function block instance and it is not reinitialized each time the method is called.

VAR_INST variables are only permitted in methods. These variables can only be accessed within the method.

The variable values of instance variables are monitored in the declaration section of the method.

Extend instance variable with an attribute keyword.

Example:

```
METHOD meth_last : INT
VAR_INPUT
  iVar : INT;
END_VAR
VAR_INST
  iLast : INT := 0;
END_VAR
meth_last := iLast;
iLast := iVar;
```

Also refer to:

- [chapter ""Method" object " on page 532](#)

17.2.10 Constant variables - VAR CONSTANT

Constant variables are declared in global variable lists or in the declaration part of programming objects. In implementations, constant variables can be accessed as read-only via the instance path.

Syntax *Program:*

```
<scope> CONSTANT
  <identifier> : <data type> :=
  <initialization> ;
END_VAR

<scope> : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
```



Always assign an initialization value when declaring a constant variable. Then the constant cannot be written any more.

Example *Declaration*

```
VAR CONSTANT
  c_rTAXFACTOR : REAL := 1.19;
END_VAR
```

Call

```
rPrice := rValue * c_rTAXFACTOR;
```

In implementations, you can access constant variables exclusively as read-only.

Constant variables are located to the right of the assignment operator.

Also refer to [chapter 17.2.4 "Input and output variable - VAR_IN_OUT" on page 281](#)

17.2.11 Persistent variables - PERSISTENT

Persistent variables are declared in the declaration section `VAR_GLOBAL RETAIN PERSISTENT` in the persistent global variable list. For variables that are marked with the `PERSISTENT` keyword outside of the persistence editor, instance paths are added there.

Syntax of the declaration in the global persistent variable list PersistentVars

```
VAR_GLOBAL PERSISTENT RETAIN
  <identifier>: <data type> (: <initialization>)?;
  <instance path to POU variable>
END_VAR
```

Syntax of the declaration in POU's

```
<scope> PERSISTENT RETAIN
  <identifier>: <data type> ( := <initialization> )?; // (...)? : Optional
END_VAR
<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT | VAR_GLOBAL
```



An assignment of inputs, outputs, or memory addresses with the `AT` keyword is not permitted.

Never use the `POINTER TO` data type in persistent variable lists, as the addresses might change when the application is downloaded again. The corresponding compiler warnings are shown in the message window.

If you frequently change the names or data types of remanent variables, it is recommended to declare the remanent variables as retain variables only with the keyword `RETAIN`.

Declare variables directly in the list of persistent variables and avoid inserting instance paths, as the use of instance paths requires double the memory space and increases the cycle time!

Example *Declaration in the persistent variable list PersistentVars:*

```
{attribute 'qualified_only'}
VAR_GLOBAL PERSISTENT RETAIN
  g_iCounter : INT;
  // Generated instance path of persistent variable
```

```

PLC_PRG.fb_A.iPersistentCounter_A: INT;
END_VAR

```

Declaration in the function block FB_A:

```

FUNCTION_BLOCK FB_A
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR_PERSISTENT
    iPersistentCounter_A : INT;
END_VAR

```

Declaration in the program PLC_PRG:

```

VAR
    fb_A1 : FB_A;
END_VAR

```

Possible declaration locations

Declaration location:	Effect:
Directly in the persistent global variable list	The variable is persistent and lies in the protected memory area.
Locally in a program with an instance path in the persistent variable list	The variable is persistent and lies in the protected memory area and in the memory (double allocation).
Locally in a function block with an instance path in the persistent variable list	
Only locally in a program	This variable is not persistent. A warning is shown in the message window. Tip: Click on "Declarations" → "Add all instance paths" to import the variables into the persistent variable list.
Only locally in a function block	
Locally in a function	This declaration does not have any effect. The variable is not persistent.

Tab. 17-10:

In the persistence editor, click "Add All Instance Paths" if local variables are marked with `PERSISTENT`.



Avoid marking variables, which are declared in a function block, with `PERSISTENT`. This is because the function block instance is stored entirely in remanent memory and not just the marked variable.

Also refer to:

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter 9.16.4 "Declaring VAR PERSISTENT variables" on page 143](#)
- [chapter "Command 'Add all instance paths'" on page 679](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)

17.2.12 Retain Variable - RETAIN

Retain variables are declared by adding the keyword `RETAIN` to programming objects in the validity range `VAR`, `VAR_INPUT`, `VAR_OUTPUT`, `VAR_IN_OUT`, `VAR_STAT` or `VAR_GLOBAL`.

Syntax in the declaration

```
<scope> RETAIN
  <identifier>: <data type> ( := <initialization> )? // (...)?: Optional
END_VAR
<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT | VAR_GLOBAL
```



An assignment of inputs, outputs, or memory addresses with the `AT` keyword is not permitted.

Example *In a POU:*

```
VAR RETAIN
  iVarRetain: INT;
END_VAR
```

In a GVL:

```
VAR_GLOBAL RETAIN
  g_iVarRetain: INT;
END_VAR
```

Possible declaration locations

Declaration location:	Effect:
Locally in a program	Only the variable lies within the retain memory area.
Global in a global variable list	Only the variable lies within the retain memory area.
Local in a function block	The entire instance of the function block with all of its data lies within the retain memory area. Only the declared retain variable is protected.
Locally in a function	Even the variable does not lie within the retain memory area. This declaration does not have any effect.
Local and persistent in a function	Even the variable does not lie within the retain memory area. This declaration does not have any effect.

Tab. 17-11:



Avoid marking variables of a function block with `RETAIN`.

Also refer to:

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter 9.16.4 "Declaring VAR PERSISTENT variables" on page 143](#)
- [chapter "Command 'Add all instance paths'" on page 679](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)

17.2.13 Configuration variable - VAR_CONFIG

Configuration variables are used for assigning the complete address to variables that are declared in function blocks with incomplete address information and that should be mapped to device I/Os.

Declare the variables in a Global Variable List between `VAR_CONFIG` and `END_VAR`. This Global Variable List is called "variable configuration".

Enter the configuration variables with a complete instance path and the correct address into the "Variable configuration".

Example

Declaration of the variable `xLocIn` with the incomplete address `%I*` in a function block:

```
FUNCTION_BLOCK locio
VAR
    xLocIn AT %I*: BOOL := TRUE;
END_VAR
```

The `locio` function block is used in the `PlcProg` program:

```
PROGRAM PlcProg
VAR
    locioVar1: locio;
END_VAR
```

The correct variable configuration in the Global Variable List is:

```
VAR_CONFIG
    PlcProg.locioVar1.xLocIn AT %IX1.0 : BOOL;
END_VAR
```

Also refer to

- [chapter 9.10.1 "Variable configuration - VAR_CONFIG" on page 123](#)

17.2.14 SUPER

`SUPER` is a special variable used for object-oriented programming.

`SUPER` is the pointer of a function block to the basic function block instance used to generate the function block. The `SUPER` pointer thus also allows to access the implementation of the methods of the basic function block (basic class). A `SUPER` pointer is automatically available for each function block.

`SUPER` can only be used in methods and in the respective function block implementations.

Dereferencing the pointer: `SUPER^`

Use of the `SUPER` pointer: Use the keyword `SUPER` to call a method that is valid in the instance of the basic class or parent class.

Example:

Examples

ST:

```
SUPER^.METH_DoIt();
```

FBD/CFC/LD

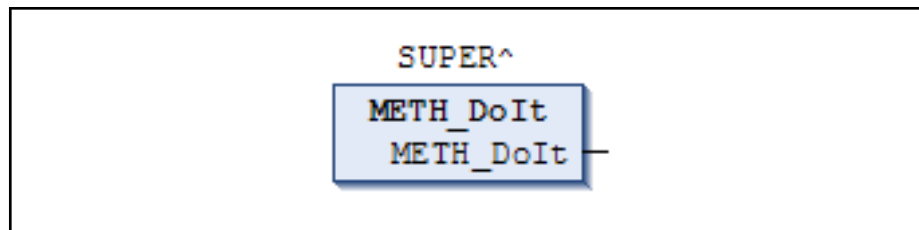


Fig. 17-33:



SUPER is not yet implemented for the statement list (IL).

Examples:

Use of SUPER and THIS pointers

```

FUNCTION_BLOCK FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
    iCnt : INT;
    iRes : INT;
END_VAR
VAR
END_VAR
END_VAR

THIS^.METH_DoIt();
THIS^.METH_DoAlso();

    METHOD METH_DoIt : BOOL
    VAR
    END_VAR
    iCnt := -1;
    METH_DoIt := TRUE;

    METHOD METH_DoAlso : BOOL
    VAR
    END_VAR
    iRes := -5;
    METH_DoAlso := TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
    iCnt_Base: INT;
    iCnt_THIS: INT;
    iRes_Base: INT;
    iRes_THIS: INT;
END_VAR
VAR
END_VAR
// Calls the method defined under FB_1
THIS^.METH_DoIt();
THIS^.METH_DoAlso();
iCnt_THIS:= iCnt;
iRes_THIS:= iRes;
// Calls the method defined under FB_Base
SUPER^.METH_DoIt();
SUPER^.METH_DoAlso();
iCnt_Base:= iCnt;
iRes_Base:= iRes;

    METHOD METH_DoIt : BOOL
    VAR
    END_VAR
    iCnt := 1111;
    METH_DoIt := TRUE;

PROGRAM PlcProg
VAR
    Myfb_1: FB_1;
    iFB: INT;
    iBase: INT;
END_VAR

```

```
Myfb_1();
iBase := Myfb_1.iCnt_Base;
iFB := Myfb_1.iCnt_THIS;
```

Also refer to

- [chapter 17.5 "Data types" on page 351](#)
- [chapter 17.2.15 "THIS" on page 293](#)

17.2.15 THIS

THIS is a special variable and used for object-oriented programming.

THIS is the pointer of a function block to its own function block instance. A THIS pointer is automatically available for each function block.

THIS can only be used in methods and in function blocks. THIS is provided for the implementation in the input assistant in the category **Keywords**.

Dereferencing the pointer: THIS^

Using the THIS pointer

- If a local variable shades a function block variable in a method, the function block variable can be set with the THIS pointer. See example below (1)
- If the pointer to the own function block instance is referenced for use in a function. (See example below (2))

Example:

Examples

ST:

```
THIS^.METH_DoIt();
```

FBD/CFC/LD:

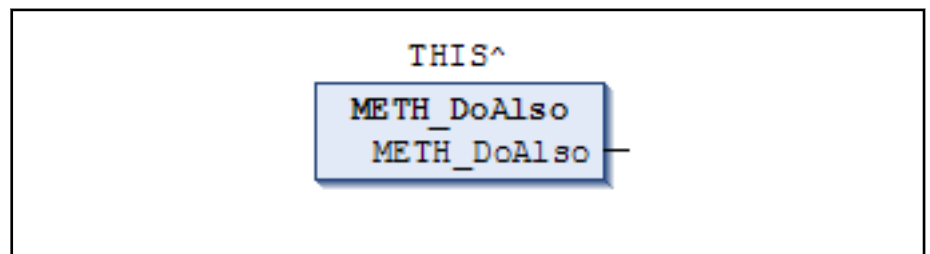


Fig. 17-34:



THIS is not yet implemented for the statement list (IL).

Examples

The local variable iVarB shades the function block variable iVarB:

```
FUNCTION_BLOCK fbA
VAR_INPUT
    iVarA: INT;
END_VAR
iVarA := 1;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;
```

```

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
    iVarB: INT;
END_VAR
iVarB := 22;           // Here the local iVarB is set.
THIS^.iVarB := 222;    // Here the function block variable iVarB is set,
                        // although iVarB is overloaded.

PROGRAM PlcProg
VAR
    MyfbB: fbB;
END_VAR

MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();

```

A function call requires the reference to the individual instance:

```

FUNCTION funA : BOOL
VAR_INPUT
    pFB: fbA;
END_VAR
...;

FUNCTION_BLOCK fbA
VAR_INPUT
    iVarA: INT;
END_VAR
...;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
    iVarB: INT;
END_VAR
iVarB := 22;           // Here the local iVarB is set.
funA(pFB := THIS^);    // Here funA is called with THIS^.

PROGRAM PlcProg
VAR
    MyfbB: fbB;
END_VAR
MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();

```

Also refer to

- [chapter 17.5.12 "Data type 'POINTER'" on page 359](#)
- [chapter 17.2.14 "SUPER" on page 291](#)

17.3 Operators

IndraLogic 2G supports all IEC-61131-3 operators.

These operators are implicitly known throughout the project.

In addition to these IEC operators, IndraLogic also supports some non-IEC 61131-3 operators.

Operators are used in a function block like functions.



For information on the processing order (binding strength) of the ST operators, refer to [chapter "ST expressions" on page 226](#).

NOTICE**Operations with floating point data points**

For operations with floating point data types, the computational result depends on the applied target system hardware.

NOTICE**For operations with overflow or underflow**

For operations with overflow or underflow in the data type, the computational result depends on the applied target system hardware.

Overflow/underflow in the data type

The IndraLogic compiler generates code for the control type and always computes temporary results with the native size defined by the control. This provides considerable advantages in the computation speed and often also produces the desired result. But this also means that an overflow or underflow in the data type is not truncated in some cases.

Example1 The result of this addition is not truncated and the result in `dwVar` is 65536.

```
VAR
    wVar : WORD;
    dwVar: DWORD;
END_VAR

wVar := 65535;
dwVar := wVar + 1;
```

Example2 The overflow and underflow in the data type is not truncated and the results (`bVar1`, `bVar2`) of both comparisons are `FALSE`.

```
VAR
    wVar1 : WORD;
    wVar2 : WORD;
    bVar1 : BOOL;
    bVar2 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
bVar1 := (wVar1 + 1) = wVar2;
bVar2 := (wVar2 - 1) = wVar1;
```

Example3 By the assignment to `wVar3`, the value is truncated to the target data type `WORD` and the result `bVar1` is `TRUE`.

```
VAR
    wVar1 : WORD;
    wVar2 : WORD;
    wVar3 : WORD;
    bVar1 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
wVar3 := (wVar1 + 1);
bVar1 := wVar3 = wVar2;
```

Example4 In order to force the compiler to truncate the temporary results, a conversion can be inserted.

The type conversion makes sure that both comparisons are 16-bit only and the results (bVar1, bVar2) of both comparisons are each TRUE.

```

VAR
  wVar1 : WORD;
  wVar2 : WORD;
  bVar1 : BOOL;
  bVar2 : BOOL;
END_VAR

wVar1 := 65535;
wVar2 := 0;
bVar1 := TO_WORD(wVar1 + 1) = wVar2;
bVar2 := TO_WORD(wVar2 - 1) = wVar1;

```

Arithmetic Operators	chapter 17.3.1 "Operator "ADD"" on page 298 chapter 17.3.3 "Operator "SUB"" on page 299 chapter 17.3.2 "Operator "MUL"" on page 298 chapter 17.3.4 "Operator "DIV"" on page 299 chapter 17.3.5 "Operator "MOD"" on page 300 chapter 17.3.6 "Operator "MOVE"" on page 301 chapter 17.3.7 "Operator "INDEXOF"" on page 301 chapter 17.3.8 "Operator "SIZEOF"" on page 301
Bit String Operators	chapter 17.3.10 "Operator "AND"" on page 302 chapter 17.3.11 "Operator "OR" " on page 302 chapter 17.3.12 "Operator "XOR"" on page 303 chapter 17.3.9 "Operator "NOT"" on page 302 chapter 17.3.13 "Operator "AND_THEN"" on page 303 chapter 17.3.14 "Operator "OR_ELSE"" on page 304
Bit shift operators	chapter 17.3.15 "Operator "SHL"" on page 304 chapter 17.3.16 "Operator "SHR"" on page 305 chapter 17.3.17 "Operator "ROL"" on page 305 chapter 17.3.18 "Operator "ROR"" on page 306
Selection Operators	chapter 17.3.19 "Operator 'SEL'" on page 307 chapter 17.3.20 "Operator "MAX"" on page 308 chapter 17.3.21 "Operator "MIN"" on page 308 chapter 17.3.22 "Operator "LIMIT"" on page 308 chapter 17.3.23 "Operator "MUX"" on page 309
Relational Operators	The comparison operators are Boolean operators comparing two inputs (first operand and second operand). chapter 17.3.24 "Operator "GT"" on page 309 chapter 17.3.25 "Operator "LT"" on page 310 chapter 17.3.26 "Operator "LE"" on page 310 chapter 17.3.27 "Operator "GE"" on page 310 chapter 17.3.28 "Operator "EQ"" on page 311 chapter 17.3.29 "Operator "NE"" on page 311
Address Operators	chapter 17.3.30 "Operator "ADR"" on page 311 chapter 17.3.31 "Operator "Content operator"" on page 312

	chapter 17.3.32 "Operator "BITADR"" on page 313
Call Operators	chapter 17.3.33 "Operator "CAL"" on page 313
Type Conversion Operators	Implicit conversion from a "larger" type into a "smaller" type is not possible, e.g. from INT into BYTE or from DINT into WORD. To convert from a "larger" type into a "smaller" type, use special type conversion. As a rule, any elementary type can be converted into any other elementary type. Types conversion: <Elementary Type1>_TO_<Elementary Type2> Overflow conversion: TO_<Elementary Type2> IndraLogic generates the string left-aligned in ...TO_STRING conversions. If the string length defined is too short, it is truncated from the right. If the input value of a type conversion operator is beyond the value range of the output data type, the result of the operation is undefined and depends on the control type. One example is if the input value for the conversion from LREAL to DINT is negative. chapter 17.3.34 "Operator 'BOOL_TO'" on page 314 Also refer to chapter 17.3.35 "Operator 'TO_BOOL'" on page 315 chapter 17.3.37 "Operator "<INT type>_TO_<INT type>" on page 316 chapter 17.3.38 "Operator 'REAL_TO-/ LREAL-TO'" on page 316 chapter 17.3.39 "Operator 'TIME_TO/TIME_OF_DAY_TO'" on page 317 chapter 17.3.40 "Operator 'DATE_TO/DT_TO'" on page 318 chapter 17.3.41 "Operator 'STRING_TO'" on page 319 chapter 17.3.42 "Operator 'TRUNC'" on page 320 chapter 17.3.43 "Operator 'TRUNC_INT'" on page 321 chapter 17.3.36 "Operator "TO_<xxx>" on page 315
Numeric Operators	chapter 17.3.44 "Operator "ABS"" on page 321 chapter 17.3.45 "Operator "SQRT"" on page 321 chapter 17.3.46 "Operator "LN"" on page 322 chapter 17.3.47 "Operator "LOG"" on page 322 chapter 17.3.48 "Operator "EXP"" on page 322 chapter 17.3.49 "Operator "EXPT"" on page 323 chapter 17.3.50 "Operator "SIN"" on page 323 chapter 17.3.53 "Operator "ASIN"" on page 324 chapter 17.3.51 "Operator "COS"" on page 324 chapter 17.3.52 "Operator "TAN"" on page 324 chapter 17.3.54 "Operator "ACOS"" on page 325 chapter 17.3.55 "Operator "ATAN"" on page 325
Namespace operators	Namespace operators are an extension of IEC 61131-3 operators. They provide unique access to variables and modules, even if the same name is used multiple times for variables or modules in the project. chapter 17.3.67 "Operator - Global namespace" on page 337

Other operators

[chapter 17.3.68 "Operator - Namespace for Global Variable Lists" on page 337](#)

[chapter 17.3.70 "Operator - Enumeration namespace" on page 338](#)

[chapter 17.3.69 "Operator - Library namespace" on page 338](#)

[chapter 17.3.58 "Operator '__NEW'" on page 327](#)

[chapter 17.3.56 "Operator '__DELETE'" on page 326](#)

[chapter 17.3.57 "Operator '__ISVALIDREF'" on page 327](#)

[chapter 17.3.59 "Operator '__QUERYINTERFACE'" on page 329](#)

[chapter 17.3.60 "Operator '__QUERYPOINTER'" on page 330](#)

[chapter 17.3.71 "Operator 'INI'" on page 338](#)

[chapter 17.3.61 "Operators '__TRY', '__CATCH', '__FINALLY', '__END-TRY'" on page 331](#)

17.3.1 Operator "ADD"

This IEC operator is used for adding variables.

Data types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME, TIME_OF_DAY (TOD), DATE, DATE_AND_TIME (DT)

Possible combinations for TIME data types: TIME+TIME = TIME, TOD+TIME = TOD, DT+TIME = DT

Feature in the FBD/LD editor: The ADD operator can be extended by function block inputs. The number of additional function block inputs is limited.

Examples

ST:

```
var1 := 7+2+4+7;
```

FBD:

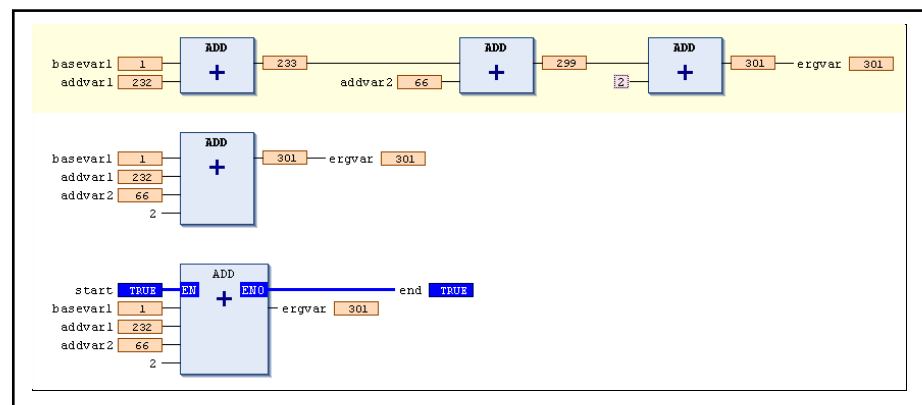


Fig. 17-35:

17.3.2 Operator "MUL"

This IEC operator is used to multiply variables.

Data types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

Feature in the FBD/LD editor: The MUL operator can be extended by additional function block inputs. The number of additional function block inputs is limited.

Examples

ST:

```
var1 := 7*2*4*7;
```

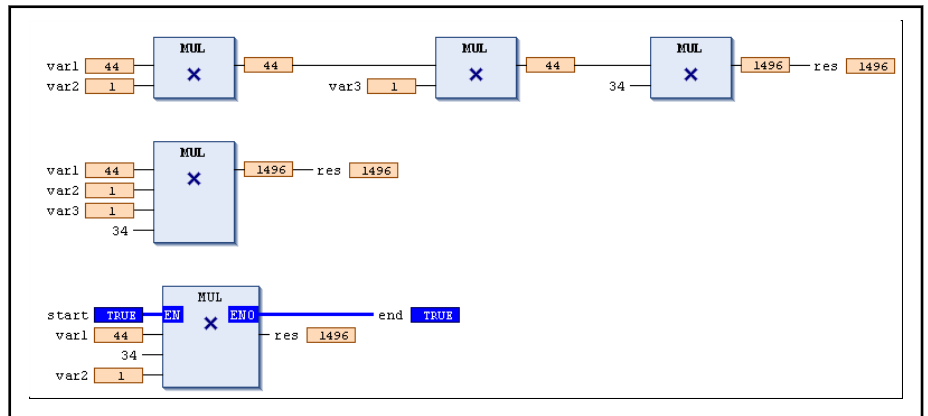
FBD:

Fig. 17-36:

17.3.3 Operator "SUB"

This IEC operator is used to subtract variables.

Data types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME, TIME_OF_DAY (TOD), DATE, DATE_AND_TIME (DT)

Possible combinations for TIME data types: TIME-TIME = TIME, DATE-DATE = TIME, TOD-TIME = TOD, TOD-TOD = TIME, DT-TIME = DT, DT-DT = TIME



Negative TIME values are undefined.

Examples

ST:

```
var1 := 7-2;
```

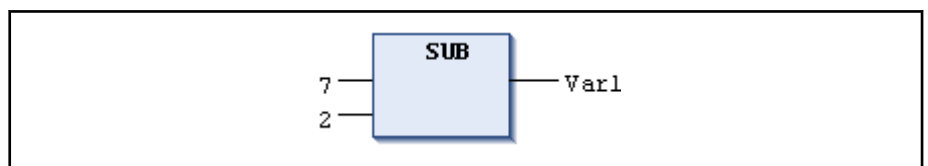
FBD:

Fig. 17-37:

17.3.4 Operator "DIV"

This IEC operator is used for dividing variables.

Data types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME



Division by zero may have different results depending on the target system!

Examples

ST:

```
var1 := 8/2;
```

FBD:

1. Series of DIV function blocks, 2. Single DIV function block, 3. DIV function blocks with EN/ENO parameters

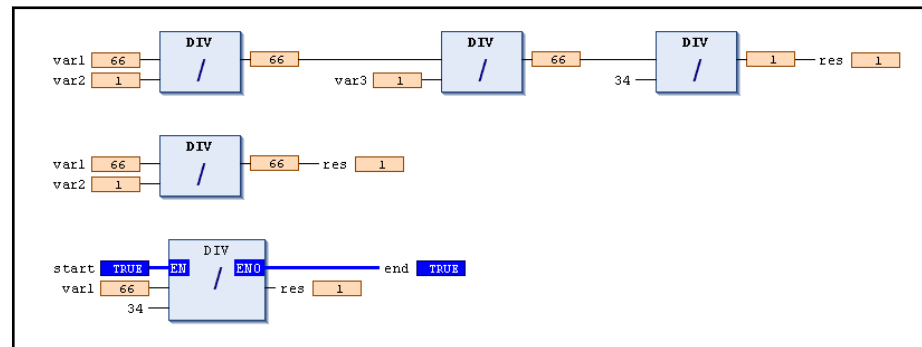


Fig. 17-38:



Note that it is possible to monitor a division by zero at runtime by using the implicit monitoring functions `CheckDivInt`, `CheckDivLint`, `CheckDivReal` and `CheckDivLReal`.

Also refer to

- [chapter 18.1.18 "Monitoring the PLC program of the application at run-time" on page 542](#)
- ["CheckDivInt" on page 544](#)
- ["CheckDivLint" on page 544](#)
- ["CheckDivReal" on page 544](#)
- ["CheckDivLReal" on page 544](#)

17.3.5 Operator "MOD"

This IEC operator is used for modulo division.

The result of the function is the integer remaining of the division.

Data types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT



Division by zero may have different results depending on the target system!

Examples

Result in Var1: 1

ST:

```
var1 := 9 MOD 2;
```

FBD:

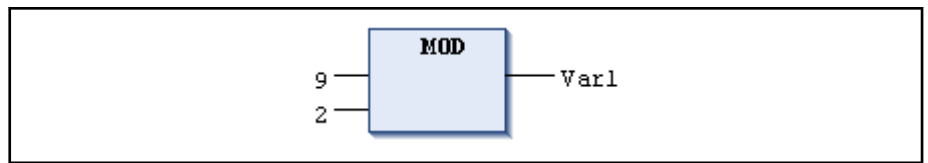


Fig. 17-39:

17.3.6 Operator "MOVE"

This IEC operator is used for assigning a variable to another variable of a corresponding type.

As **MOVE** is available as function block in the CFC, FBD and LD editors, the EN/ENO functionality can also be used for a variable assignment.

Example:

CFC: Together with EN/ENO function:

IndraLogic assigns the variable value `var1` to the variable `var2` only if `en_i` is TRUE.

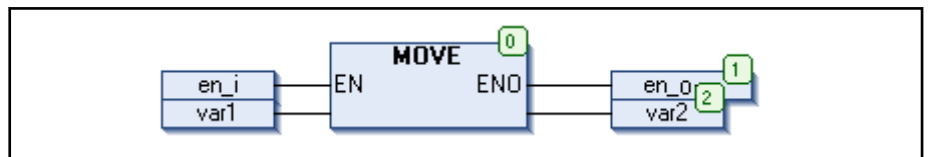


Fig. 17-40:

ST:

```
ivar2 := MOVE(ivar1);
```

This corresponds to:

```
ivar2 := ivar1;
```

17.3.7 Operator "INDEXOF"

The operator is an extension of the IEC 61131-3 standard.

Instead of the **INDEXOF** operator, the **ADR** operator is provided in IndraLogic 2G to obtain a pointer at the index of a function block.

Also refer to

- [chapter 17.3.30 "Operator "ADR"" on page 311](#)

17.3.8 Operator "SIZEOF"

This operator is an extension of the IEC 61131-3 standard.

This operator is used for defining the number of bytes required by the variable `x`. The **SIZEOF** operator always provides an unsigned value. The type of return variable adapts to the detected size of the variable `x`.

Return value of SIZEOF (x)	Data type of the constant used implicitly for the detected size by IndraLogic.
0 <= size of x < 256	USINT
256 <= size of x < 65536	UINT
65536 <= size of x < 4294967296	UDINT
4294967296 <= size of x	ULINT

Examples

Result in var1 is 10.

Declaration

```
arr1 : ARRAY[0..4] OF INT;
var1 : INT;
```

ST:

```
var1 := SIZEOF(arr1); (* var1:=USINT#10; *)
```

17.3.9 Operator "NOT"

The IEC operator is used for the bit-by-bit NOT of a bit operand.

When the respective input bit is 0, the output bit is also 1 and vice-versa.

Data types allowed: BOOL, BYTE, WORD, DWORD, LWORD

Examples

Result in Var1 is 2#0110_1100

ST:

```
Var1 := NOT 2#1001_0011
```

FBD:

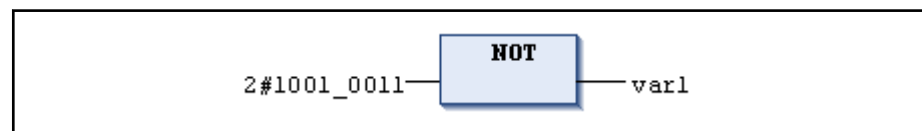


Fig. 17-41:

17.3.10 Operator "AND"

This IEC operator is used for the bit-by-bit AND of bit operands.

If the input bits are 1, the output bit is 1. Otherwise, it is 0.

Data types allowed: BOOL, BYTE, WORD, DWORD, LWORD

Examples

Result in Var1: 2#1000_0010

ST:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

FBD:

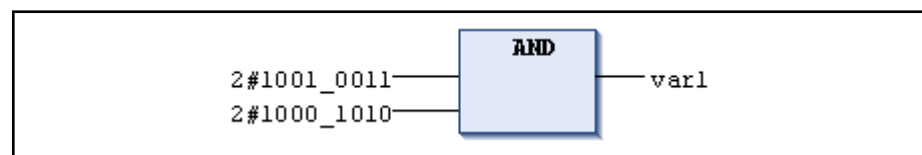


Fig. 17-42:

17.3.11 Operator "OR"

This IEC operator is used for the bit-by-bit OR of bit operands.

If at least one of the input bits is 1, the output bit is also 1, otherwise 0.

Data types allowed: BOOL, BYTE, WORD, DWORD, LWORD

Examples

Result in Var1 is 2#1001_1011

ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

FBD:

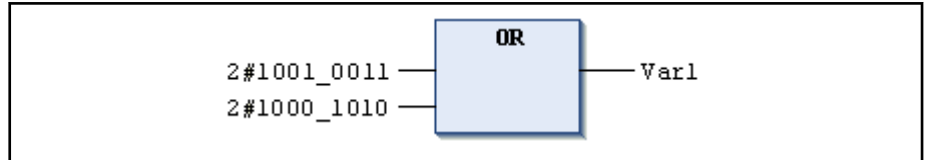


Fig. 17-43:

17.3.12 Operator "XOR"

The IEC operator is used for the bit-by-bit **XOR** of bit operands.

If only one of the two input bits is 1, the output bit is also 1. If both inputs are 1 or 0, the output is 0.

Data types allowed: BOOL, BYTE, WORD, DWORD, LWORD



Please note the following response of the **XOR** function block in extended version (for more than two inputs): IndraLogic compares the inputs in pairs and then the corresponding results (according to the standard, but not necessarily according to expectations).

Examples

Result in Var1 is 2#0001_1001

ST:

```
var1 := 2#1001_0011 XOR 2#1000_1010
```

FBD:

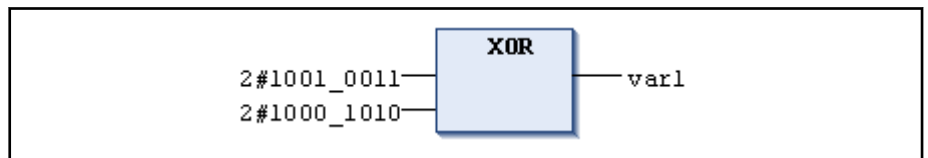


Fig. 17-44:

17.3.13 Operator "AND_THEN"

This operator is an extension of the IEC 61131-3 standard.

The **AND_THEN** operator is only allowed for programming in Structured Text in case of the following operation: The **AND** operation of **BOOL** and **BIT** operands with short-circuit evaluation. That means:

If all operands are **TRUE**, the result of the operands is also **TRUE**, otherwise **FALSE**.

But: Only if the first operand of the **AND_THEN** operator is **TRUE**, IndraLogic also executes the expressions on other operands. This can prevent problems with null pointers, e.g. in conditions such as **IF (ptr <> 0 AND_THEN ptr^ = 99) THEN....**

In contrast, IndraLogic always evaluates all operands when using the **AND** IEC operator.

Also refer to

- [chapter 17.3.10 "Operator "AND"" on page 302](#)

17.3.14 Operator "OR_ELSE"

The operator is an extension of the IEC 61131-3 standard.

The `OR_ELSE` operator is only allowed for programming in Structured Text: OR operation of `BOOL` and `BIT` operands with short-circuit evaluation. That means:

If at least one of the operands is `TRUE`, the result of the operation is also `TRUE`. Otherwise, it is `FALSE`.

In contrast to the use of the `OR` IEC operator, for `OR_ELSE`, the expressions on all other operators are not evaluated anymore as soon as one of the operands was evaluated with `TRUE`

Example:

```
VAR
  bEver : BOOL;
  bX    : BOOL;
  dw    : DWORD := 16#000000FF;
END_VAR

bEver := FALSE;
bX := dw.8
  OR_ELSE dw.1
  OR_ELSE dw.1
  OR_ELSE (bEver := TRUE);
```

`dw.8` is `FALSE` and `dw.1` is `TRUE`. Thus, `bX` is the result of the operation `TRUE`. However, the expression at the third input is not executed and `bEver` remains `FALSE`. If the standard `OR` operation was used instead, `bEver` would have been set to `TRUE`.

Also refer to

- [chapter 17.3.11 "Operator "OR" " on page 302](#)

17.3.15 Operator "SHL"

This IEC operator is used for the bit-by-bit shifting of an operand to the left.

`erg := SHL (in, n)`

`in`: Operand shifted to the left.

`n`: Number of bits by which `in` is shifted to the left.



If `n` overwrites the data type width, it depends on the target system how the `BYTE`, `WORD`, `DWORD` and `LWORD` operands are filled. The target systems cause a filling with zeros or `n MOD <Tab-Width>`.



Note that the number of bits that IndraLogic uses for this computation is specified by the data type of the input variable `in`.

Examples

The results for `erg_byte` and `erg_word` are different, although the values of the input variables `in_byte` and `in_word` are the same and the data types of the input variables are different.

ST:

```
PROGRAM shl_st
VAR
  in_byte : BYTE:=16#45; (* 2#01000101 )
  in_word : WORD:=16#0045; (* 2#0000000001000101 )
```



```

erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR

erg_byte := SHL(in_byte,n); (* Result is 16#14, 2#00010100 *)
erg_word := SHL(in_word,n); (* Result is 16#0114, 2#0000000100010100 *)

```

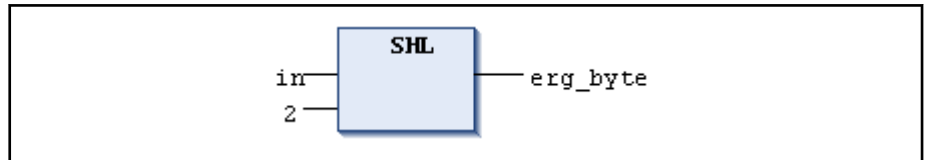
FBD:

Fig. 17-45:

17.3.16 Operator "SHR"

This IEC operator is used for the bit-by-bit shifting of an operand to the right.

```
erg := SHR (in, n)
```

in: Operand shifted to the right.

n: Number of bits by which in is shifted to the right.



If *n* exceeds the data type width, it depends on the target system how BYTE, WORD, DWORD and LWORD operands are completed with data. The target systems cause a filling with zeros or $n \bmod \langle \text{TabWidth} \rangle$.

Examples**ST:**

```

PROGRAM shr_st
VAR
  in_byte : BYTE :=16#45; (* 2#01000101 *)
  in_word : WORD :=16#0045; (* 2#0000000001000101 *)
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
END_VAR

erg_byte := SHR(in_byte,n); (* Result is 16#11, 2#00010001 *)
erg_word := SHR(in_word,n); (* Result is 16#0011, 2#0000000000010001 *)

```

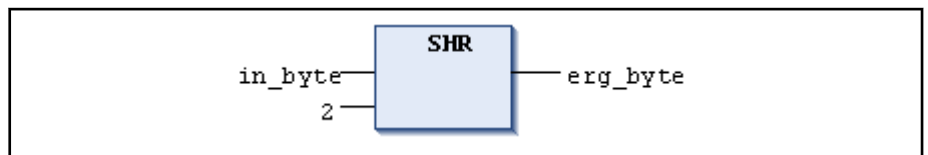
FBD:

Fig. 17-46:

17.3.17 Operator "ROL"

This IEC operator is used for a bit-by-bit rotation of an operand to the left.

Data types allowed: BYTE, WORD, DWORD, LWORD

```
erg := ROL (in, n)
```

IndraLogic moves *in* *n*-times to the left by one bit and inserts the bit into the leftmost position from the right.



Note that the number of bits that IndraLogic uses for this computation is specified by the data type of the input variable `in`. If this is a constant, IndraLogic uses the smallest possible data type. The data type of the output variables still does not affect this computer operation.

Examples

The results for `erg_byte` and `erg_word` are different depending on the data type of the input variables, although the values of the input variables `in_byte` and `in_word` are the same.

ST:

```
PROGRAM rol_st

VAR
  in_byte  : BYTE := 16#45;
  in_word  : WORD := 16#45;
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE := 2;
END_VAR

erg_byte := ROL(in_byte,n); (* Result: 16#15 *)
erg_word := ROL(in_word,n); (* Result: 16#0114 *)
```

FBD:

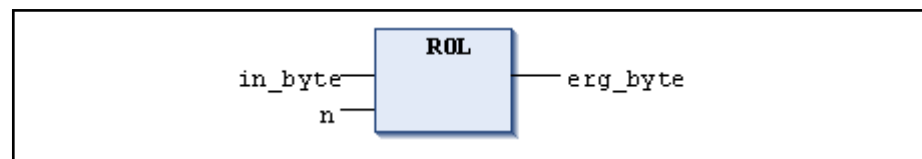


Fig. 17-47:

17.3.18 Operator "ROR"

The IEC operator is used for a bit-by-bit rotation of an operand to the right.

Data types allowed: BYTE, WORD, DWORD, LWORD

`erg := ROR(in,n)`

IndraLogic moves `in` `n`-times one bit to the right and adds the bit to the right-most position from the left.



Note that the number of bits that IndraLogic uses for this computation is specified by the data type of the input variable `in`. If this is a constant, IndraLogic uses the smallest possible data type. The data type of the output variable does not affect the arithmetic operation.

Examples

The results for `erg_byte` and `erg_word` are different depending on the data type of the input variables, although the values of the input variables `in_byte` and `in_word` are the same.

ST:

```
PROGRAM ror_st

VAR
  in_byte  : BYTE := 16#45;
  in_word  : WORD := 16#45;
```

```

erg_byte : BYTE;
erg_word : WORD;
n        : BYTE := 2;
END_VAR

erg_byte := ROR(in_byte,n); (* Result: 16#51 *)
erg_word := ROR(in_word,n); (* Result: 16#4011 *)

```

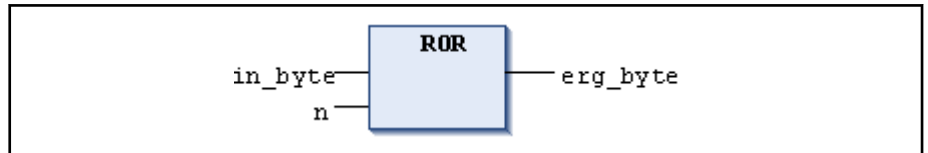
FBD:

Fig. 17-48:

17.3.19 Operator 'SEL'

The IEC operator is used for bitwise selection.

OUT := SEL(G, IN0, IN1) means:

OUT := IN0; if G = FALSE

OUT := IN1; if G = TRUE

Data types allowed:

IN0, ..., INn and OUT: Any, identical data type. Ensure that variables of the identical type are used at all three positions, especially when using user-defined data types.

The compiler checks for type identity and returns any compile errors. The assignment of function block instances to interface variables is specifically not supported.

G: BOOL



IndraLogic does not compute an expression preceding IN0 if G is TRUE.

IndraLogic does not compute an expression preceding IN1 if G is FALSE.

Note: In the case of graphical programming languages, the expressions at IN0 and IN1 are computed independently of the G input when a "Box", "Jump", "Return", "Line Branch", or "Edge Detection" precedes.

Examples

ST:

```
Var1 := SEL(TRUE,3,4); (* Result: 4 *)
```

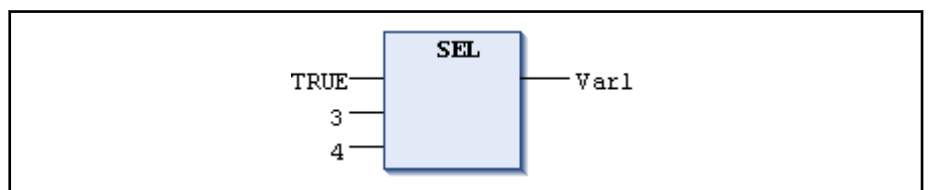
FBD:

Fig. 17-49:

17.3.20 Operator "MAX"

This IEC operator is used for the maximum function. It provides the larger value of the two values.

OUT := MAX(IN0, IN1)

Permitted data types: All

Examples

ST, result: 90

Var1 := MAX(30, 40);

Var1 := MAX(40, MAX(90, 30));

FBD:

Result: 90

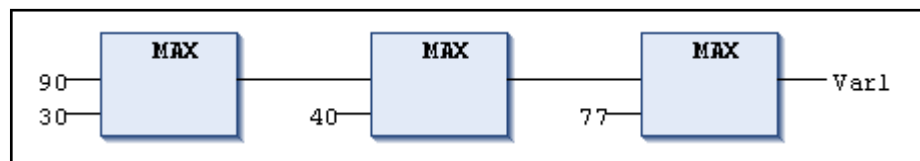


Fig. 17-50:

17.3.21 Operator "MIN"

This IEC operator is used for the minimum function. It provides the smaller value of the two values.

OUT := MIN(IN0, IN1)

Permitted data types: All

Examples

Result: 30

ST

Var1 := MIN(90, 30);

Var1 := MIN(MIN(90, 30), 40);

FBD:

Result: 30

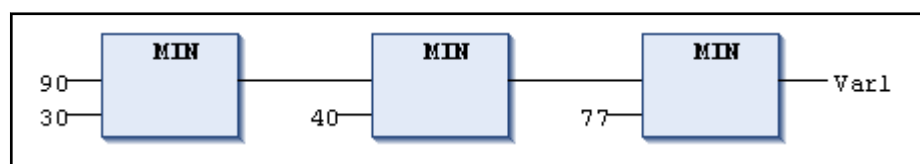


Fig. 17-51:

17.3.22 Operator "LIMIT"

This IEC selection operator is used for limiting.

OUT := LIMIT(Min, IN, Max)

This means: OUT := MIN (MAX (IN, Min), Max)

Max is the upper limit and Min is the lower limit for the result. If the value IN exceeds the upper limit Max, LIMIT Max returns Max. If IN is lower than IN Min, the result is Min.

Permitted data types for IN and OUT: All

Examples

Result in Var1 is 80

ST:

```
Var1 := LIMIT(30,90,80);
```

17.3.23 Operator "MUX"

This IEC operator is used as a multiplexer

```
OUT := MUX(K, IN0, ..., INn)
```

This means $OUT = IN_K$

Permitted data types for K: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT, UDINT.

IN0, ..., INn, and OUT: Any, identical data type. Ensure that variables of the identical type are used at all three positions, especially when using user-defined data types.

The compiler checks for type identity and returns any compile errors. The assignment of function block instances to interface variables is specifically not supported.

MUX selects the kth value from a set of values.

The first value is $K=0$. If K is greater than the number of other inputs (n), IndraLogic transfers the last value (INn).



For runtime optimization, IndraLogic computes only the expression that precedes IN_K . However, IndraLogic computes all branches in simulation mode.

Examples

Result in Var1 is 30.

ST:

```
Var1 := MUX(0,30,40,50,60,70,80);
```

17.3.24 Operator "GT"

This IEC operator is used for the "greater than" function.

Permitted data types of the operands: Any basic data type.

If the first operand is greater than the second operand, the operator provides the result **TRUE**, otherwise **FALSE**.

Examples

Result: FALSE

ST:

```
VAR1 := 20 > 30;
```

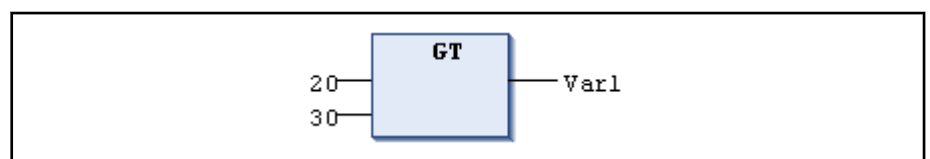
FBD:

Fig. 17-52:

17.3.25 Operator "LT"

This IEC operator is used for the "less than" function.

Permitted data types of the operands: Any basic data type.

If the first operand is less than the second operand, the operator provides the result **TRUE**, otherwise **FALSE**.

Examples

Result: **TRUE**

ST:

```
Var1 := 20 < 30;
```

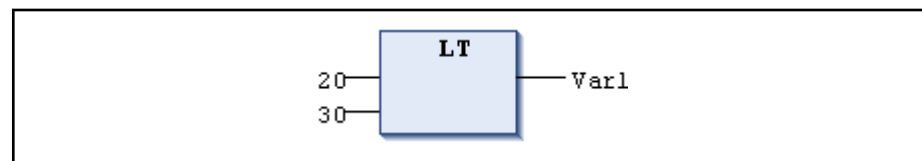


Fig. 17-53:

17.3.26 Operator "LE"

This IEC operator is used for the "less than or equal to" function.

Permitted data types of the operands: Any basic data type.

If the first operand is less than or equal to the second operand, the operator provides the result **TRUE**, otherwise **FALSE**.

Examples

Result in Var1: **TRUE**

ST:

```
Var1 := 20 <= 30;
```

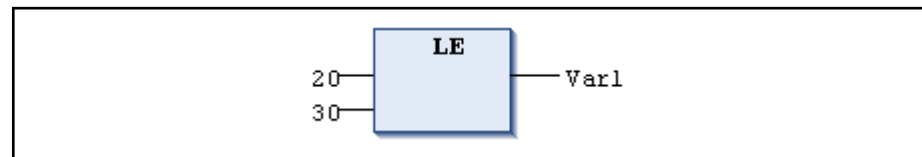


Fig. 17-54:

17.3.27 Operator "GE"

This IEC operator is used for the "greater than or equal to" function.

Permitted data types of the operands: Any basic data type.

If the first operand is greater than or equal to the second operand, the operator returns the result **TRUE**, otherwise **FALSE**.

Examples

Result: **TRUE**

ST:

```
VAR1 := 60 >= 40;
```

FBD:

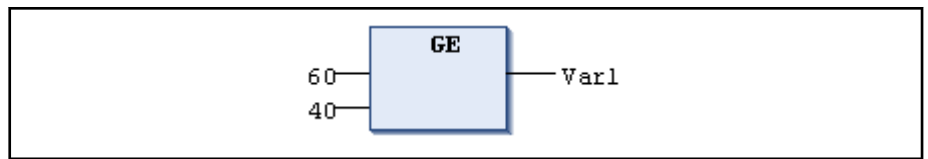


Fig. 17-55:

17.3.28 Operator "EQ"

This IEC operator is used for the "equals" function.

Permitted data types of the operands: Any basic data type, dependent on target system and compiler version: Structural data type.

If the operands are equal, the operator provides the result **TRUE**, otherwise **FALSE**.

Examples

Result: **TRUE**

ST:

```
Var1 := 40 = 40;
```

FBD:

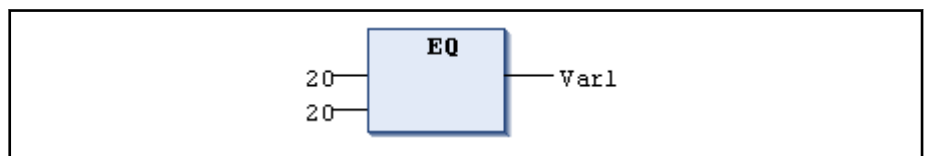


Fig. 17-56:

17.3.29 Operator "NE"

This IEC operator is used for the "does not equal" function.

Permitted data types of the operands: Any basic data type, dependent on target system and compiler version: Structural data type.

If the operands are not equal, the operator provides the result **TRUE**, otherwise **FALSE**.

If the target system supports the data type, operands of type **STRUCT** (structure) can also be compared in the compiler version $\geq 3.5.7.0$ and later.

Example: **IF** (stStruct1 $\neq$ stStruct2) **THEN**...

Examples

Result in Var1 is **FALSE**

ST:

```
Var1 := 40 <> 40;
```

FBD:

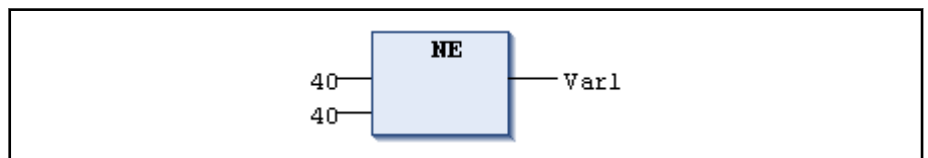


Fig. 17-57:

17.3.30 Operator "ADR"

This operator is an extension of the IEC 61131-3 standard.

ADR provides the address of its argument in a `DWORD`. Apply this address to the manufacturer functions or assign them to a pointer in the project.



In contrast to IndraLogic 1x, the ADR operator can be used with function names, program names, function block names and method names.

Therefore, ADR replaces the `INDEXOF` operator.

Note the following when using function pointers:

Function pointers can be transferred to external libraries, but it is not possible to call a function pointer from inside IndraLogic!

To enable a system call (runtime system), set the respective object property (**Build** tab) for the function object.

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'" on page 619](#).

⚠ CAUTION

Pointer-TO

Do not return `Pointer-TO` variables of functions and methods to the caller or do not assign them to global variables.

Examples:

ST:

```
dwVar := ADR(bVAR);
```

Also refer to

- [chapter 17.5.12 "Data type 'POINTER'" on page 359](#)
- [chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)

17.3.31 Operator "Content operator"

The operator is an extension of the IEC 61131-3 standard.

The operator allows a pointer to be dereferenced. They attach the operator as `^` to the pointer identifier.

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'" on page 619](#).

Example

ST:

```
pt      : POINTER TO INT;
var_int1 : INT;
var_int2 : INT;
pt      := ADR(var_int1);
var_int2 := pt^;
```

17.3.32 Operator "BITADR"

This operator is an extension of the IEC 61131-3 standard.

BITADR provides the bit offset within the segment in a **DWORD**.



The offset depends on whether the option byte addressing is enabled in the target system.

The highest value nibble (4 bits) in this **DWORD** defines the memory range:

Flag: 16#4000 0000

Input: 16#8000 0000

Output: 16#C000 0000

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'" on page 619](#).

Examples

ST:

```
VAR
  var1 AT %IX2.3 : BOOL;
  bitoffset      : DWORD;
END_VAR

bitoffset := BITADR(var1); (* Result if byte
  addressing = TRUE: 16#80000013,
  if byte addressing = FALSE: 16#80000023 *)
```

17.3.33 Operator "CAL"

This IEC operator is used for calling a function block.

In IL, CAL calls the instance of a function block.

Syntax

```
CAL <function block>
(<input variable1> := <value>, <input variableN> := <value>)
```

Example

Call of the **Inst** instance of a function block with assignment of the input variables **Par1** and **Par2** with 0 or TRUE.

ST:

```
CAL Inst(Par1 := 0, Par2 := TRUE) ;
```

17.3.34 Operator 'BOOL_TO'

The IEC operator is used for converting the `BOOL` data type to another data type.

`BOOL_TO_<data type>>`

For number types, the result is 1 if the operand is `TRUE` and 0 if the operand is `FALSE`.

For the `STRING` data type, the result is `TRUE` or `FALSE`.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Example:

ST code	Result
<code>i := BOOL_TO_INT(TRUE);</code>	1
<code>str := BOOL_TO_STRING(TRUE);</code>	TRUE
<code>t := BOOL_TO_TIME(TRUE);</code>	T#1ms
<code>tof := BOOL_TO_TOD(TRUE);</code>	TOD#00:00:00.001
<code>dat := BOOL_TO_DATE(FALSE);</code>	D#1970
<code>dandt := BOOL_TO_DT(TRUE);</code>	DT#1970-01-01-00:00:01

FBD code	Result
	1
	TRUE
	T#1ms
	TOD#00:00:00.001
	D#1970-01-01
	DT#1970-01-01-00:00:01

17.3.35 Operator 'TO_BOOL'

The IEC operator is used for converting from another variable type into a BOOL variable.

<data type>_TO_BOOL

The result is **TRUE** if the operand is not 0. The result is **FALSE** when the operand is equal to 0.

For the **STRING** type, the result is **TRUE** if the operand is **TRUE**. Otherwise, it is **FALSE**.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from **LREAL** to **DINT** is negative.

Example:

Examples

ST code	Result
b := BYTE_TO_BOOL(2#11010101);	TRUE
b := INT_TO_BOOL(0);	FALSE
b := TIME_TO_BOOL(T#5ms);	TRUE
b := STRING_TO_BOOL('TRUE');	TRUE

FBD code	Result
	TRUE
	FALSE
	TRUE
	TRUE

17.3.36 Operator "TO_<xxx>"

The IEC operator is used for converting variables of one data type into another data type, where the input type is not explicitly given ("overflow conversion").

TO_<DataType>

The function depends on the descriptions for each typed conversion.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Examples

ST implementation language:

```
VAR
  iVar : INT;
  bVar : BOOL;
  sVar : STRING;
  rVar : REAL;
END_VAR

iVar := TO_INT(4.22);      (* Result: 4 *)
bVar := TO_BOOL(1);       (* Result: TRUE *)
sVar := TO_STRING(342);   (* Result: '342' *)
rVar := TO_WORD('123');  (* Result: 123 *)
```

17.3.37 Operator "<INT type>_TO_<INT type>"

Conversion of one integer data type into another integer data type.

<INT-DataType>_TO_<INT-DataType>



Information can be lost when converting from larger data types into smaller data types. If the number to be converted exceeds the range limit, IndraLogic ignores the first bytes of the number.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Examples

Result in `si`: 127

ST:

```
si := INT_TO_SINT(4223);
```

When saving the integer 4223 (16#107f in hexadecimal notation) as a `SINT` variable, this variable contains 127 (16#7f in hexadecimal notation).

FBD:

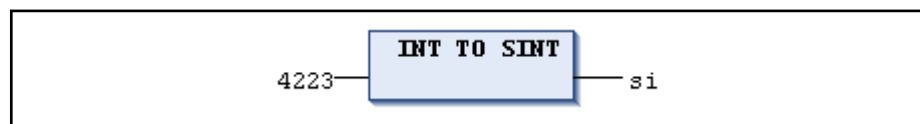


Fig. 17-58:

17.3.38 Operator 'REAL_TO-/ LREAL-TO'

The IEC operator is used for converting the `REAL` and `LREAL` data types into another data type.

`REAL_TO_<DataType>`

`LREAL_TO_<DataType>`

IndraLogic rounds the real value of the operand up or down to an integer value and converts it into the respective type. Exceptions are the types `STRING`, `BOOL`, `REAL`, and `LREAL`.



When converting data types from `REAL` or `LREAL` into `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, or `ULINT` and if the value of the `REAL` or `LREAL` number is outside the value range of the integer, an undefined result is output depending on the target system. An exception is then possible! To obtain code that is independent of the target system, use the application to catch value range overflows. If the `REAL/LREAL` number is within the range, the conversion operates equally on all systems.

When converting the `STRING` type, note that the total length of the decimal number is limited to 16 characters. If the (L)REAL number has more decimal places, the 16th place is rounded and represented rounded in the string. If the `STRING` is defined too short for the number, IndraLogic truncates the string from the right.



When converting larger types to smaller types, information can be lost.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Example:

ST code	Result
<code>i := REAL_TO_INT(1.5);</code>	2
<code>j := REAL_TO_INT(1.4);</code>	1
<code>i := REAL_TO_INT(-1.5);</code>	-2
<code>j := REAL_TO_INT(-1.4);</code>	-1

FBD code	Result
	2

17.3.39 Operator 'TIME_TO/TIME_OF_DAY_TO'

The IEC operator is used for converting the `TIME` and `TIME_OF_DAY` data types into another data type.

`<TIME data type>_TO_<data type>`

Internally, IndraLogic saves the time (in milliseconds) to a `DWORD` (for `TIME_OF_DAY` since 00:00). IndraLogic converts this value.

For the `STRING` data type, the result is the time constant.



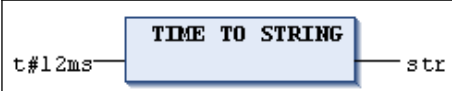
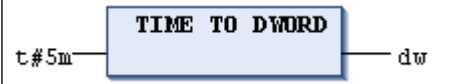
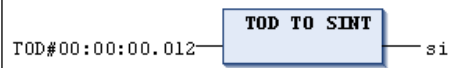
When converting larger types to smaller types, information can be lost.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Example:

ST code	Result
<code>str := TIME_TO_STRING(T#12ms);</code>	T#12ms
<code>dw := TIME_TO_DWORD(T#5m);</code>	300000
<code>si := TOD_TO_SINT(TOD#00:00:00.012);</code>	12

FBD code	Result
	T#12ms
	30000
	12

17.3.40 Operator 'DATE_TO/DT_TO'

The IEC operator is used for converting the `DATE` and `DATE_AND_TIME` data types into another data type.

`<DATE data type>_TO_<data type>`

Internally, IndraLogic saves the date to a `DWORD` (in seconds since January, 1st 1970). IndraLogic converts this value.

For the `STRING` data type, the result is the date constant.



When converting larger types to smaller types, information can be lost.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Example:

ST code	Result
b DATE_TO_BOOL(D#1970-01-01);	FALSE
i DATE_TO_INT(D#1970-01-15);	29952
byt DT_TO_BYTE(DT#1970-01-15-05:05:05);	129
str DT_TO_STRING(DT#1998-02-13-14:20);	'DT#1998-02-13-14:20

FBD code	Result
D#1970-01-01 — DATE TO BOOL — b	FALSE
D#1970-01-15 — DATE TO INT — i	29952
D#1970-01-15-05:05:05 — DATE TO BYTE — i	129
D#1998-02-13-14:20 — DT TO STRING — str	DT#1998-02-13-14:20

17.3.41 Operator 'STRING_TO'

The IEC operator is used for converting the `STRING` data type into another data type.

`STRING_TO_<data type>`

Define the `STRING` operand according to the IEC 61131-3 standard. The value has to be a valid constant (literal) of the target type. This affects any given exponential values, infinite values, prefixes, grouping characters (,) and commas.

Additional characters after the digits of a number are permitted (for example, `23xy`).

Additional characters in front of a number are not permitted.

The operand has to be a valid value of the target data type.



If the data type of the operand does not match the target data type or if the value is outside the range of the target data type, the result output depends on the processor type and is therefore undefined. When converting larger types to smaller types, information can be lost.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Example:

ST code	Result
<code>b := STRING_TO_BOOL('TRUE');</code>	TRUE
<code>w := STRING_TO_WORD('abc34');</code>	0
<code>w := STRING_TO_WORD('34abc');</code>	34
<code>t := STRING_TO_TIME('T#127ms');</code>	T#127ms
<code>r := STRING_TO_REAL('1.234');</code>	1.234
<code>bv := STRING_TO_BYTE('500');</code>	244

FBD code	Result
	TRUE

17.3.42 Operator 'TRUNC'

The IEC operator is used for converting the `REAL` data type into the `DINT` data type. IndraLogic 2G applies only the integer part of the number.



In IndraLogic 1x, the `TRUNC` operator converts from `REAL` into `INT`.

When importing an IndraLogic 1x project, IndraLogic 2G automatically replaces `TRUNC` with `TRUNC_INT`.

If IndraLogic 2G cannot represent the input value with a `DINT` or `INT`, the result of this function is undefined. The behavior of such input values depends on the platform.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Examples

Result in diVar: 1

ST:

```
diVar := TRUNC(1.9); (* Result: 1 *)
```

```
diVar := TRUNC(-1.4); (* Result: -1 *)
```


17.3.43 Operator 'TRUNC_INT'

The IEC operator is used for converting the `REAL` data type into the `INT` data type. IndraLogic 2G uses only the integer part of the number.



`TRUNC_INT` corresponds to the `TRUNC` operator in IndraLogic 1x and when IndraLogic 1x projects are imported, it is automatically used at its place.

Please note the changed function of `TRUNC`.

If IndraLogic 2G cannot represent the input value by a `DINT` or `INT`, the result of this function is undefined. The behavior of such input values depends on the platform.



If the input value of a type conversion operator is outside the value range of the output data type, then the result of the operation is undefined and dependent on the platform. This is the case, for example, if the input value for the conversion from `LREAL` to `DINT` is negative.

Examples

Result in `iVar`: 1

ST:

```
iVar := TRUNC_INT(1.9); (* Result: 1 *)
```

```
iVar := TRUNC_INT(-1.4); (* Result: -1 *)
```

17.3.44 Operator "ABS"

This IEC operator provides the absolute value of a number.

Permitted data types for input variables, output variables and numeric constants: Any numeric basic data type

Examples

Result in `i`: 2

ST:

```
i := ABS(-2);
```

FBD:

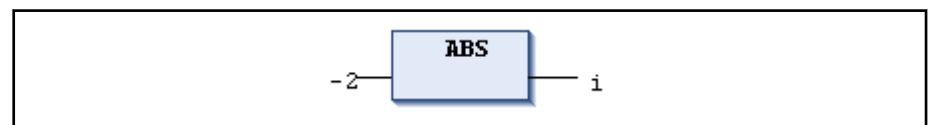


Fig. 17-59:

17.3.45 Operator "SQRT"

This IEC provides the square root of a number.

Permitted data types for input variables: Any numeric basic data type

Permitted data types for output variables: `REAL` or `LREAL`

Examples

Result in `q` is 4.

ST:

```
q := SQRT(16);
```

FBD:



Fig. 17-60:

17.3.46 Operator "LN"

This IEC operator provides the natural logarithm of a number.

Permitted data types for input variables: Any numeric basic data type

Permitted data types for output variables: REAL and LREAL

Examples

Result in q is 3.80666

ST:

```
q := LN(45);
```

FBD:

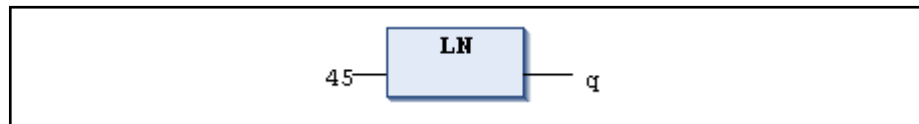


Fig. 17-61:

17.3.47 Operator "LOG"

This IEC operator provides the base-10 logarithm of a number.

The input variable can be any numeric basic data type, but the output variable has to be of type REAL or of type LREAL.

Examples

Result in q: 2.49762

ST:

```
q := LOG(314.5);
```

FBD:

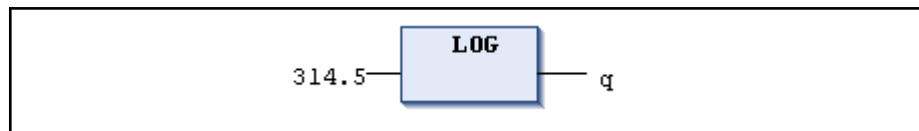


Fig. 17-62:

17.3.48 Operator "EXP"

This IEC operator provides the exponential function.

Permitted data types for input variables: Any numeric basic data type

Permitted data types for output variables: REAL and LREAL

Examples

Result in q is 7.389056099.

ST:

```
q := EXP(2);
```

FBD:

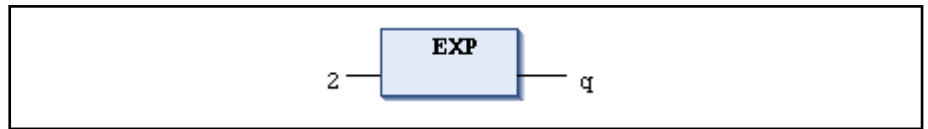


Fig. 17-63:

17.3.49 Operator "EXPT"

The IEC operator raises a number to a higher power with another number and returns the power of the base to the power of the exponent: $\text{power} = \text{base}^{\text{exponent}}$. Base as well as exponent are input values (parameters). The power function is not defined if the base is 0 and the exponent is negative. However, the behavior depends on the platform in this case.

Syntax:

```
EXPT(<Basic>,<Exponent>)
```

Valid data types for the input values: Numerical basic data types (SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LWORD)

Valid data types for the return value: Floating comma types (REAL, LREAL)

Examples:

Result in Var1: 49

Power function with literals

```
Var1 := EXPT(7,2);
```

FBD:

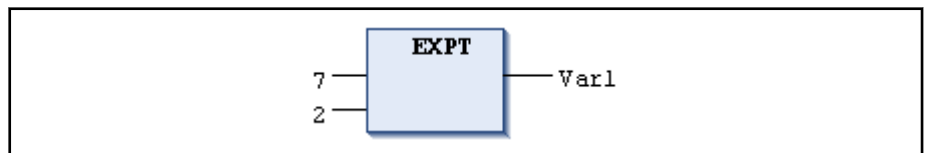


Fig. 17-64:

Return value: Var1 = 49

Power function with variables

Program:

```
PROGRAM PLC_PRG
VAR
  lrPow      : LREAL;
  iBase      : INT := 7;
  iExponent  : INT := 2;
END_VAR

lrPow := EXPT(iBase, iExponent);
```

Return value: lrPow = 49

17.3.50 Operator "SIN"

This IEC operator provides the sine value for a number.

Permitted data types for the input variable measuring the angle in radians:
Any numeric basic data type

Permitted data types for output variable: REAL and LREAL

Examples

Result in q is 0.479426.

ST:

```
q := SIN(0.5);
```

FBD:

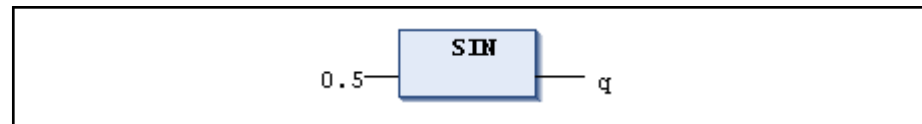


Fig. 17-65:

17.3.51 Operator "COS"

This IEC operator provides the cosine value for a number.

Permitted data types for the input variable measuring the angle in radians:
Any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in q : 0.877583

ST:

```
q := COS(0.5);
```

FBD:

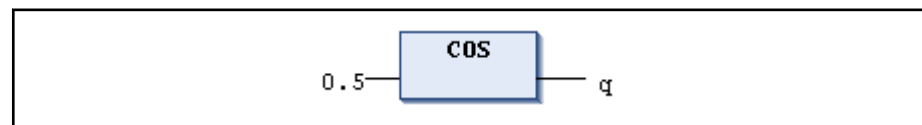


Fig. 17-66:

17.3.52 Operator "TAN"

This IEC operator provides the tangent value of a number.

Permitted data types for the input variable measuring the angle in radians:
Any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in q is 0.546302

ST:

```
q := TAN(0.5);
```

FBD:

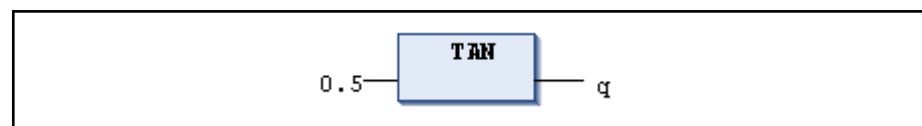


Fig. 17-67:

17.3.53 Operator "ASIN"

This IEC operator provides the arcsine value of a number.

Permitted data types for the input variable: Any numerical basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in `q` is 0.523599

ST:

```
q := ASIN(0.5);
```

FBD:

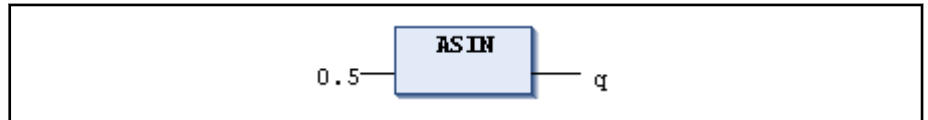


Fig. 17-68:

17.3.54 Operator "ACOS"

This IEC operator obtains the arccosine value of a number. The value is calculated in radians.

Permitted data types for the input variable measuring the angle in radians:
Any numeric basic data type

Permitted data types for output variable: `REAL` and `LREAL`

Examples

Result in `q` is 1.0472

ST:

```
q := ACOS(0.5);
```

FBD:

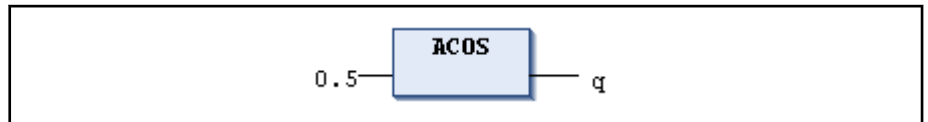


Fig. 17-69:

17.3.55 Operator "ATAN"

This IEC operator provides the arctangent value of a number. The value is calculated in radians.

Permitted data types for the input variable measuring the angle in radians:
Any numeric basic data type

Permitted data types for output variables: `REAL` and `LREAL`

Examples

Result in `q` is 0.463648

ST:

```
q := ATAN(0.5);
```

FBD:



Fig. 17-70:

17.3.56 Operator '___DELETE'

This operator is not described in the standard IEC 61131-3.

The operator releases the memory of instances dynamically generated by the ___NEW operator. The ___DELETE operator does not have a return value and the operand is set to zero after this operation.



In the "Properties" dialog of the application, the "Use dynamic memory allocation" option is selected in the "Application build options" tab.

Syntax:

___DELETE (<Pointer>)



Two tasks should not call ___DELETE simultaneously. Either use a semaphore (SysSemEnter) or comparable method to prevent any concurrent calling of ___DELETE or use ___DELETE in one tasks only (recommended).

An extensive use of ___DELETE function causes higher jitter.

If a pointer points to a function block, the associated method FB_EXIT is called before the pointer is set to 0.

Program:

```
FUNCTION_BLOCK FBDynamic
VAR_INPUT
    in1, in2 : INT;
END_VAR
VAR_OUTPUT
    out : INT;
END_VAR
VAR
    test1 : INT := 1234;
    _inc : INT := 0;
    _dut : POINTER TO DUT;
    neu : BOOL;
END_VAR

out := in1 + in2;
```

Program:

```
METHOD INC : INT
VAR_INPUT
    _inc : INT;
END_VAR

_inc := _inc + 1;
INC := _inc;
```

Program:

```
PLC_PRG (PRG)
VAR
    pFB : POINTER TO FBDynamic;
    bInit: BOOL := TRUE;
    bDelete: BOOL;
    loc : INT;
END_VAR

IF (bInit) THEN
    pFB := ___NEW(FBDynamic);
    bInit := FALSE;
END_IF

IF (pFB <> 0) THEN
    pFB^(in1 := 1, in2 := loc, out => loc);
```

```

    pFB^.INC();
END_IF

IF (bDelete) THEN
    __DELETE(pFB);
END_IF

```

Also refer to:

- [chapter ""Properties" - "Application build options" dialog" on page 707](#)
- [chapter 17.10 ""FB_Init", 'FB_Reinitand 'FB_Exit' methods" on page 427](#)
- [chapter 17.3.58 "Operator '__NEW'" on page 327](#)
- ["Use cases" on page 328](#)

17.3.57 Operator "__ISVALIDREF"

This operator is an extension of the IEC 61131-3 standard.

The operator is used for checking whether a reference refers to a valid value. For a description of use and for an example, refer to the description for the `REFERENCE` data type.

Also refer to

- [chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)

17.3.58 Operator '__NEW'

This operator is an extension of the IEC 61131-3 standard.

The `__NEW` operator reserves dynamic memory to instantiate function blocks, user-defined data types, or arrays of standard types. The operator returns a matching typed pointer.

Prerequisite: In the Properties dialog of the application object, the "Use dynamic memory allocation" option is selected in the "Application build options" tab ([chapter ""Properties" - "Application build options" dialog" on page 707](#)).

Syntax

```

<pointer name> := __NEW( <type> ( , <size> )? );
__DELETE( <pointer name> );

<Type> : <function block> | <data unit type> | <standard data type>

```

The operator generates an instance of the type `<type>` and returns a pointer to this instance. Subsequently, the initialization of the instance is called. If `<type>` is a scalar standard type, the optional operand `<size>` is analyzed. Then the operator generates an array of the type `<standard data type>` and of the size `<size>`. If the attempt to allocate memory fails, then `__NEW` returns the value 0.

Use the operator within the assignment `:=`. Otherwise an error message is displayed.

A function block or a user-defined data type whose instance is created dynamically with `__NEW` uses a fixed memory area. Here it is required that you mark the objects with the pragma `{attribute 'enable_dynamic_creation'}`. It is not required for function blocks that are part of a library.



If you change the data layout of the function block in online mode, then you cannot execute a login with an online change afterwards. The memory area of the function block instance became invalid due to the change. You change the data layout when you add new variables to the function block, delete existing variables, or change the data types of variables.

Use cases *Array (DWORD):*

```

PROGRAM PLC_PRG
VAR
    pdwScalar : POINTER TO DWORD; //Typed pointer
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pdwScalar := __NEW(DWORD, 16);
    // Allocates memory (16 dwords) and assigns them to pointer pdwScalar
END_IF
IF (xDelete) THEN
    __DELETE(pdwScalar); // Frees memory of pointer
END_IF

```

Function block

```

{attribute 'enable_dynamic_creation'}
FUNCTION BLOCK FBComputeGamma
VAR_INPUT
    iAlpha : INT;
    iBeta : INT;
END_VAR
VAR_OUTPUT
    iGamma : INT;
END_VAR
VAR
    iResult : INT;
END_VAR

iGamma := iAlpha + iBeta;

PROGRAM PLC_PRG
VAR
    pComputeGamma : POINTER TO FBComputeGamma; // Typed pointer
    xInit : BOOL := TRUE;
    xDelete : BOOL;
    iResult : INT;
END_VAR

IF (xInit) THEN
    pComputeGamma := __NEW(FBComputeGamma); // Allocates memory
    xInit := FALSE;
END_IF
pComputeGamma^.iAlpha := (pComputeGamma^.iAlpha + 1)MOD 100;
    // Sets first input of pComputeGamma
pComputeGamma^.iBeta := 10; // Sets second input of pComputeGamma
pComputeGamma^(); // Calls the FB pComputeGamma is pointing to
iResult := pComputeGamma^.iGamma; // Reads output of pComputeGamma
IF (xDelete) THEN
    __DELETE(pComputeGamma); // Frees memory
END_IF

```

User-defined data type (DUT):

```

{attribute 'enable_dynamic_creation'}
TYPE ABCDATA :
STRUCT
    iA, iB, iC, iD : INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    pABCDData : POINTER TO ABCDATA; // Typed pointer
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pABCDData := __NEW(ABCDATA); // Allocates memory
    xInit := FALSE;
END_IF
IF (xDelete) THEN

```



```

    __DELETE(pABCDData); // Frees memory
END_IF

```

Array (BYTE)

```

PROGRAM PLC_PRG
VAR
    pbDataAlpha : POINTER TO BYTE;
    pbDataBeta  : POINTER TO BYTE;
    xInit       : BOOL := TRUE;
    xDelete     : BOOL;
    usiCnt      : USINT;
    bTestC      : BYTE;
END_VAR

IF (xInit) THEN
    pbDataAlpha := __NEW(BYTE, 16); // Allocates 16 bytes for pbDataAlpha
    pbDataBeta  := __NEW(BYTE);    // Allocates memory for pbDataBeta
    xInit := FALSE;

    FOR usiCnt := 0 TO 15 DO
        pbDataAlpha[usiCnt] := usiCnt; // Writes to new array
    END FOR
    pbDataBeta^:= 16#FF; // Writes to new data
END_IF

bTestC := pbDataAlpha[12]; // Reads new array by index access

IF (xDelete) THEN // Frees memory
    __DELETE(pbDataAlpha);
    __DELETE(pbDataBeta);
END_IF

```



Do not execute two tasks simultaneously which call the `__NEW` operator. You use either a semaphore (`SysSemEnter`) or a comparable technique to prevent a concurrent call of `__NEW`. A higher jitter results when `__NEW` is applied extensively.

Also refer to:

- [chapter "Properties" - "Application build options" dialog](#) on page 707
- [chapter 18.1.8 "Object "DUTon"](#) page 488
- [chapter "Function block" object \(FUNCTION_BLOCK\)"](#) on page 523
- [chapter "Attribute 'enable_dynamic_creation'"](#) on page 412

17.3.59 Operator “__QUERYINTERFACE”

This operator is an extension of the IEC 61131-3 standard.

During runtime, the operator executes a type conversion of an interface reference into another type. The operator returns a `BOOL` result. `TRUE` means that IndraLogic completed the conversion successfully.

Syntax:

```

__QUERYINTERFACE(<ITF_Source>, <ITF_Dest>);

```

1. Operand: Interface reference or FB interface
2. Operand: Interface reference with required target type

The requirement for the explicit conversion is that both the `ITF_Source` and `ITF_Dest` are derived from `Interface __System.IQueryInterface`. This interface is implicitly available and does not require a library.

Example

```

INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL

```

```

END_METHOD

INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD

INTERFACE ItfDerived2 EXTENDS ItfBase
METHOD mderived2 : BOOL
END_METHOD

FUNCTION_BLOCK FB1 IMPLEMENTS ItfDerived1
METHOD mbase : BOOL
    mbase := TRUE;
END_METHOD
METHOD mderived1 : BOOL
    mderived1 := TRUE;
END_METHOD
END_FUNCTION_BLOCK

FUNCTION_BLOCK FB2 IMPLEMENTS ItfDerived2
METHOD mbase : BOOL
    mbase := FALSE;
END_METHOD
METHOD mderived2 : BOOL
    mderived2 := TRUE;
END_METHOD
END_FUNCTION_BLOCK

PROGRAMM POU
VAR
    inst1 : FB1;
    inst2 : FB2;
    itfbase1 : ItfBase := inst1;
    itfbase2 : ItfBase := inst2;
    itfderived1 : ItfDerived1 := 0;
    itfderived2 : ItfDerived2 := 0;
    xResult1, xResult2, xResult3, xResult4: BOOL;
END_VAR

xResult1 := __QUERYINTERFACE(itfbase1, itfderived1);
// xResult = TRUE, itfderived1 <> 0
// references the instance inst1
xResult2 := __QUERYINTERFACE(itfbase1, itfderived2);
// xResult = FALSE, itfderived2 = 0
xResult3 := __QUERYINTERFACE(itfbase2, itfderived1);
// xResult = FALSE, itfderived1 = 0
xResult4 := __QUERYINTERFACE(itfbase2, itfderived2);
// xResult = TRUE, itfderived2 <> 0
// references the instance inst2

```

17.3.60 Operator "__QUERYPOINTER"

This operator is an extension of the IEC 61131-3 standard.

At runtime, the operator makes it possible to convert the type of an interface reference of a function block to a pointer. The operator returns a `BOOL` result. `TRUE` means that IndraLogic completed the conversion successfully.



For compatibility reasons, the definition of the pointer to be converted has to be an extension of the base interface `__SYSTEM.IQueryInterface`.

Syntax:

```
__QUERYPOINTER (<ITF_Source>,< Pointer_Dest>);
```

The operator receives an interface reference or a FB instance with the required target types as the first operand and a pointer as the second operand.

After processing `__QUERYPOINTER`, `pointer_Dest` contains the pointer to the reference or instance of a function block to which the `ITF_Source` interface currently references.

Pointer `_Dest` is not typed and can be cast to any type. You have to make sure of the type. For example, the interface could offer a method that returns a type code.

17.3.61 Operators “__TRY”, “__CATCH”, “__FINALLY”, “__ENDTRY”

The operators are extended from the IEC 61131-3 standard and they are used for specific exception handling in IEC code.

Syntax

```

__TRY
  <statements_try>
__CATCH(exec)
  <statements_catch>
__FINALLY
  <statements_finally>
__ENDTRY
  <statements_next>

```

When an statement in the `__Try` operator throws an exception, the application does not stop. Instead, the application executes the statements in `__Catch` and starts the exception handling. Then, it executes the statements under `__FINALLY`. The exception handling ends with `__ENDTRY`. Then, the application executes the subsequent statements.

An IEC variable for an exception has the data type `__System.ExceptionCode`.

Example

If the statement in `__TRY` throws an exception, program execution is not stopped. The statement `__CATCH` is processed instead. Therefore, in this example, the `exc` function is executed and then the statements under `__FINALLY`. Then, the application executes the statement under `__ENDTRY`.

Example

```

FUNCTION Tester : UDINT
VAR_INPUT
  _count : UDINT;
END_VAR
VAR_OUTPUT
  _strExceptionText : STRING;
END_VAR
VAR
  exc : __SYSTEM.ExceptionCode;
END_VAR

__TRY
Tester := tryFun(count := count, testcase := g_testcase); (*Diese Anweisung wird getestet.
                                                         Wenn sie eine Exception produziert,
                                                         werden zuerst die Anweisung
                                                         unter __CATCH
                                                         und danach die Anweisung
                                                         unter __FINALLY ausgeführt.*)
__CATCH(exc)
HandleException(exc, _strExceptionText => _strExceptionText);
__FINALLY
GVL.g_count := GVL.g_count + 2;
__ENDTRY

```

Also refer to

- [chapter "Stop execution on handled exceptions" command](#) on page 700

Data type
__System.ExceptionCode

__System.ExceptionCode

```

TYPE ExceptionCode :
(

```

```

RTSEXCPT_UNKNOWN      := 16#FFFFFFF,
RTSEXCPT_NOEXCEPTION  := 16#00000000,
RTSEXCPT_WATCHDOG     := 16#00000010,
RTSEXCPT_HARDWAREWATCHDOG := 16#00000011,
RTSEXCPT_IO_CONFIG_ERROR := 16#00000012,
RTSEXCPT_PROGRAMCHECKSUM := 16#00000013,
RTSEXCPT_FIELDBUS_ERROR := 16#00000014,
RTSEXCPT_IOUPDATE_ERROR := 16#00000015,
RTSEXCPT_CYCLE_TIME_EXCEED := 16#00000016,
RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED := 16#00000017,
RTSEXCPT_UNRESOLVED_EXTREMS := 16#00000018,
RTSEXCPT_DOWNLOAD_REJECTED := 16#00000019,
RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR := 16#0000001A,
RTSEXCPT_LOADBOOTPROJECT_FAILED := 16#0000001B,
RTSEXCPT_OUT_OF_MEMORY := 16#0000001C,
RTSEXCPT_RETAIN_MEMORY_ERROR := 16#0000001D,
RTSEXCPT_BOOTPROJECT_CRASH := 16#0000001E,
RTSEXCPT_BOOTPROJECTTARGETMISMATCH := 16#00000021,
RTSEXCPT_SCHEDULEERROR := 16#00000022,
RTSEXCPT_FILE_CHECKSUM_ERR := 16#00000023,
RTSEXCPT_RETAIN_IDENTITY_MISMATCH := 16#00000024,
RTSEXCPT_IEC_TASK_CONFIG_ERROR := 16#00000025,
RTSEXCPT_APP_TARGET_MISMATCH := 16#00000026,
RTSEXCPT_ILLEGAL_INSTRUCTION := 16#00000050,
RTSEXCPT_ACCESS_VIOLATION := 16#00000051,
RTSEXCPT_PRIV_INSTRUCTION := 16#00000052,
RTSEXCPT_IN_PAGE_ERROR := 16#00000053,
RTSEXCPT_STACK_OVERFLOW := 16#00000054,
RTSEXCPT_INVALID_DISPOSITION := 16#00000055,
RTSEXCPT_INVALID_HANDLE := 16#00000056,
RTSEXCPT_GUARD_PAGE := 16#00000057,
RTSEXCPT_DOUBLE_FAULT := 16#00000058,
RTSEXCPT_INVALID_OPCODE := 16#00000059,
RTSEXCPT_MISALIGNMENT := 16#00000100,
RTSEXCPT_ARRAYBOUNDS := 16#00000101,
RTSEXCPT_DIVIDEBYZERO := 16#00000102,
RTSEXCPT_OVERFLOW := 16#00000103,
RTSEXCPT_NONCONTINUABLE := 16#00000104,
RTSEXCPT_PROCESSORLOAD_WATCHDOG := 16#00000105,
RTSEXCPT_FPU_ERROR := 16#00000150,
RTSEXCPT_FPU_DENORMAL_OPERAND := 16#00000151,
RTSEXCPT_FPU_DIVIDEBYZERO := 16#00000152,
RTSEXCPT_FPU_INEXACT_RESULT := 16#00000153,
RTSEXCPT_FPU_INVALID_OPERATION := 16#00000154,
RTSEXCPT_FPU_OVERFLOW := 16#00000155,
RTSEXCPT_FPU_STACK_CHECK := 16#00000156,
RTSEXCPT_FPU_UNDERFLOW := 16#00000157,
RTSEXCPT_VENDOR_EXCEPTION_BASE := 16#00002000
RTSEXCPT_USER_EXCEPTION_BASE := 16#00010000
) UDINT ;
END_TYPE

```

17.3.62 Operator '__VARINFO'

This operator is an extension of the IEC 61131-3 standard.

The operator yields information about a variable. You can save the information as data structure in a variable of data type `__SYSTEM.VAR_INFO`.

Syntax in the declaration <name of the info variable> : `__SYSTEM.VAR_INFO`; // Data structure for info variable

Syntax upon call <name of the info variable> := `__VARINFO( <variable name> );` // Call of the operator

Example

```

FUNCTION_BLOCK FB_Velocity
VAR_INPUT
    rVelocity: REAL := 1.2;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    infoVelocity: __SYSTEM.VAR_INFO; //Info of Velocity
END_VAR

infoVelocity := __VARINFO(rVelocity); // Gets the info of Velocity locally

```

```

PROGRAM PLC_PRG
VAR
  iCounter : INT := 0; // Counts the calls
  infoCounter : __SYSTEM.VAR_INFO; //Info of Counter
  arrA : ARRAY [1..2, 1..2, 1..2] OF INT := [0, 1, 2, 3, 4, 5, 6, 7]; // Stores the A data
  infoA : __SYSTEM.VAR_INFO; //Info of A
  fbVel : FB_Velocity;
END_VAR

iCounter := iCounter + 1;
infoCounter := __VARINFO(iCounter);
infoA := __VARINFO(arrA);
fbVel();

```

The `iCounter` and `arrA` variables are recognized in the application code. The variable information is saved in the `infoCounter` and `infoA` variables. Moreover, the `FB_Velocity` function block is instantiated.

Data type `SYSTEM.VAR_INFO`

Name	Data type	Initialization	Description
ByteAddress	DWORD	0	Address of variables. Example: 16#072E35EC
ByteOffset	DWORD	0	Offset of the variable address (in bytes). Example: 13936 bytes. Notes: - If the variable is global, then the offset is relative to the beginning of the area. - If the variable is a local variable in a function or method, then the offset is relative to the current stack frame. - If the variable is a local variable in a function block, then the offset is relative to the function block instance.
Area	DINT	0	Memory area number <code>Area</code> in the runtime system. Example: -1 . means that the variable is not global in the memory, but relative to an instance or on the stack. Note: The memory areas depend on the device.
BitNo	INT	0	Number of bits in bytes. Example: 16#00FF bytes Note: If the variable is not an integer data type, the following applies: <code>BitNo = -1 = 16#FFFF</code>
BitSize	INT	0	Memory size of the variable in bits. Example: 16 bits
BitAddress	UDINT	0	Bit address of variables. Prerequisite: The variable is located in the input memory area <code>I</code> , output memory area <code>Q</code> , or marker memory area <code>M</code> . Otherwise, the value is not defined.
TypeClass	TYPE_CLASSES	TYPE_BOOL	Data type class of the variables. Example: <code>TYPE_INT</code> , <code>TYPE_ARRAY</code> Note: For user-defined data types or function block instances, <code>TYPE_USER-DEF</code> is output as the data type class.

Name	Data type	Initialization	Description
TypeName	STRING(79)	' '	Date type name of the variable as STRING(79). Note: For user-defined data types, the function block name or the DUT name is output. Example: 'INT', 'ARRAY'
NumElements	UDINT	0	Number of array elements. Prerequisite: The variable has the data type ARRAY. Example: 8
BaseType-Class	TYPE_CLASSES	TYPE_BOOL	Elementary basic data type of the array elements. Prerequisite: The variable has the data type ARRAY. Example: TYPE_INT for arrA : ARRAY [1..2,1..2,1..2] OF INT;
ElemBit-Size	UDINT	0	Memory size of the array element in bits. Prerequisite: The variable has the data type ARRAY. Example: 16 bits for arrA: ARRAY [1..2,1..2,1..2] OF INT;
MemoryArea	MEMORY_AREA	MEM_MEMORY	Information on the memory area: - MEM_GLOBAL: Global memory area, for example in Area 0 - MEM_LOCAL: Local memory area in Area -1 - MEM_MEMORY: Flag memory area %M, e.g. in 16#10 in Area 1 - MEM_INPUT: Input memory area %I, e.g. in 16#04 in Area 2 - MEM_OUTPUT: Output memory area %Q, e.g. in 16#08 in Area 3 - MEM_RETAIN: Retain memory area, e.g. in 16#20 in Area 0 Example: MEM_GLOBAL Note: The memory area configuration is device-dependent.
Symbol	STRING(39)	' '	Variable name as STRING(39) Example: 'iCounter', 'arrA'
Comment	STRING(79)	' '	Comment of the variable declaration Example: 'Counts the calls' or 'Stores the A data'

Tab. 17-12:

17.3.63 Operator '__CURRENTTASK'

This operator is an extension of the IEC 61131-3 standard.

At runtime, the operator provides information about the IEC task that is currently running.



The operator is supported only on target systems in which the target system setting `memory-layout\max-stack-size` is set to a value > 0.

The operator allows for access to a structure with two variables:

- **TaskIndex:** Zero-based index that identifies the task.

- `pTaskInfo`: Detailed information about the currently running task. It can be assigned to a `POINTER TO Task_Info2` from the library `CmpIecTask`.

The operator cannot be used in the declaration of a POU. This would result in an error message. If the current task cannot be determined, then the `TaskIndex` is -1 and the `pTaskInfo` is zero.

Example

```
//Declaration
VAR
  idx : INT;
  pInfo : POINTER TO Task_Info2;
END_VAR

//Program code

idx := __CURRENTTASK.TaskIndex;
pInfo := __CURRENTTASK.pTaskInfo;
```

17.3.64 Operator '___COMPARE_AND_SWAP'

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used for implementing a semaphore, for example to guarantee exclusive access to a variable written to by different tasks.

Functioning

`___COMPARE_AND_SWAP` gets a pointer to a data type `___XWORD` variable, an old value, and a new value as its input (example: `bMutex := ___COMPARE_AND_SWAP(ADR(dwSynch), dwOld, dwNew);`).

The old and new values can also be data type `___XWORD` variables.

The referenced `___XWORD` variable is compared with the old value and if both are equal, then the new value is written.

The result of the function is `TRUE` when the new value could be written.



The compiler automatically replaces the data type `___XWORD` with `DWORD` on 32-bit systems and `LWORD` on 64-bit systems.

This operation is atomic. Thus, it cannot be interrupted by another task.

Example

The following example shows a typical usage. Exclusive access to a type `STRING` variable, which is addressed via the `pstrOutput` pointer, should be implemented.

The access to a string is not atomic. If multiple tasks write to the same string at the same time, then the contents may be inconsistent.

With the function shown in the example, the same `STRING` variable can be written to different tasks:

```
FUNCTION ExclusiveStringWrite : BOOL
VAR_INPUT
  strToWrite : STRING;
  pstrOutput : POINTER TO STRING;
END_VAR
VAR_STAT
  dwSynch : ___XWORD;
END_VAR
VAR
  bMutex: BOOL;
END_VAR

bMutex:= ___COMPARE_AND_SWAP(ADR(dwSynch), 0, 1);
(* compare with 0 and write 1 as atomic operation *)
IF bMutex THEN
  pstrOutput^ := strToWrite; // bMutex is TRUE if write could be done
  dwSynch := 0;             // Now you can write safely on the string
                             // The ___XWORD variable must be reset.
```

```

ExclusiveStringWrite := TRUE; // Writing was successful
ELSE
ExclusiveStringWrite := FALSE; // Writing was not successful
END_IF

```

17.3.65 Operator '___XADD'

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used for implementing an atomic counter.

Functioning If an integer variable is incremented by means of ordinary addition, for example `iTest := iTest + 1;`, then this operation is not executed atomically. Another access to the variable could take place between reading and writing the variable.

If the counter is incremented in multiple tasks, then the counter result can be less than the number of counting cycles. So if two tasks execute the above code one time and the variable previously had the value 0, then the variable can then have the value 1. This is especially problematic if arrays are being processed in multiple tasks and a unique index is required for the array in each processing cycle.

When the `___XADD` operator is called, it gets a pointer to a type `DINT` variable as the first summand and a type `DINT` value as the second summand. `___XADD` returns the old value of the first summand and in the same step adds the second summand to the first summand.

For example, the function call can be: `diOld := ___XADD(ADR(diVar), deAdd);`

Example The following example shows a typical usage. An array should be populated from two tasks. In the process, all positions in the array should be used and no position should be overwritten.

With the function shown in the example, multiple tasks can populate a Boolean array:

```

FUNCTION WriteToNextArrayPosition : BOOL
VAR_EXTERNAL
    g_diIndex : DINT; // Index and array are globally defined and used by multiple tasks
    g_boolArray : ARRAY [0..1000] OF BOOL;
END_VAR
VAR_INPUT
    bToWrite : BOOL;
END_VAR
VAR
    diIndex : DWORD;
END_VAR

diIndex := ___XADD(ADR(g_diIndex), 1); // Returns a unique index
WriteToNextArrayPosition := FALSE;
IF (diIndex >= 0 AND diIndex <= 1000) THEN
    g_boolArray[diIndex] := bToWrite; //Writes to unique index
    WriteToNextArrayPosition := TRUE; // TRUE: Array was not full yet
END_IF

```

17.3.66 Operator 'TEST_AND_SET'

The multicore operator is an extension of the IEC 61131-3 standard.

The operator can be used to implement a semaphore, for example to guarantee exclusive access to a variable written by different tasks.

Functioning `TEST_AND_SET` gets a type `DWORD` variable as its input. Write access to this variable must be possible. The variable is set to 1 and the previous value is returned as the result.

The operation is atomic, which means that it cannot be interrupted by another task. This also applies to multicore platforms.

For example, the call in the program is `dwOldValue := TEST_AND_SET(dw);`, in which the variables `dwOldValue` and `dw` must be of data type `DWORD`.

Example The following example shows a typical usage. Exclusive access to a type `STRING` variable, which is addressed via the `pstrOutput` pointer, should be implemented. The access to a string is not atomic. If multiple tasks write to the same string at the same time, then the contents may be inconsistent. With the `TEST_AND_SET` function, it is now possible to write the same `STRING` variable in different tasks.

```

FUNCTION ExclusiveStringWrite : BOOL
VAR_INPUT
    strToWrite : STRING;
    pstrOutput : POINTER TO STRING;
END_VAR
VAR_STAT
    dwSynch : DWORD;
END_VAR
VAR
    dwOldValue: DWORD;
END_VAR

dwOldValue := TEST_AND_SET(dwSynch); // Write the 1 and read the old value at the same time
IF dwOldValue = 0 THEN                // 0 means: no other task is currently writing
    pstrOutput^ := strToWrite;         // Now you can write safely on the string
    dwSynch := 0;                     // The DWORD must be reset
    ExclusiveStringWrite := TRUE;      // Writing was successful
ELSE
    ExclusiveStringWrite := FALSE;     // Writing was not successful

```

17.3.67 Operator - Global namespace

This operator is an extension of the IEC 61131-3 standard.

An instance path that begins with a dot `.` always opens a global namespace. If there is a local variable that has the same name `<varname>` as a global variable, address the global variable as `.<varname>`.

17.3.68 Operator - Namespace for Global Variable Lists

This operator is an extension of the IEC 61131-3 standard.

The name of a Global Variable List (GVL) can be used as a namespace identifier for the variables defined in the list. This allows to use variables with the same name in different Global Variable Lists and to still access specific variables uniquely. Place the name of the Global Variable List in front of the variable name and separate them with a dot `..`

Syntax

```
<global variable list name>.<variable>
```

Example

ST:

```
globlist1.varx := globlist2.varx;
```

The global variable lists `globlist1` and `globlist2` contain a variable `varx` each.

IndraLogic copies the `varx` global variable from the `globlist2` list to `varx` in the `globlist1` list.



When referencing a variable declared in several Global Variable Lists without referencing the preceding list name, an error message is output

17.3.69 Operator - Library namespace

This operator is an extension of the IEC 61131-3 standard.

Syntax:

<LibraryNamespace>.<LibraryFunctionBlockIdentifier>

Example:

LIB_A.FB_A

A library function block identifier is extended with the library namespace (as a prefix separated by a dot) for a unique and qualified access to the library function block. The namespace usually matches the library name.

Example:

One library is included in a project and contains the function block `FB_A`. However, the function block with the same name is already available locally in the project. Identify the library function block as `LIB_A.FB_A` to access the library function block, not the local function block.

ST:

```
var1 := FB_A(in := 12); // Call of the project function FB_A
var2 := LIB_A.FB_A(in := 22); // Call of the library function FB_A
```

You can define another identifier for the namespace. To do this, specify a namespace in the project information (library developers: when creating a library project). As an alternative, specify a specific namespace for a library in the library manager in the "Properties" dialog (application developers: when creating an application).

Also refer to:

- [chapter 16 "Creating and managing libraries" on page 219](#)
- [chapter 18.1.14 "Object 'Library manager'" on page 516](#)

17.3.70 Operator - Enumeration namespace

This operator is an extension of the IEC 61131-3 standard.

The TYPE name of an enumeration can be used for an unambiguous access to an enumeration constant. Thus, the same constant names can be used in different enumerations.

The enumeration name precedes the constant name separated by a dot (.).

<EnumerationName>.<ConstantName>

Example

The constant `Blue` is a component of the enumeration `Colors` and the enumeration `Feelings`.

ST:

```
color:= Colors.Blue; //Access to component Blue in enum Colors
feeling:= Feelings.Blue; //Access to component Blue in enum Feelings
```

17.3.71 Operator 'INI'



The INI operator is an operator used in IndraLogic 1.x. In IndraLogic 2G, the "FB_init" method replaces the INI operator.

In projects imported from IndraLogic 1.x, this operator can still be used

Also refer to

- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)
- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)

17.4 Operands

Constants Label constants with the keyword `CONSTANT`. Constants can be declared either locally or globally.

Syntax:

Program:

```
VAR CONSTANT
  <Identifier>:<Type> := <Initialization>;
END_VAR
```

- [chapter 17.4.1 "BOOL constants" on page 339](#)
- [chapter 17.4.2 "TIME constants" on page 339](#)
- [chapter 17.4.3 "DATE constant" on page 341](#)
- [chapter 17.4.4 "DATE_AND_TIME constant" on page 341](#)
- [chapter 17.4.5 "TIME_OF_DAY constant" on page 342](#)
- [chapter 17.4.6 "Number constants" on page 342](#)
- [chapter 17.4.7 "REAL/LREAL constant" on page 343](#)
- [chapter 17.4.8 "String constant" on page 344](#)
- [chapter 17.4.9 "WSTRING constants " on page 345](#)
- [chapter 17.4.10 "Typed literals" on page 346](#)

Variables Variables can be declared either as local or as global in the declaration section of a block or in a Global Variable List.

The position where a variable can be used depends on its data type.

Also refer to

- [chapter 17.4.11 "Access to variables in arrays, structures and function blocks" on page 346](#)
- [chapter 17.4.12 "Bit access to variables " on page 347](#)
- [chapter "'Input wizard' command" on page 587](#)
- [chapter "Command 'Declare variable'" on page 588](#)

- Other**
- [chapter 17.4.13 "Addresses" on page 348](#)
 - [chapter 17.4.14 "Functions" on page 350](#)

17.4.1 BOOL constants

BOOL constants are the truth values `TRUE` (1) and `FALSE` (0).

Also refer to

- [chapter 17.5.1 "'BOOL" data type" on page 352](#)

17.4.2 TIME constants

You can use `TIME` constants to operate the standard timer modules. The constant has a size of 32 bits and therefore a resolution in milliseconds.

In addition, the time constant `LTIME` is available as a time basis for high-resolution timers. The `LTIME` constant has a dimension of 64 bits and therefore a resolution in nanoseconds.

TIME constant Syntax

```
<time keyword> # <length of time>

<time keyword> : TIME | time | T | t
<length of time> : ( <number of days>d )? ( <number of hours>h )? ( <number of minutes>m )?
                  ( <number of seconds>s )? ( <number of milliseconds>ms )? // ( ... )? Optional
```

The order of time units must not be changed. However, not all units have to be declared. It is permitted to write the units in uppercase.

- Time units:*
- d: Days
 - h: Hours
 - m: Minutes
 - s: Seconds
 - ms: Milliseconds

Example of a correct ST assignment of a time constant:

```
VAR
  timLength   : TIME := T#14ms;
  timLength1  : TIME := T#100s12ms; // Overflow in the highest unit is allowed.
  timLength2  : TIME := T#12h34m15s;
  timCompare  : TIME;
  xIsOK       : BOOL;
  timLongest  := T#49D17H2M47S295MS; // 4294967295
END_VAR

IF timLength < T#15MS THEN
  IF timCompare < timLength1 THEN
    xIsOK := TRUE;
  END_IF;
END_IF
```

Incorrect usage:

timIncorrect := t#5m68s;	Overflow at a lower position
timIncorrect1 := 15ms;	Time marker T# missing
timIncorrect2 := t#4ms13d;	Incorrect order of time units

Tab. 17-13:

LTIME constant Syntax

```
<long time keyword> # <length of high resolution time>

<long time keyword> : LTIME | ltime
<length of high resolution time> : ( <length of time> ) ( <number of microseconds>us )?
                                   ( <number of nanoseconds>ns )? // ( ... )? Optional
```

For the LTIME constants, the same units of time as for TIME constants can be used. Moreover, you can specify microseconds and nanoseconds because the specified time is calculated in higher time resolution. LTIME literals are treated internally as data type LWORD and therefore the value resolved in nanoseconds.

- Additional time units:*
- US | us: microseconds
 - NS | ns: nanoseconds

Example of correct use of an ST assignment:

```
PROGRAM PLC_PRG
VAR
  ltimLength := LTIME#1000d15h23m12s34ms2us44ns;
  ltimLength1 := LTIME#3445343m3424732874823ns;
END_VAR
```

Also refer to

- [chapter 17.5.5 "Time data types" on page 353](#)
- [chapter 17.4.5 "TIME_OF_DAY constant" on page 342](#)

17.4.3 DATE constant

Use DATE constants for specifying dates.

Syntax

<date keyword>#<year>-<month>-<day>

<date keyword> : DATE | date | D | d
 <Year> : <1970-2106>
 <month> : <1-12>
 <day> : <1-31>

DATE literals are treated internally as data type DWORD, which corresponds to an upper limit of DATE#2106-2-7.

Example

```
PROGRAM PLC_PRG
VAR
  dateStart      : DATE := D#2018-8-8;
  dateEnd        : DATE := d#2018-8-31;
  dateCompare    : DATE := DATE#1996-05-06;
  xIsDuringTheTime: BOOL;
  dateEarliest   : DATE : D#1970-1-1; // = 0
  dateLatest     : DATE : DATE#2106-2-7; // = 4294967295
END_VAR

IF dateStart < dateCompare THEN
  IF dateCompare < dateEnd THEN
    xIsDuringTheTime := TRUE;
  END_IF;
END_IF
```

Also refer to

- [chapter 17.5.5 "Time data types" on page 353](#)
- [chapter 17.4.4 "DATE_AND_TIME constant" on page 341](#)
- [chapter 17.4.5 "TIME_OF_DAY constant" on page 342](#)

17.4.4 DATE_AND_TIME constant

DATE_AND_TIME constants are a combination of date and time.

Syntax

<date and time keyword>#<date and time value>

<date and time keyword> : DATE_AND_TIME | date_and_time | DT | dt
 <date and time value> : <year>-<month>-<day>-<hour>:<minute>:<second>
 <year> : <1970-2106>
 <month> : <1-12>
 <day> : <1-31>
 <hour> : <0-24>

Reference "Programming"

```
<minute> : <0-59>
<second> : <0-59>
```

DATE_AND_TIME literals are treated internally as data type DWORD. The time is therefore processed in seconds and can take values from January 1, 1970 00:00 and until February 7, 2106 06:28:15.

Example

```
PROGRAM PLC_PRG
VAR
  dtDate : DATE_AND_TIME := DATE_AND_TIME#1996-05-06-15:36:30;
  dtDate1 : DATE_AND_TIME := DT#1972-03-29-00:00:00;
  dtDate2 : DATE_AND_TIME := DT#2018-08-08-13:33:20.5;

  dtEarliest : DATE_AND_TIME := DATE_AND_TIME#1979-1-1-00:00:00; // 0
  dtLatest : DATE_AND_TIME := DATE_AND_TIME#2106-2-7-6:28:15; // 4294967295
END_VAR
```

Also refer to

- [chapter 17.5.5 "Time data types" on page 353](#)
- [chapter 17.4.3 "DATE constant" on page 341](#)
- [chapter 17.4.5 "TIME_OF_DAY constant" on page 342](#)

17.4.5 TIME_OF_DAY constant

Use this constant to define the time of day.

Syntax

```
<time keyword>#<time value>

<time keyword> : TIME_OF_DAY | time_of_day | TOD | tod
<time value> : <hour>:<minute>:<second>
<hour> : <0-23>
<minute> : <0-59>
<second> : <0.000-59.999>
```

You can also specify fractions of a second.

TIME_OF_DAY literals are treated internally as DWORD and the value is resolved in milliseconds.

Example

```
PROGRAM POU
VAR
  todClockTime : TIME_OF_DAY := TIME_OF_DAY#15:36:30.123;
  todEarliest : TIME_OF_DAY := TIME_OF_DAY#0:0:0.000;
  todLatest : TOD := TOD#23:59:59.999;
END_VAR
```

Also refer to

- [chapter 17.5.5 "Time data types" on page 353](#)
- [chapter 17.4.3 "DATE constant" on page 341](#)
- [chapter 17.4.4 "DATE_AND_TIME constant" on page 341](#)

17.4.6 Number constants

Number values can be displayed as binary numbers, octal numbers, decimal numbers and hexadecimal numbers.

If an integer value is not a decimal number, write its base followed by a hash (#) in front of the integer constant. Enter the hexadecimal digit values for the numbers 10 to 15 as usual with the letters A-F.

Use an underscore within a numeric value.

Example:

Examples:

14	Decimal number
2#1001_0011	Binary number
8#67	Octal number
16#A	Hexadecimal number
DINT#16#A1	Typed data type DINT# and basis 16# combined.

This type of numeric value can be BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL or LREAL.



Implicit conversions from "larger" to "smaller" types are not permitted.

A DINT variable cannot simply be used as an INT variable. Use a type conversion function.

Also refer to

- [chapter 17.3 "Operators" on page 294](#)



As numeric constants are generally handled as integer values, they have to specify the constant as floating number in divisions in order not to lose the rest.

Example:

Division 1/10 results in 0,

Division 1.0/10 results in 0.1.

17.4.7 REAL/LREAL constant

You can specify floating-point numbers as REAL and LREAL constant either in decimal notation or exponential notation with mantissa and exponent

The decimal point serves as the decimal separator according to the International System of Units (English).

Syntax of exponential notation

<significand> e | E <exponent>

```
exponent : -44..38 // REAL
exponent : -324..308 // LREAL
```

Example:

REAL literal	
7.4	Decimal number Note: 7,4 with a comma returns a compiler error
1/3.0	Decimal fraction for 0.333333343 Note: In the case of division of integer types, the result remains an integer type. In this case, the value is rounded. For example, 1/3 yields 0 as the result
1.64e+009	Exponential notation
-3.402823e+38	Smallest number
-1E-44	Largest negative number
1.0E-44	Smallest positive number
3.402823e+38	Largest number

LREAL literal	
-1.7976931348623157E+308	Smallest number
-4.94065645841247E-324	Largest negative number
4.94065645841247E-324	Smallest positive number
1.7976931348623157E+308	Largest number

Also refer to [chapter 17.5.3 "Data type 'REAL/ 'LREAL'"](#) on page 352

17.4.8 String constant

A string constant is a character string enclosed in single quotation marks. The characters are coded according to the character set specified in ISO/IEC 8859-1. Therefore, a string constant can include spaces and accented characters (umlauts), as these belong to this character set. This is also referred to as a string literal, or simply a string.

Example: "Hello World!"

When a dollar sign (\$) is in a string constant, the following two characters are interpreted as a hexadecimal code according to the coding in ISO/IEC 8859-1.

The code corresponds to the ASCII code.

Also note [tab. "Special cases"](#) on page 345

String with \$ code	Interpretation
\$<8-bit code>	8-bit code: Two-digit hexadecimal number interpreted according to ISO/IEC 8859-1.
'\$41'	A
'\$9A'	©
'\$40'	@

String with \$ code	Interpretation
'\$0D'	Control character: Line break, corresponds to '\$R'
'\$0A'	Control character: New line, corresponds to '\$L' and '\$N'

Special cases

String with \$ code	Interpretation
'\$L', '\$l'	Control character: Line feed, corresponds to '\$0A'
'\$N', '\$n'	Control character: New line, corresponds to '\$0A'
'\$P', '\$p'	Control character: Page break
'\$R', '\$r'	Control character: Line break, corresponds to '\$0D'
'\$T', '\$t'	Control character: Tab
'\$\$'	Dollar sign: \$
'\$'	Simple quotation mark: '

Example:

Declaration of constants

```
VAR CONSTANT
  constA : STRING := 'Hello Allgäu';
  constB : STRING := 'Hello Allgäu $21'; //
Hello Allgäu!
END_VAR
```

Also refer to:

- [chapter 17.5.9 ""WSTRING" data type" on page 358](#)
- [chapter 17.5.4 ""STRING" data type" on page 353](#)
- [chapter 17.4.9 "WSTRING constants " on page 345](#)

17.4.9 WSTRING constants

A WSTRING is any character string in Unicode (16 bits).

WSTRING constants are enclosed by quotation marks before and after.

Spaces and umlauts can also be entered. They are treated as any other characters.

In character strings, the combination of the dollar sign (\$) followed by four hexadecimal numbers is interpreted as the hexadecimal display of the 16 bit character code. In addition, combinations of characters that start with a dollar sign are interpreted as follows:

Specified combination	Interpretation
'\$<four hex. numbers>'	Hexadecimal display of the 16 bit character code
" "	Space
"\$\$"	Dollar sign
"\$"	Simple quotation marks
"\$L" or "\$l"	Line feed
"\$N" or "\$n"	New line
"\$P" or "\$p"	form feed

"\$R" or "\$r"	Line break
"\$T" or "\$t"	Tabulator

Example:

Valid character strings:

"w1Wüß?"

":-)"

"costs (\$\$)"

"Память"

"蓄能器"

Also refer to:

- [chapter 17.5.9 "'WSTRING' data type" on page 358](#)
- [chapter 17.5.4 "'STRING' data type" on page 353](#)
- [chapter 17.4.8 "String constant" on page 344](#)

17.4.10 Typed literals

With the exception of REAL/LREAL constants (LREAL is always used here), IndraLogic uses the smallest possible data type when calculating with IEC constants.

When using another data type, typed literals (typed constants) can be used without having to declare the constants explicitly.

Provide the constants with a prefix that indicates the type.

Syntax:

`<Type>#<Literal>`

`<Type>` specifies the data type. Possible entries: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. The type has to be written in upper case letters.

`<Literal>` indicates the constant. The input has to match the data types specified under `<Type>`.

Example:

```
var1 := DINT#34;
```

If IndraLogic cannot convert the constant into the target type without data loss, an error message is issued.

Typed constants can be used wherever normal constants can be used.

17.4.11 Access to variables in arrays, structures and function blocks

Notation for the access to

- Two-dimensional array components: `<Array_instance_name>[Index1, Index2]`
- Structural variables: `<Structure_instance_name>.<Variable_name>`
- Function block and program variables: `<Functionblock_instance_name>.<Variable_name>`, `<Program_name>.<Variable_name>`

Also refer to

- [chapter 17.5.14 "Data type 'ARRAY'" on page 363](#)
- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)
- [chapter ""Function block" object \(FUNCTION_BLOCK\)" on page 523](#)
- [chapter ""Program" object \(PROGRAM\)" on page 522](#)

17.4.12 Bit access to variables

If two tasks execute bit access simultaneously, even though the processor cannot perform bit access directly in the memory, then proceed as follows.

You use either a semaphore (`SysSemEnter`) or a comparable technique to prevent a concurrent bit access. However, it is best to execute the bit access within a task.

Syntax for declarations

```
<variable name> : <data type> ( := <initialization> )? ;
```

```
<data type> = BYTE | WORD | DWORD | LWORD | SINT | USINT | INT | DINT | UDINT | LINT | ULINT
```

Syntax for calls

```
<variable name>.<index>
```

With a bit access, a single bit is accessed with read or write permissions within an integer variable. To do this, add a period and an index to the variable identifier that addresses the bit within the variable. Indexing is 0-based. The index is a positive integer starting at 0. You can also specify the index as a constant variable.

Example:

```
PROGRAM PLC_PRG
VAR
  wA : WORD := 16#FF;
  xB : BOOL := 0;
END VAR
wA.2 := xB;
```

In the program, the third bit of the variable `wA` is set to the value of variable `xB`.

```
wA = 2#1111_1111_1111_1011 = 16#FFFB
```

Bit access to a variable with a global constant variable

You can use a global constant as an index to access a bit within an integer variable.

Example: Bit access to an integer variable

Declaration in a global variable list:

```
VAR_GLOBAL CONSTANT
  _enable : INT := 2;
END_VAR
```

The constant `c_usiENABLE` addresses the bit in the integer variable.

Declaration of a POU:

```
VAR
```

```
iX : INT;
END_VAR
```

Call with bit access:

```
iX.c_usiENABLE := TRUE; // The third bit in the variable iX is set TRUE
```

Bit access with BIT data types

With the `BIT` data type, you can combine single bits into a structure and then access them individually. The bit is then addressed with the component name.

Example: Bit access to BIT data types

Declaration of the structure:

```
TYPE ControllerData :
STRUCT
    Status_OperationEnabled : BIT;
    Status_SwitchOnActive : BIT;
    Status_EnableOperation : BIT;
    Status_Error : BIT;
    Status_VoltageEnabled : BIT;
    Status_QuickStop : BIT;
    Status_SwitchOnLocked : BIT;
    Status_Warning : BIT;
END_STRUCT
END_TYPE
```

Declaration of the POU:

```
VAR
    ControllerDrive1 : ControllerData;
END_VAR
```

Bit access:

```
ControllerDrive1.OperationEnabled := TRUE;
```

Also refer to

- [chapter 17.5.2 "Integer data types" on page 352](#)
- [chapter 17.5.10 "Data type 'BIT'" on page 359](#)
- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)

17.4.13 Addresses

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'" on page 619](#).

Syntax

```
%<memory area prefix> ( <size prefix> )? <memory position>
```

```
<memory area prefix> : I | Q | M
```

```
<size prefix> : X | B | W | D
```

```
<memory position> : <number> (.<number>)* //Depends on the target system
```

When specifying an address, use specific character strings to express memory position and size. An address is marked with the percent sign (%), followed by the memory range prefix, the optional size prefix, and the memory range position. The numbering that you use for addressing the memory position depends on the target system.

Supported memory range prefixes

I	Input memory range ("inputs", physical inputs via input drivers, "sensors")
Q	Output memory range ("outputs", physical outputs via output drivers, "actuators")
M	Flag memory range

Supported Site prefixes:

Size Prefix	Data type	Data width
No size prefix	BIT = Bit	Single bit
X	BIT = Bit	Single bit
B	BYTE = Byte	8 bits
W	WORD = Word	16 bits
D	DWORD = Double word	32 bits
L	LWORD = Long word	64 bits

Example:

%QX7.5	Single bit address of the output bit 7.5
%Q7.5	
%IW215	Word address of the input word 215
%QB7	Byte address of the output byte 7
%MD48	Address of a double word at memory position 48 in flag memory
%IW2.5.7.1	Word address of an input word; interpretation dependent on the current control configuration.
VAR wVar AT %IW0 : WORD; END_VAR	Variable declaration with address information of an input word chapter 9.10.2 "AT declaration" on page 125
VAR xActuator AT %QW0 : BOOL; END_VAR	Boolean variable declaration Note: For Boolean variables, one byte is allocated internally if a single bit address is not specified. A change in the value of xActuator affects the range from QX0.0 to QX0.7.
VAR xSensor AT IX7.5 : BOOL; END_VAR	Boolean variable declaration with explicit specification of a single bit address. On access, only the input bit 7.5 is read.

Memory position

The ensures that the address is valid:

To assign a valid address to an application, the required position in the process mapping has to be known (applicable memory range): Input (I), output (Q) or flag memory ranges (M), see above. In addition, define the required size: BIT, BYTE, WORD, DWORD, LWORD(see above: X, B, W, D, L)

The current device configuration and device settings (hardware structure, device description, I/O settings) play a decisive role. Please note the differences in the interpretation of bit addresses for devices with "byte addressing mode" and devices with "word-oriented IEC addressing mode":

For example, the first element of the bit address `%IX5.5` addresses byte 5 in a byte-addressed device and it reaches word 5 in a word-addressed device.

Moreover, addressing with word addresses or byte addresses does not depend on the device type:

Word 5 is always addressed with `%IW5` and byte 5 always with the byte address `%IB5`.

Depending on size and addressing mode, the same address specification can thus be used to address different memory cells.

The following table shows a comparison of byte addressing and word-oriented IEC addressing for `BIT`, `BYTE`, `WORD` and `DWORD` variables. It also shows overlapping memory spaces that are also present for byte addressing (also refer to the example below the table).

Note that the syntax in IEC addressing mode is always word-oriented. Word number before point and bit number after point.

DWords / Words				Bytes	X (bits)					
byte addressing		word oriented IEC addressing			byte addressing			word oriented IEC addressing		
D0	W0	D0	W0	B0	x0.7	...	x0.0	x0.7	...	x0.0
D1	W1			B1	x1.7	...	x1.0	x0.15	...	x0.8
...	W2		W1	B2	...			x1.7	...	x1.0
	W3			B3				x1.15	...	x1.8
	W4	D1	W2	B4						
	...			B5						
			W3	B6						
				B7						
		D2	W4	B8						
		...		...						
		...		...						
		...		...						
D(n-3)		D(n/4)	...							
	W(n-1)		W(n/2)							
				Bn	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8

Fig. 17-71: Comparison of byte-oriented or word-oriented addressing of the address formats *D*, *W*, *B* and *X*

n = byte number; word-oriented, addressing

Example:

Memory range overlapping in the case of the byte addressing mode

D0 contains B0 - B3, W0 contains B0 and B1, W1 contains B1 and B2 and W2 contains B2 and B3 -> To avoid overlapping, W1 or D1, D2, D3 may not be used for addressing!



IndraLogic allocates Boolean values byte-by-byte if a single-bit address was not explicitly specified.

Example: A value change of `varbool1` AT `%QW0` affects the range from `QX0.0` to `QX0.7`.

17.4.14 Functions

In ST, use a function call as operand.

Example:

```
Result := Fct(7) + 3;
```

TIME() function

This function provides the time in milliseconds passed since the system start. The data type is TIME.

Example in ST:

```
sysTime := TIME();
```

Also refer to

- [chapter ""Function" object \(FUNCTION\)" on page 525](#)

17.5 Data types

When programming in IndraLogic, different data types or instances of function blocks can be used.

Thus, one type is assigned to every identifier.

The type specifies the memory space and how these values are interpreted.

The following groups of types are provided:

Standard data types

IndraLogic supports all IEC 61131-3 data types.

[chapter 17.5.1 ""BOOL" data type" on page 352](#)

[chapter 17.5.2 "Integer data types" on page 352](#)

[chapter 17.5.3 "Data type 'REAL/ 'LREAL'" on page 352](#)

[chapter 17.5.4 ""STRING" data type" on page 353](#)

[chapter 17.5.5 "Time data types" on page 353](#)

Extensions to the IEC 61131-3 standard

[chapter 17.5.6 ""UNION" data type" on page 354](#)

[chapter 17.5.7 "Data type 'ANYund 'ANY <type>" on page 355](#)

[chapter 17.5.8 ""LTIME" data type" on page 358](#)

[chapter 17.5.9 ""WSTRING" data type" on page 358](#)

[chapter 17.5.10 "Data type 'BIT'" on page 359](#)

[chapter 17.5.11 "Special data types ""__UXINT"" and ""__XWORD"" on page 359](#)

[chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)

[chapter 17.5.12 "Data type 'POINTER'" on page 359](#)

User-defined data types

There are two options to create user-defined data types:

- Create a DUT object below "Application" or in the "General module folder"
- Data type in a function block in the declaration



Note the recommendations on naming objects

[chapter 17.5.14 "Data type 'ARRAY'" on page 363](#)

[chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)

[chapter 17.5.17 "Enumerations" on page 374](#)

[chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)

[chapter 17.5.12 "Data type 'POINTER'" on page 359](#)

[chapter 17.5.18 "Subrange types" on page 377](#)

[chapter 17.7 "Identifier" on page 419](#)

17.5.1 "BOOL" data type

Data type	Values	Memory size
BOOL	TRUE (1), FALSE (0)	8 bits

Also refer to

- [chapter 17.4.1 "BOOL constants" on page 339](#)

17.5.2 Integer data types

The following integer data types are available in IndraLogic.

Data type	Lower limit	Upper limit	Memory size
BYTE	0	255	8 bits
WORD	0	65535	16 bits
DWORD	0	4294967295	32 bits
LWORD	0	$2^{64}-1$	64 bits
SINT	-128	127	8 bits
USINT	0	255	8 bits
INT	-32768	32767	16 bits
UINT	0	65535	16 bits
DINT	-2147483648	2147483647	32 bits
UDINT	0	4294967295	32 bits
LINT	-2^{63}	$2^{63}-1$	64 bits
ULINT	0	$2^{64}-1$	64 bits



Information can be lost when converting from larger into smaller types.

Also refer to

- [chapter 17.4.6 "Number constants" on page 342](#)

17.5.3 Data type 'REAL' / 'LREAL'

The data types `REAL` and `LREAL` are floating-point types according to IEEE 754.

They are necessary when using decimal numbers and floating-point numbers in decimal notation or exponential notation.

Data type	Smallest value number	Largest value number	Memory size
REAL	1.0E-44	3.402823E+38	32 bits
LREAL	4.94065645841247E-324	1.7976931348623157E+308	64 bits

Example

```

PROGRAM PLC_PRG
VAR
  rMax: REAL      := 3.402823E+38;      // Largest number
  rPosMin : REAL  := 1.0E-44;           // Smallest positive number
  rNegMax: REAL   := -1.0E-44;          // Largest negative number
  rMin: REAL      := -3.402823E+38;      // Smallest number

  lrMax: LREAL    := 1.7976931348623157E+308; // Largest number
  lrPosMin : LREAL := 4.94065645841247E-324; // Smallest positive number
  lrNegMax: LREAL  := -4.94065645841247E-324; // Largest negative number
  lrMin: LREAL     := -1.7976931348623157E+308; // Smallest number
END_VAR

```



If the value of the REAL/LREAL number is outside of the value range of the integer, then an undefined result is yielded from a data type conversion from REAL or LREAL to SINT, USINT, INT, UINT, DINT, UDINT, LINT, or ULINT. The result depends on the target system. An exception error is also possible! To obtain code that is independent of the target system, the application must catch value range violations.

If the REAL/LREAL number is within the value range of the integer data type, then the conversion operates the same way on all systems.

Also refer to

- [chapter 17.4.7 "REAL/LREAL constant" on page 343](#)

17.5.4 "STRING" data type

A variable of the STRING data type can include any character string. The storage space reserved during a declaration refers to characters and is shown in parentheses or brackets. If a size is not defined, IndraLogic assumes 80 characters by default.

As a rule, IndraLogic does not limit the string length. However, the string function processes lengths from 1 to 255 only!

If a variable is initialized with a string that is too long for the data type, IndraLogic truncates the string accordingly from the right.



The storage space required for a STRING variable is always one byte per character plus one additional byte (for example, 81 bytes for a STRING[80] declaration).

Example:

Example of a string declaration with 35 characters:

```
str : STRING(35) := 'This is a String';
```

Also refer to

- [chapter 17.5.9 "WSTRING" data type" on page 358](#)
- [chapter 17.4.8 "String constant" on page 344](#)
- [chapter 17.4.9 "WSTRING constants " on page 345](#)

17.5.5 Time data types

IndraLogic handles the time data types internally like DWORD.

TIME and TIME_OF_DAY are resolved in milliseconds, and DATE_AND_TIME is resolved in seconds.

Data type	Lower limit 0	Upper limit 4294967295	Memory size	Resolution
TIME	T#0d0h0m0s0ms	T#49d17h2m47s295ms	32 bits	Milliseconds
TIME_OF_DAY TOD	TOD#0:0:0.000 (00:00:00:000 o'clock)	TOD#23:59:59.999 (23:59:59:999 o'clock)	32 bits	Milliseconds
DATE	D#1970-1-1 (01/01/1970)	DATE#2106-2-7 (February 07, 2106)	32 bits	Day
DATE_AND_TIME DT	DT#1979-1-1-00:00:00 (01.01.1970, 00:00 o'clock)	DT#2106-2-7-6:28:15 (02/07/2106, 6:28:15)	32 bits	Seconds

Also refer to

- [chapter 17.5.8 "LTIME" data type](#) on page 358
- [chapter 17.4.2 "TIME constants"](#) on page 339
- [chapter 17.4.3 "DATE constant"](#) on page 341
- [chapter 17.4.4 "DATE_AND_TIME constant"](#) on page 341
- [chapter 17.4.5 "TIME_OF_DAY constant"](#) on page 342

17.5.6 "UNION" data type

A UNION is a data structure that usually contains different data types.

The syntax for the union declaration is as follows:

SYNTAX:

```

TYPE <union_name>:
  UNION
    <variable_declaration 1>
    ...
    <variable_declaration n>
  END UNION
END_TYPE

```

<union_name> is a type that is identified in the entire validity range and that can be used as a default data type.

In a union, all components have the same basic address, i.e. they occupy the same memory location.

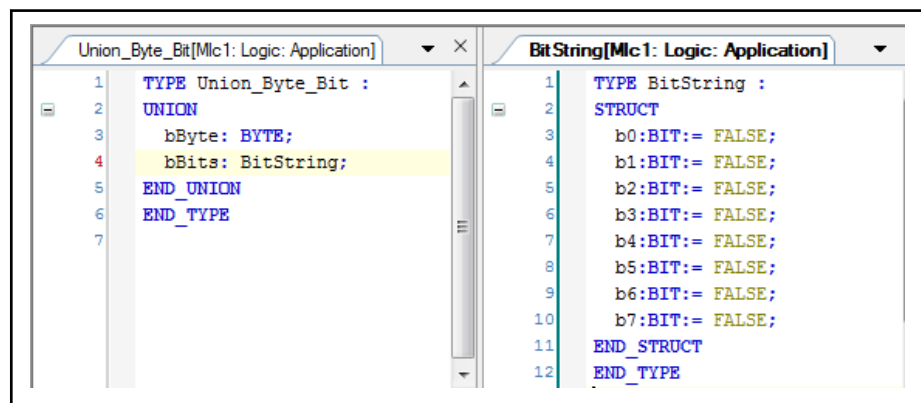


Fig. 17-72: Superimposing a byte and a byte string (structure)

If a value is assigned to the byte (union_01.bByte), Bit string is also affected.

MotionProg[Mlc1: Logic: Application]		
Mlc1.Application.MotionProg		
Expression	Type	Value
union_01	Union_Byte_Bit	
bByte	BYTE	4
bBits	BitString	
b0	BOOL	FALSE
b1	BOOL	FALSE
b2	BOOL	TRUE
b3	BOOL	FALSE
b4	BOOL	FALSE
b5	BOOL	FALSE


```

1 union_01.bByte[4] := 16#4; RETURN

```

Fig. 17-73: Using a union

A complex example includes the superimposition of an enumeration type and a structure.

The user can access the operation mode via enumeration or bit structure.



Fig. 17-74: Data type MB_KINEMATICS_MODE_UNION with MB_KINEMATICS_MODE and MB_KINEMATICS_MODE_BITS

17.5.7 Data type 'ANY' und 'ANY <type>'

When implementing a function or a method, you can declare inputs (VAR_INPUT) as variables with the generic IEC data type (ANY or ANY <type>). As

Reference "Programming"

a result, you can implement calls that have different call parameters for the data type.

At runtime, you can retrieve the transferred value and its type by means of a predefined structure within the programming block for the input variable.

From the compiler version $\geq 3.5.1.0$, the generic IEC data types listed below are supported when using function inputs. The table shows the generic data type that permits elementary data types.

Hierarchy of generic data types			Elementary data types
ANY	ANY_BIT		BYTE, WORD, DWORD, LWORD
	ANY_DATE		DATE_AND_TIME, DT, DATE, TIME_OF_DAY, TOD
	ANY_NUM	ANY_REAL	REAL, LREAL
		ANY_INT	USINT, UINT, UDINT, ULINT SINT, INT, DINT, LINT
	ANY_STRING		STRING, WSTRING

Example

```

FUNCTION AnyBitFunc : BOOL
VAR_INPUT
    _value : ANY_BIT;
END_VAR
FUNCTION AnyDateFunc : BOOL
VAR_INPUT
    _value : ANY_DATE;
END_VAR
FUNCTION AnyFunc : BOOL
VAR_INPUT
    _value : ANY;
END_VAR
FUNCTION AnyIntFunc : BOOL
VAR_INPUT
    _value : ANY_INT;
END_VAR
FUNCTION AnyNumFunc : BOOL
VAR_INPUT
    _value : ANY_NUM;
END_VAR
FUNCTION AnyRealFunc : BOOL
VAR_INPUT
    _value : ANY_REAL;
END_VAR
FUNCTION AnyStringFunc : BOOL
VAR_INPUT
    _value : ANY_STRING;
END_VAR

PROGRAM PLC_PRG
VAR
    xValue   : BOOL           := TRUE;
    byValue  : BYTE          := 16#AB;
    wValue   : WORD           := 16#1234;
    dwValue  : DWORD          := 16#6789ABCD;
    lwValue  : LWORD          := 16#0123456789ABCDEF;
    sValue   : STRING         := 'xyz';
    wsValue  : WSTRING        := "abc";
    dtValue  : DATE_AND_TIME  := DT#2017-02-20-11:07:00;
    dValue   : DATE           := D#2017-02-20;
    todValue : TIME_OF_DAY    := TOD#11:07:00;
    rValue   : REAL           := 42.24;
    lrValue  : LREAL          := 24.42;
    usiValue : USINT          := 12;
    uiValue  : UINT           := 1234;
    udiValue : UDINT          := 12345;
    uliValue : ULINT          := 123456;
    siValue  : SINT           := -12;
    iValue   : INT            := -1234;
    diValue  : DINT           := -12345;
    liValue  : LINT           := -123456;
END_VAR

```

```

AnyFunc(xValue);
AnyFunc(byValue);
AnyFunc(wValue);
AnyFunc(dwValue);
AnyFunc(lwValue);
AnyFunc(sValue);
AnyFunc(wsValue);
AnyFunc(dtValue);
AnyFunc(todValue);
AnyFunc(rValue);
AnyFunc(lrValue);
AnyFunc(usiValue);
AnyFunc(uiValue);
AnyFunc(udiValue);
AnyFunc(uliValue);
AnyFunc(siValue);
AnyFunc(iValue);
AnyFunc(diValue);
AnyFunc(liValue);

AnyBitFunc(byValue);
AnyBitFunc(wValue);
AnyBitFunc(dwValue);
AnyBitFunc(lwValue);

AnyStringFunc(sValue);
AnyStringFunc(wsValue);

AnyDateFunc(dtValue);
AnyDateFunc(dValue);
AnyDateFunc(todValue);

AnyNumFunc(rValue);
AnyNumFunc(lrValue);
AnyNumFunc(usiValue);
AnyNumFunc(uiValue);
AnyNumFunc(udiValue);
AnyNumFunc(uliValue);
AnyNumFunc(siValue);
AnyNumFunc(iValue);
AnyNumFunc(diValue);
AnyNumFunc(liValue);

AnyRealFunc(rValue);
AnyRealFunc(lrValue);

AnyIntFunc(usiValue);
AnyIntFunc(uiValue);
AnyIntFunc(udiValue);
AnyIntFunc(uliValue);
AnyIntFunc(siValue);
AnyIntFunc(iValue);
AnyIntFunc(diValue);
AnyIntFunc(liValue);

```

If the function `ANYBIT_TO_BCD` is called, then a variable of data type `BOOL`, `BYTE`, `WORD`, `DWORD`, or `LWORD` can be passed to the function as a parameter.

Predefined structure

When compiling the code, an `ANY` data type is replaced internally with the following structure:

Program:

```

TYPE AnyType :
STRUCT
  // the type of the actual parameter
  typeclass : __SYSTEM.TYPE_CLASS ;
  // the pointer to the actual parameter
  pvalue : POINTER TO BYTE;
  // the size of the data, to which the pointer points
  diSize : DINT;
END_STRUCT
END_TYPE

```

The actual call parameter assigns the structure elements at runtime.

Example: It is compared whether the two passed variables have the same type and the same value.

Declaration part:

Program:

```

FUNCTION Generic_Compare : BOOL
VAR_INPUT
    any1 : ANY;
    any2 : ANY;
END_VAR
VAR
    pTest : POINTER TO ARRAY [0..100] OF POINTER TO DWORD;
    icount: DINT;
END_VAR

```

Program:

Program:

```

pTest := ADR(any1);
Generic_Compare := FALSE;
IF any1.typeclass <> any2.typeclass THEN
    RETURN;
END_IF
IF any1.diSize <> any2.diSize THEN
    RETURN;
END_IF
// Byte comparison
FOR icount := 0 TO any1.diSize-1 DO
    IF any1.pvalue[iCount] <> any2.pvalue[iCount] THEN
        RETURN;
    END_IF
END_FOR
Generic_Compare := TRUE;
RETURN;
// END_FUNCTION

```

17.5.8 "LTIME" data type

Use the data type LTIME as a time base for high-resolution timers.

A high-resolution timer has a resolution in the range of nanoseconds.

Data type	Lower limit	Upper limit	Memory size
LTIME	0	213503d23h34m33s709ms5 51us615ns	64 bits

Syntax:

LTIME#<TimeDeclaration>

The time declaration can include units of time that apply for the TIME constant and:

- µs: microseconds
- ns: nanoseconds

Example:

```

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns

```

Also refer to

- [chapter 17.5.5 "Time data types" on page 353](#)

17.5.9 "WSTRING" data type

The data type WSTRING is interpreted in Unicode format, in contrast to the STRING (ASCII) data type. The number of characters to be displayed de-

depends on the characters due to this coding in `WSTRING`. A length of 10 for `WSTRING` means that the length of `WSTRING` can assign a maximum of 10 `WORD`. In Unicode, several `WORD` are required for some characters for coding a character so that the number of characters does not have to correspond to the length of the `WSTRING` (here: 10). The data type requires 1 `WORD` per character and additionally 1 `WORD` per memory location. A `STRING` requires only 1 byte. The data type `WSTRING` is terminated with a 0.

Example:

```
wstr : WSTRING := "This is a WString";
```

Also refer to

- [chapter 17.5.4 "'STRING' data type" on page 353](#)
- [chapter 17.4.8 "String constant" on page 344](#)
- [chapter 17.4.9 "WSTRING constants" on page 345](#)

17.5.10 Data type 'BIT'

You can use the `BIT` data type only for single variables in structures or in a function block.

The possible values are `TRUE` (1) and `FALSE` (0).

A `BIT` element requires 1 bit of memory. Therefore, you can reference single bits of a structure by name.

`BIT` elements that are declared in succession are bundled in bytes.

In this way, you can optimize memory use as opposed to `BOOL` types, which reserve 8 bits each.

Bit access requires significantly more time. Therefore, you should use the `BIT` data type only when you need to define data in a predefined format.

Also refer to

- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)

17.5.11 Special data types “__UXINT” and “__XWORD”

IndraLogic supports systems with address registers of a 32-bit width and a 64-bit width.

For making the IEC code as independent as possible from the target system, use the "pseudo" data types `__UXINT` and `__XWORD`.

The compiler checks which target system types are currently used and then converts these data types into the appropriate standard data types.

`__UXINT` is converted into `ULINT` on 64-bit platforms and `UDINT` on 32-bit platforms.

`__XWORD` is converted into `LWORD` on 64-bit platforms and to `DWORD` on 32-bit platforms.

17.5.12 Data type 'POINTER'

Pointers store the addresses of variables, programs, function blocks, methods, and functions while an application program is running. A pointer points to one of the objects mentioned or to a variable with any data type.

Syntax of a pointer declaration:

```
<Identifier>: POINTER TO <DataType | FunctionBlock | Program | Method | Function>;
```

When dereferencing a pointer, the value of the address to which the pointer points is determined. In order to dereference a pointer, append the content operator to the pointer identifier (see `pt^` in the example below).

Using the address operator `ADR`, you can assign the address of a variable to a pointer.

Example:

```
VAR
    pt:POINTER TO INT; (* declaration of pointer pt *)
    var_int1:INT := 5;  (* declaration of variables var_int1 and var_int2 *)
    var_int2:INT;
END_VAR

pt := ADR(var_int1); (* pointer pt is assigned to address of var_int1 *)
var_int2:= pt^;      (* value 5 of var_int1 is assigned to variable var_int2 by dereferencing of pointer pt *)
```



If a pointer to a device input is used, the access (e.g. `pTest := ADR(input);`) is considered to be a write access.

This outputs a compiler warning when code is generated :
 "...invalid assignment target".

If this is required, copy the input value (`input`) to a variable with write access first.

Also refer to

- [chapter 17.3.31 "Operator "Content operator"" on page 312](#)
- [chapter 17.3.30 "Operator "ADR"" on page 311](#)

Function pointers to external functions

IndraLogic supports those function pointers that replace the `INDEXOF` operator. Provide these pointers for external libraries. However, IndraLogic does not provide any option to call a function pointer from within an application in the development system!

The function of the runtime system to register callback functions (system library function) expects the function pointer. Depending on the callback registered, the runtime system implicitly calls the respective function (e.g. for STOP).

To allow such a system call (runtime system), set the corresponding object property in the **Compile** tab.

The `ADR` operator can be used for functions, programs, function blocks and methods. Since functions can change their value after an online change, IndraLogic does not output the address of the function, but the address of a pointer to the function. This address is valid as long as the function exists on the target system.

Also refer to

- [chapter ""Properties" - "Build" dialog" on page 705](#)
- [chapter 17.3.30 "Operator "ADR"" on page 311](#)

Index access to pointers

In IndraLogic, the index access `[]` is allowed to variables of the types `POINTER`, `STRING` or `WSTRING`.

- `pint[i]` returns the basic data type
- The index access to pointers takes place arithmetically:

When using the index access for a variable of type `POINTER TO`, IndraLogic computes the offset by

```
pint[i] = (pint + i * SIZEOF(base type))^.
```

The index access also causes an implicit dereferencing of the pointer. The resulting data type is the basic data type of the pointer.

Observe that `pint[7] ≠ (pint + 7)^!`

- When using the index access for a variable of type `STRING`, the character is provided at the offset of the index expression.

The result is of type `BYTE`. `str[i]` returns the "i-th" character of the string as `SINT` (ASCII).

- When using the index access for a variable of type `WSTRING`, the character is provided at the offset of the index expression.

The result is of type `WORD`. `wstr[i]` returns the "i-th" character of the string as `INT` (Unicode).



The result of the difference between two pointers is of type `DWORD`, even on 64-bit platforms if the pointers are 64-bit pointers.



Note that references can be used, which, in contrast to pointers, can control a value directly.



Note that the memory access of pointers can be monitored at runtime using the implicit monitoring function `CheckPointer`

Also refer to

- [chapter 17.3.30 "Operator "ADR" on page 311](#)
- [chapter 17.5.13 "Data type 'REFERENCE' on page 361](#)
- ["CheckPointer" on page 544](#)

17.5.13 Data type 'REFERENCE'

A `REFERENCE` is also a pointer, however, has the following advantages over the `POINTER`:

- Simple use: The reference does not have to be dereferenced (using `^`) to access contents of referenced objects.
- Improved syntax when transmitting values: If an input is a `REFERENCE TO`, `ADR(value)` does not have to be written in `a(refInput := value)`.
- Type security: In contrast to the pointer, the compiler checks if the basic type is identical when assigning two references.

Declare a reference acc. to the following syntax:

Syntax

```
<Identifier>: REFERENCE TO <DataType>
```

Example

```
A : REFERENCE TO DUT;
B : DUT;
C : DUT;
```

```
A REF= B; // corresponds to A := ADR(B);
A := C; // corresponds to A^ := C;
```



References cannot be declared as follows:

- REFERENCE TO REFERENCE
- Or ARRAY OF REFERENCE
- Or POINTER TO REFERENCE.

Furthermore, a reference cannot be declared to a bit variable.



With compiler versions $\geq$ V3.3.0.0 IndraLogic initializes references (with 0).



When a reference is used for a device entry, the access is a write access (e.g. `ref REF= input;`). This outputs a compiler warning when code is generated: "...invalid assignment target".

If this is required, copy the input value (`input`) to a variable with write access first

Check for valid references

The operator `__ISVALIDREF` can be used to check whether a reference points to a valid value, i.e. to a value not equal to 0.

Syntax:

`<BooleanVariable> := __ISVALIDREF (with REFERENCE TO <DataType> declared identifier);`

`<BooleanVariable>` becomes TRUE if the reference points to a valid value. Otherwise, it becomes FALSE.

Example:

Declaration:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0 : REFERENCE TO INT;
testref : BOOL := FALSE;
```

Implementation:

```
ivar := ivar + 1;
ref_int REF= ivar;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);
(* becomes TRUE, because ref_int points to ivar, which is non-zero *)
testref := __ISVALIDREF(ref_int0);
(* becomes FALSE, because ref_int0 is set to 0 *)
```



In compiler version 3.5.7.40 and later, the implicit monitoring function "CheckPointer" affects REFERENCE variables like the pointer variables ([chapter 18.1.18 "Monitoring the PLC program of the application at runtime" on page 542](#)).

Also refer to:

- [chapter 17.3.57 "Operator "__ISVALIDREF"" on page 327](#)
- [chapter 17.3.30 "Operator "ADR"" on page 311](#)
- [chapter 17.5.12 "Data type 'POINTER'" on page 359](#)

17.5.14 Data type 'ARRAY'

An array is a collection of data elements of the same data type. IndraLogic supports one- and multi-dimensional arrays of fixed or variable length.

Array of fixed length

You can define arrays in the declaration part of a POU or in global variable lists.

Syntax of the declaration of a one-dimensional array

```
<variable_Name> : ARRAY[ <dimension> ] OF <data type> ( := <initialization> )? ;

<dimension> : <lower index bound>..<upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

Syntax of the declaration of a multi-dimensional array

```
<variable_Name> : ARRAY[ <1st dimension> (, <next dimension> )+ ] OF <data type> ( := <initialization> )? ;

<1st dimension> : <1st lower index bound>..<1st upper index bound>
<next dimension> : <next lower index bound>..<next upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...) + : One or more further dimensions
// (...)? : Optional
```

The index limits are integers up to the data type DINT.

Syntax for data access

```
<variable_Name>[ <index of 1st dimension> (, <index of next dimension> )* ]
// (...) * : 0, one or more further dimensions
```



Note the capability of using the implicit monitoring function `CheckBounds ()` to monitor the index limits at runtime.

Example: One-dimensional array of ten integer elements

Declaration

```
VAR
    aiCounter : ARRAY[0..9] OF INT;
END_VAR
```

Lower index limit: 0

Upper index limit: 9

Initialization

```
aiCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70, 80, 90];
```

Data access

```
iLocalVariable := aiCounter[2];
```

The value 20 is assigned to the local variable.

Example: 2-dimensional ARRAY

Declaration

```
VAR
    aiCardGame : ARRAY[1..2, 3..4] OF INT;
END_VAR
```

First dimension 1 to 2

Second dimension 3 to 4

Initialization

```
aiCardGame : ARRAY[1..2, 3..4] OF INT := [2(10), 2(20)];
// Short notation for [10, 10, 20, 20]
```

Data access

```
iLocal_1 := aiCardGame[1, 3]; // Assignment of 10
iLocal_2 := aiCardGame[2, 4]; // Assignment of 20
```

Example: 3-dimensional ARRAY

Declaration

```
VAR
  aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT;
END_VAR
```

First dimension 1 to 2

Second dimension 3 to 4

Third dimension 5 to 6

$2 * 2 * 2 = 8$ array elements

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [10, 20, 30, 40, 50, 60, 70, 80];
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 20
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 30
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 40
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 50
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 60
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 70
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 80
```

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [2(10), 2(20), 2(30), 2(40)];
// Short notation for [10, 10, 20, 20, 30, 30, 40, 40]
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 10
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 20
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 20
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 30
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 30
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 40
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 40
```

Example: 3-dimensional arrays of a customized structure

```

TYPE DATA_A
STRUCT
  iA_1 : INT;
  iA_2 : INT;
  dwA_3 : DWORD;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
  aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A;
END_VAR

```

The array aData_A consists of a total of $3 * 3 * 10 = 90$ array elements of data type DATA_A.

Initialize partially

```

aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A := [(iA_1 := 1, iA_2 := 10, dwA_3 := 16#00FF),
(iA_1 := 2, iA_2 := 20, dwA_3 := 16#FF00), (iA_1 := 3, iA_2 := 30, dwA_3 := 16#FFFF)];

```

In the example, only the first 3 elements are initialized explicitly. Elements to which no initialization value is assigned explicitly are initialized internally with the default value of the basic data type. This initializes the structure components at 0 starting with the element aData_A[2, 1, 1].

Data access

```

iLocal_1 := aData_A[1,1,1].iA_1; // Assignment of 1
dwLocal_2 := aData_A[3,1,1].dwA_3; // Assignment of 16#FFFF

```

Example: Array of a function block

```

FUNCTION_BLOCK FBlock_A
VAR
  iCounter : INT;
END_VAR

PROGRAM PLC_PRG
VAR
  aObject_A : ARRAY[1..4] OF FBlock_A;
END_VAR

```

The array aObject_A consists of 4 elements. Each element instantiates a FBlock_A function block.

Function call

```

aObject_A[2]();

```

Example: Implementation of FB_Something with the method FB_Init method

```

FUNCTION_BLOCK FB_Something
VAR
  _nId : INT;
  _lrIn : LREAL;
END_VAR

METHOD FB_Init : BOOL
VAR_INPUT
  bInitRetains : BOOL;
  bInCopyCode : BOOL;

```

Reference "Programming"

```

        nId : INT;
        lrIn : LREAL;
    END_VAR

    _nId := nId;
    _lrIn := lrIn;

```

The function block FB_Something has a method FB_Init that requires 2 parameters.

Instantiation of the array with initialization

```

PROGRAM PLC_PRG
VAR
    fb_Something_1 : FB_Something(nId := 11, lrIn := 33.44);
    a_Something : ARRAY[0..1, 0..1] OF FB_Something[(nId := 12, lrIn := 11.22),
        (nId := 13, lrIn := 22.33), (nId := 14, lrIn := 33.55), (nId := 15, lrIn := 11.22)];
END_VAR

```

"Array of arrays"

The declaration of an "array of arrays" is an alternative syntax for multidimensional arrays. A collection of elements is nested instead of dimensioning the elements. The nesting depth is unlimited.

Syntax for declaration

```

<variable name> : ARRAY[<first>] ( OF ARRAY[<next>] )+ OF <data type> ( := <initialization> )? ;

<first> : <first lower index bound>..<first upper index bound>
<next> : <lower index bound>..<upper index bound> // one or more arrays
<data type> : elementary data types | user defined data types | function block types
// (...) + : One or more further arrays
// (...) ? : Optional

```

Syntax for data access

```

<variable name>[<index of first array>] ( [<index of next array>] )+ ;
// (...) * : 0, one or more further arrays

```

Example

```

PROGRAM PLC_PRG
VAR
    aiPoints : ARRAY[1..2,1..3] OF INT := [1,2,3,4,5,6];
    ai2Boxes : ARRAY[1..2] OF ARRAY[1..3] OF INT := [ [1, 2, 3], [4, 5, 6]];
    ai3Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF INT :=
        [ [ [1, 2, 3, 4], [5, 6, 7, 8] ], [9, 10, 11, 12] ],
        [ [13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24] ] ];
    ai4Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF ARRAY[1..5] OF INT;
END_VAR

aiPoints[1, 2] := 1200;
ai2Boxes[1][2] := 1200;

```

The variables aiPoints and ai2Boxes collect the same data elements, but the spelling in the declaration and the data access differ.

Array of variable length

In function blocks, functions, or methods, you can declare arrays of variable length in the VAR_IN_OUT declaration section. The LOWER_BOUND and UPPER_BOUND operators are provided for determining the index limits of the actual used array at runtime.

Syntax of the declaration of a one-dimensional array of variable length

```
<variable name> : ARRAY[*] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

Syntax of the declaration of a multi-dimensional array of variable length

```
<variable name> : ARRAY[* (, * )+ ] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...) + : One or more further dimensions
// (...) ? : Optional
```

Syntax of operator for limit index calculation

```
LOWER_BOUND( <variable name> , <dimension number> )
UPPER_BOUND( <variable name> , <dimension number> )
```

Example:

The SUM function adds the integer values of the array elements and returns the calculated sum as a result. The sum is calculated across all array elements available at runtime. As the actual number of array elements will only be known at runtime, the local variable is declared as a one-dimensional array of variable length.

```
FUNCTION SUM: INT;
VAR_IN_OUT
  aiData : ARRAY[*] OF INT;
END_VAR
VAR
  diCounter, diResult : DINT;
END_VAR

diResult := 0;
FOR diCounter := LOWER_BOUND(aiData, 1) TO UPPER_BOUND(aiData, 1) DO
  // Calculates the length of the current array
  diResult := diResult + A[i];
END_FOR;
SUM := diResult;
```

Also refer to

- ["Checkbounds" on page 544](#)
- [chapter 9.2.3 "Declaring arrays" on page 86](#)

17.5.15 Data type '__VECTOR'

Vector operations are supported natively only on 64-bit processors and offer a performance advantage only on these processors. The data sheet of the controller provides information about the processor used on the controller.

Currently, vector operations on the x86/64-bit platforms with SSE2 and ARM64 with NEON are supported natively.

On all other platforms, vector operations are translated into individual statements. For example, vector addition is then executed with multiple single addition operations.

The command set extensions of the processors are SIMD extensions. SIMD (Single Instruction, Multiple Data) describes a computer architecture in which multiple data sets of the same type are processed simultaneously in parallel and therefore faster with one command call. In vector operations, for example, 4 pairs of numbers can then be added at the same time.

Syntax

```
<variable name> : __VECTOR[ <vector size> ] OF <element type> ( := <initialization> )? ;
<vector size> : 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
<element type> : REAL | LREAL
// (...)? : Optional
```

A vector data type is a floating point number array of up to eight elements. The operators `__vc<operator name>` are available for this data type. You can use these to implement vector operations without additional function calls.

Syntax for index access

```
<variable name>[ <index> ]
<index> : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
```

When indexing a vector variable, you can access a single element of the vector. The index starts at 0 and end at `<vector size> -1`.

Example

```
PROGRAM PLC_PRG
VAR
    vcA : __VECTOR[3] OF REAL;
END_VAR

vcA[0] := 1.1;
vcA[1] := 2.2;
vcA[2] := 3.3;
```

Determining the optimal vector size



Use the optimal vector size depending on your target system as the vector size in order to program the most efficient code possible.

For target systems whose computer architecture is generally suitable for vector processing, we do not recommend using vectors of arbitrary size. There is an optimal vector size depending on the type of data processing of the processor. Vectors that are declared with this array size are processed as quickly as possible. Vectors that are declared as a larger array do not have an advantage in speed. Vectors that are declared as smaller arrays do not take full advantage of the processor's capabilities.

You can query the optimal size at runtime. For information, go to the constants `Constants.vcOptimalREAL` (for vectors with `REAL` elements) and `Constants.vcOptimalLREAL` (for vectors with `LREAL` elements). The data type of the constants is `INT`. If a constant returns the value 1 as the optimal value, the accelerated vector processing is not supported for the target system.

Operator '`__VCADD`' The operator calculates the sum of two vectors.

Syntax

```
<vector variable> := <1st vector operand> __VCADD <2nd vector operand>;
```

Example: Addition

```
FUNCTION_BLOCK FB_ADD
VAR
    vcA      : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB      : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
    vcResult : __VECTOR[3] OF REAL;
END_VAR

vcResult := vcA __VCADD vcB;
```

Operator '__VCSUB'

The operator calculates the difference between two vectors.

Syntax

```
<vector variable> := <vector minuend> __VCADD <vector subtrahend>;
```

Example: Subtraction

```
FUNCTION_BLOCK FB_SUB
VAR
    vcA      : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB      : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
    vcResult0 : __VECTOR[3] OF REAL;
    vcResult1 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCSUB vcB;
vcResult1 := vcB __VCSUB vcA;
```

Operator '__VCMUL'

The operator calculates the product of two vectors or a scalar (floating-point number) and a vector.

Syntax

```
<vector variable> := <1st vector operand> __VCMUL <2nd vector operand> |
                    <scalar operand> __VCMUL <vector operand> |
                    <vector operand> __VCMUL <scalar operand>;
```

Example: Multiplication

```
FUNCTION_BLOCK FB_MUL
VAR
    rScalar   : REAL := 1.1;
    vcA       : __VECTOR[3] OF REAL;
    vcB       : __VECTOR[3] OF REAL;
    vcResult0 : __VECTOR[3] OF REAL;
    vcResult1 : __VECTOR[3] OF REAL;
    vcResult2 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCMUL vcB;
vcResult1 := rScalar __VCMUL vcB;
vcResult2 := vcA __VCMUL 3.3;
```

Operator '__VCDIV'

The operator calculates the quotient of two vectors or a vector and a scalar.

Syntax

```
<vector variable> := <vector dividend> __VCDIV <vector divisor> |
                    <vector dividend> __VCMUL <scalar divisor>;
```

Example: Division

```

FUNCTION_BLOCK FB_DIV
VAR
  iScalar    : INT := 3;
  rScalar    : REAL := 1.5;
  vcA        : VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
  vcB        : VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
  vcResult0  : VECTOR[3] OF REAL;
  vcResult1  : VECTOR[3] OF REAL;
  vcResult2  : VECTOR[3] OF REAL;
END_VAR

vcResult0 := vcA __VCDIV vcB;
// ERROR CODE vcResult1 := rScalar __VCDIV vcB;
// ERROR CODE vcResult1 := iScalar __VCDIV vcB;
// ERROR CODE vcResult1 := 3.3 __VCDIV vcB;
vcResult2 := vcA __VCDIV 1.5;
vcResult2 := vcA __VCDIV iScalar;
vcResult2 := vcA __VCDIV rScalar;

```

Operator '___VCDOT'

The operator calculates the dot product (scalar product) of two vectors.

Syntax

```
<skalar variable> := <1st vector operand> ___VCDOT <2nd vector operand> ;
```

Example: Dot product

```

FUNCTION_BLOCK FB_DOT
VAR
  rResult : REAL;
  vcA     : VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
  vcB     : VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
END_VAR

rResult := vcA ___VCDOT vcB; // = 18

```

Operator '___VCSQRT'

The operator calculates the square root of each element in the vector.

Syntax

```
<vector variable> := ___VCSQRT( <vector operand> );
```

Example: Square root

```

FUNCTION_BLOCK FB_SQRT
VAR
  vcA      : VECTOR[3] OF REAL := __VCSET_REAL(4, 9, 16);
  vcResult0 : VECTOR[3] OF REAL;
END_VAR

vcResult0 := ___VCSQRT(vcA);

```

Operator '___VCMAX'

The operator calculates the maximum vector of two vectors. The maximum is determined element by element.

Syntax

```
<vector variable> := ___VCMAX(<1st vector operand>, <2nd vector operand>);
```

Example: Maximum vector

```
FUNCTION_BLOCK FB_MAX
```

```

VAR
    vcA      : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB      : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 6);
    vcResult0 : __VECTOR[3] OF REAL;
END_VAR

vcResult0 := __VCMAX(vcA, vcB);

```

Operator '__VCSET_REAL'

The operator sets all elements of a vector in a statement. The elements have the REAL data type.

Syntax

```

<vector variable> := __VCSET_REAL( <first literal>, ( <next literal> )+ );
( ... )+ // number of elements have to match

```

Example:

```

FUNCTION_BLOCK FB_SET
VAR
    vcA : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
    vcB : __VECTOR[3] OF REAL := __VCSET_REAL(1, 2, 3);
END_VAR

vcA := __VCSET_REAL(4, 4, 4);
vcB := __VCSET_REAL(1.1, 2.2, 3.3);

```

Operator '__VCSET_LREAL'

The operator sets all elements of a vector at once in a statement. The elements have the LREAL data type.

They can be used wherever variables are valid, such as in assignments in implementations or as parameters in function calls.

Syntax

```

<vector variable> := __VCSET_LREAL( <first literal>, ( <next literal> )+ );
( ... )+ // number of elements have to match

```

Example:

```

FUNCTION_BLOCK FB_SET
VAR
    vc1A : __VECTOR[3] OF LREAL := __VCSET_LREAL(3, 3, 3);
    vc1B : __VECTOR[3] OF LREAL := __VCSET_LREAL(1, 2, 3);
END_VAR

vc1A := __VCSET_LREAL(-1.7976931348623158E+308, 0.0, 1.7976931348623158E+308);
vc1B := __VCSET_LREAL(-1.7976931348623158E+308, 0.0, 1.7976931348623158E+308);

```

Operator '__VCLOAD_REAL'

The operator interprets any arbitrary memory area as a vector, for example to connect vector variables to an existing code.

The operator requires 2 parameters. The first parameter indicates the number of vector elements. The second parameter is a pointer to the REAL data.

Syntax

```

<vector variable> := __VCLOAD_REAL( <vector size>, <pointer to data of type REAL> );
<vector size> : 2 | 3 | 4 | 5 | 6 | 7 | 8

```

Example: Vectorization

```

FUNCTION_BLOCK FB_LOAD

```

Reference "Programming"

```

VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  rData0 : REAL      := 1.234;
  rData1 : REAL      := 5.678;
  rData2 : REAL      := 9.123;
  pData  : POINTER TO REAL := ADR(rData0);
  vcA    : __VECTOR[3] OF REAL := __VCSET_REAL(3, 3, 3);
END_VAR

vcA := __VCLOAD_REAL(3, pData);

```

Operator '___VCLOAD_LREAL'

The operator interprets any arbitrary memory area as a vector, for example to connect vector variables to an existing code.

The operator requires 2 parameters. The first parameter indicates the number of vector elements. The second parameter is a pointer to the LREAL data.

Syntax

```

<vector variable> := __VCLOAD_LREAL( <vector size>, <pointer to data of type LREAL> );
<number of vector elements> : 2 | 3 | 4 | 5 | 6 | 7 | 8

```

Example: Vectorization

```

FUNCTION BLOCK FB_LOAD
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  lrData0 : LREAL      := -1.7976931348623158E+308;
  lrData1 : LREAL      := 1.6E+308;
  lrData2 : LREAL      := 1.7E+308;
  lrData3 : LREAL      := -1.6E+308;
  plData  : POINTER TO LREAL := ADR(lrData0);
  vc1A    : __VECTOR[4] OF LREAL := __VCSET_LREAL(4, 4, 4, 4);
END_VAR

vc1A := __VCLOAD_LREAL(4, plData);

```

Operator '___VCSTORE'

The operator saves or copies the contents of the vector to the specified memory address.

The number and the types of elements are automatically applied from the vector variables.

Syntax

```

<vector variable> := __VCSTORE( <pointer to data>, <vector variable> );

```

Example: Saving

```

FUNCTION BLOCK FB_STORE
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  rData0 : REAL      := 3;
  rData1 : REAL      := 3;
  rData2 : REAL      := 3;
  pData  : POINTER TO REAL := ADR(rData0);
  lrData0 : LREAL      := 4;
  lrData1 : LREAL      := 4;
  lrData2 : LREAL      := 4;
  lrData3 : LREAL      := 4;

```

```

plData : POINTER TO LREAL := ADR(lrData0);

vcA : __VECTOR[3] OF REAL := __VCSET_REAL( 1.234, 5.678, 9.123);
vc1A : __VECTOR[4] OF LREAL := __VCSET_LREAL(-1.7976931348623158E+308,
                                             1.6E+308, 1.7E+308, -1.6E+308);
END_VAR

__VCSTORE(pData, vcA);
__VCSTORE(plData, vc1A);

```

Also refer to:

- [chapter 9.2.3 "Declaring arrays" on page 86](#)
- ["Checkbounds" on page 544](#)

17.5.16 Data type 'STRUCT'

Create a structure in a project with a **DUT** object by clicking on **Add object**.

The structural declaration starts with the keywords `TYPE` and `STRUCT` and ends with the keywords `END_STRUCT` and `END_TYPE`.

Syntax for structural declaration:

```

TYPE <Structure_name>:
STRUCT
    <Variable_declaration 1>
    ...
    <Variable_declaration n>
END_STRUCT
END_TYPE

```

<structure name> is a type that IndraLogic detects in the entire project and that can be used as a standard data type.

Nested structures can also be used. The only restriction is that it is not permitted to assign addresses to variables (AT declaration is not permitted!).

Example of a structural definition named polygon line

```

TYPE polygonline:
STRUCT
    start:ARRAY [1..2] OF INT;
    point1:ARRAY [1..2] OF INT;
    point2:ARRAY [1..2] OF INT;
    point3:ARRAY [1..2] OF INT;
    point4:ARRAY [1..2] OF INT;
    end:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE

```

Also refer to

- [chapter 18.1.8 "Object "DUT" on page 488](#)
- [chapter 17.3.28 "Operator "EQ"" on page 311](#)

Initializing structures

Example

```
pPoly_1 : polygonline := ( start:=[3,3], point1:=[5,2], point2 := [7,3], point
```

There may be no initializations with variables. For an example of initializing an array of a structure, go to the help site on the `ARRAY` data type.

Also refer to

- ["Example: 3-dimensional ARRAY" on page 364](#)

Access to structural components

Access a structural component as per the following syntax:

<structure name>.<component name>

Thus, `poly_1.start` can be used to access the `start` component of the `polygon` line structure in the example above.

Also refer to

- ["Example: 3-dimensional ARRAY" on page 364](#)

Bit access in structures

`Bit` is a special data type that can only be defined in structures.

It reserves one bit of memory and allows to address individual bits of a structure using a name.

```
TYPE <Structure_name>:
STRUCT
  <Bit_name bit1> : BIT;
  <Bit_name bit2> : BIT;
  <Bit_name bit3> : BIT;
  ...
  <Bit_name bitn> : BIT;
END STRUCT
END_TYPE
```

Access the `BIT` structure component using the following syntax:

`<structure name>.<bit name>`



No references or pointers to `BIT` variables can be used.
`BIT` variables may not be used in data arrays.

Also refer to

- [chapter 17.5.10 "Data type 'BIT'" on page 359](#)

17.5.17 Enumerations

An enumeration is a user-defined data type composed of a series of comma-separated components (enumeration values) for declaring user-defined variables. Moreover, you can use the enumeration components like constants whose identifier `<enumeration name>.<component name>` is known globally in the project.

Declare an enumeration in a DUT object already created in the project via **Add**.

Declaration Syntax

```
( {attribute 'strict'} )? // Pragma optional but recommended
TYPE <enumeration name> ;
(
  <first component deklaration>,
  (<component declaration> ,)+
  <last component deklaration>
)( <basic data type> )? ( := <default variable initialization> )?
;
END_TYPE

( ... )? : Optional
<component declaration> : <component name> ( := <component initialization> )?
<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD
<variable initialization> ; <one of the component names>
```

In an enumeration declaration, at least two components are usually declared. However, you can declare as many as you want. Every single component can be assigned its own initialization. Enumerations automatically have the basic data type `INT`, but you can specify another basic data type. Moreover, you can specify a component in the declaration with which an enumeration variable is then initialized.

The pragma {attribute 'strict'} causes a strict type test to be performed as described below.

Example

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE COLOR_BASIC :
(
  yellow,
  green,
  blue,
  black
) // Basic data type is INT, default initialization for all COLOR_BASIC variables is yellow
;
END_TYPE
```

Enumeration with explicit basic data type

Extensions to the IEC 61131-3 standard.

The basic data type for an enumeration declaration is `INT` by default. However, you can also declare enumerations that are based explicitly on another integer data type.

Syntax

```
<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD
```

Example: Enumeration with basic data type DWORD

```
TYPE COLOR :
(
  white := 16#FFFFFF00,
  yellow := 16#FFFF0000,
  green := 16#00FF0000,
  blue := 16#0000FF00,
  black := 16#88000000
) DWORD := black
; // Basic data type is DWORD, default initialization for all COLOR variables is black
END_TYPE
```

Strict programming rules



The pragma {attribute 'strict'} is added automatically in the first line when declaring an enumeration.

The strict programming rules are activated when adding the pragma {attribute 'strict'} automatically.

The following code is considered a compiler error:

- Arithmetic operations with enumeration components.
For example, an enumeration variable cannot be used as a counter variable in a `FOR` loop.
- Assignment of a constant value, which does not correspond to an enumeration value, to an enumeration component.
- Assignment of a non-constant variable, which has another data type as the enumeration, to an enumeration component.

Arithmetic operations can lead to undeclared values being assigned to enumeration components.

A better programming style uses `SWITCH/Case` statements to process component values.

Declaration and initialization of enumeration variables

Syntax

Program:

```
<variable name> : <enumeration name> ( :=<initialization> )? ;
```

For a declaration of an enumeration variable with user-defined data type <EnumerationName>, this can be initialized with an enumeration component.

Example

```
PROGRAM PLC_PRG
VAR
    colorCar: COLOR;
    colorTaxi : COLOR := COLOR.yellow;
END_VAR
```

The variable `colorCar` is initialized with `COLOR.black`. That is the default initialization for all enumeration variables of type `COLOR` and defined this way in the type declaration. The variable `colorTaxi` has its own initialization.

If no initializations are specified, then the initialization value is 0.

Example

```
PROGRAM PLC_PRG
VAR
    cbFlower : COLOR_BASIC;
    cbTree: COLOR_BASIC := COLOR_BASIC.green;
END_VAR
```

The variable `cbFlower` is initialized with `COLOR_BASIC.yellow`. That is the default initialization for all enumeration variables of type `COLOR_BASIC`. Because the enumeration declaration does not specify a component for initialization, the system automatically initializes with the component that has the value 0. This is usually the first of the enumeration components. However, it can also be another component that is not in the first position but explicitly initialized with 0.

The variable `cbTree` has an explicit initialization.

If no value is specified for both the type and the variable, then the following rule applies: If an enumeration contains a value for 0, then this value is the default initialization, and if not, then the first component in the list.

Example: Initialization with the 0 component

```
TYPE ENUM :
(
    e1 := 2,
    e2 := 0,
    e3
)
;
END_TYPE

PROGRAM PLC_PRG
VAR
    e : ENUM;
END_VAR
```

The variable `e` is initialized with `ENUM.e2`.

Example: Initialization with the first component

```
TYPE ENUM2 :
(
    e1 := 3,
```

```

        e2 := 1,
        e3
    )
;
END_TYPE

PROGRAM PLC_PRG
VAR
    e2 : ENUM2;
END_VAR

```

The variable `e2` is initialized with `ENUM.e1`.

Unique access to enumeration components

Extensions to the IEC 61131-3 standard.

The enumeration components can also be used as constant variables with the identifier `<enumeration name>.<component name>`. Enumeration components are recognized globally in the project and access to them is unique. Therefore, a component name can be used in different enumerations.

Example: Component blue

```

PROGRAM PLC_PRG
VAR
    cbFlower : COLOR_BASIC;
    colorCar : COLOR;
END_VAR

(* unambiguous identifiers although the component names are identical *)
cbFlower := COLOR_BASIC.blue;
colorCar := COLOR.blue;

(* invalid code *)
cbFlower := blue;
colorCar := blue;

```

Also refer to [chapter 17.3.70 "Operator - Enumeration namespace"](#) on page 338

17.5.18 Subrange types

A subrange type is a data type whose value range is a subset of a basic type.

Syntax for the declaration:

Program:

```
<Name> : <Inttype> (<ug>..<og>);
```

<Name>	Valid IEC identifier
<Inttype>	Data type of the subrange (SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LINT, ULINT, LWORD).
<ug>	Lower limit of the range: Constant that has to be compatible with the basic data type. The lower limit itself belongs to this range.
<og>	Upper limit of the range: Constant that has to be compatible with the basic data type. The upper limit is also included in this range.

Examples:

```

VAR
    i : INT (-4095..4095);

```

```

    ui : UINT (0..10000);
END_VAR

```

When assigning a value to a subrange type in the declaration or implementation section that outside this range (example: `i:=5000`), IndraLogic issues an error message.



Note that it is possible in runtime mode to monitor the range limits of a subrange type by using the implicit monitoring functions `CheckRangeSigned` and `CheckRangeUnsigned`.

Also refer to

- ["CheckRangeUnsigned" on page 544](#)
- ["CheckRangeSigned" on page 544](#)

17.6 Pragmas

Pragma statements affect the properties of one or multiple variables with regard to the compilation or pre-compilation.

Various categories of pragmas are available.

17.6.1 Message pragmas

Message pragmas are used to force the display of messages in the "Message" window during the compilation.

Insertion location: Separate or already existing line in the text editor of a POU.

Pragma	Message type
{text <'textstring'>}	Text: Output of <textstring>.
{info <'textstring'>}	 Information : Output of <textstring>.
{warning <'textstring'>}	 Warning: Output of <textstring>. Unlike the attribute pragma "obsolete", define the warning locally for the current position.
{error <'textstring'>}	 Error: Output of <textstring>.

Tab. 17-14: Four types of message pragmas



In the IndraLogic message window, jump to the source position of the message using the commands **Next message** and **Previous message** from a message of the category **Information**, **Warning** and **Error**.

This means that you reach the position where the pragma is added to the source code.

Example

```

VAR
    ivar : INT; {info 'TODO: should get another name'}
    bvar : BOOL;
    arrTest : ARRAY [0..10] OF INT;
    i:INT;
END_VAR

```

```

arrTest[i] := arrTest[i]+1;
ivar:=ivar+1;

{warning 'This is a warning'}
{text 'Part xy has been compiled completely'}

```

Output in the message window:

Description	Project	Object	Position
----- Build started: Application: Res.App2 -----			
typify code ...			
! Compile time before typification: 0 ms			
! Compile time after typification: 15 ms			
! TODO: should get another name!	TS pragma	NewPOU	Line 3 (Decl)
! This is a warning	TS pragma	NewPOU	Line 7 (Impl)
Part xy has been compiled completely	TS pragma	NewPOU	Line 10 (Impl)
Compile complete -- 1 errors, 2 warnings			

Fig. 17-75:

Also refer to

- [chapter 17.6.3 "Conditional pragmas" on page 413](#)

17.6.2 Attribute pragmas

Attribute pragmas affect the compilation and the pre-compilation.

IndraLogic supports a series of pre-defined attribute pragmas. In addition, user-defined pragmas can be used. These pragmas can be queried using conditional pragmas before the compilation of the project.

Attributes are defined in the declaration.

Exception: For the objects "Action" and "Transition" without an own declaration, attributes can be defined at the beginning of the implementation.



When defining own attributes, ensure that these are unambiguous.

This is for example the case if the attribute name has a prefix.

The OEM can be used as vendor prefix.

User-defined attributes

User-defined attributes are any application-defined or user-defined attributes that can be applied to POU's, actions, data type definitions and variables. A user-defined attribute can be queried using conditional pragmas before the compilation of the application.



User-defined attributes can be queried with conditional pragmas with the operator `hasattribute`.

For more detailed information and examples, refer to "Conditional pragmas".

Syntax:

```
{attribute "attribute"}
```

Example for POU's and actions

Attribute "vision" for function fun1

```

{attribute 'vision'}
FUNCTION fun1 : INT

```

Example for variables

```
VAR_INPUT
  i: INT;
END_VAR
```

Attribute "DoCount" for variable ivar:

```
PROGRAM PlcProg
VAR
  {attribute 'DoCount'};
  ivar:INT;
  bvar:BOOL;
END_VAR
```

Example for data types

Attribute "aType" for data type DUT_1:

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
  a:INT;
  b:BOOL;
END_STRUCT
END_TYPE
```

Also refer to

- [chapter 17.6.3 "Conditional pragmas" on page 413](#)

Attribute "call_after_global_init_slot"

VAR_INPUT declarations in functions or methods using the attribute result in compilation errors!

Reason: In this case, input variables are not known at the time of the call that occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute are called after the global initialization. Define the calling sequence using the attribute value.

Syntax:

```
{attribute "call_after_global_init_slot" := "<slot>"}
```

<Slot>: Integral value that defines the ranking in the order of the calls: The lower the value, the earlier the call takes place. If several function blocks have the same ranking for the attribute, the order of their calls remains indefinite.

Insertion location: First line above the declaration section of functions and programs

If a method is provided with the attribute, IndraLogic determines all instances of the respective function block and calls all instances in the specified slot. In this case, the order of the instances among themselves cannot be influenced.

Also refer to

- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)
- [chapter "Attribute "global_init_slot" on page 388](#)

Attribute 'call_after_init'

VAR_INPUT declarations in functions or methods using the attribute result in compilation errors!

Reason: In this case, input variables are not known at the time of the call that occurs implicitly during the online change.

The effect of this pragma is that a method is called implicitly after the initialization of a function block instance. Due to performance, add the attribute

both to the function block and to the method in its own first line above the declaration section.

Syntax:

```
{attribute "call_after_init"}
```

Call: First line above the declaration section of the method and the function block.

IndraLogic calls the method after the method FB_init and after the variable values of an initialization expression became valid in the instance declaration.

This functionality is supported from the compiler version 3.4.1.0.

Example

Definition

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <functionblock definition>

{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

The definition implements the following declaration in the subsequent code processing for example:

```
inst : FB := (in1 := 99);
```

Code processing

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

Thus, the FB_AfterInit method can respond on the user-defined initialization.

Also refer to

- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)

Attribute 'call_after_online_change_slot'



VAR_INPUT declarations in functions or methods using the attribute result in compilation errors!

Reason: In this case, input variables are not known at the time of the call that occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute are called after an online change. Define the calling sequence using the attribute <slot>.

Syntax:

```
{attribute "call_after_online_change_slot" := "<slot>"}
```

<Slot>: Integral value that defines the ranking in the order of the calls: The lower the value, the earlier the call takes place. If several function blocks have the same priority for the attribute, the order of their calls is not defined.

Call: First line above the declaration section of functions, methods and programs.

If a method is provided with the attribute, IndraLogic determines all instances of the respective function block. IndraLogic calls all instances in the specified

slot. In this case, the order of the instances among themselves cannot be influenced.



Since the application cannot run during the online change, each code executed in this situation can cause a jitter. Therefore, keep the extent of the executive code as small as possible.

Also refer to

- [chapter "Command 'Online change'" on page 619](#)

Attribute 'call_before_global_exit_slot'



VAR_INPUT declarations in functions or methods using the attribute result in compilation errors!

Reason: In this case, input variables are not known at the time of the call that occurs implicitly during the online change.

The effect of this pragma is that all functions and programs containing this attribute in a dedicated first line of their declaration are called before the GlobalExit.

The GlobalExit takes place before a new download or during a reset. Function blocks with the FB_Exit method are affected. The calling sequence is defined by the attribute value.

Syntax:

```
{attribute "call_before_global_exit_slot" := "<slot>"}
```

Insertion location: First line above the declaration section of functions and programs.

<Slot>: Integral value that defines the ranking in the order of the calls: The lower the value, the earlier the call takes place. If several function blocks have the same ranking for the attribute, the order of their calls remains indefinite.

If a method is provided with the attribute, the method is called for all instances of the respective function block. IndraLogic calls all instances in the specified slot. In this case, the order of the instances among themselves cannot be influenced.

Also refer to:

- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)
- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)

Attribute "call_on_type_change"

With this pragma, you can mark a method of a function block A that should be called when the data type changes for one or more function blocks B, C, etc. that are referenced by A. The referencing can be defined by a pointer variable or a REFERENCE variable.

Syntax

```
{attribute 'call_on_type_change' := '<name of the first referenced function block>|  
                                     <name of the second referenced function block>|  
                                     <name of the nth referenced function block>'}
```

Insertion location: Line above the first line in the method declaration.

Example *Function blocks with references*

```
FUNCTION_BLOCK FB_A
...
VAR
  var_pt: POINTER TO FB_B;
  var_ref: REFERENCE TO FB_C;
END
```

Method for reaction to a type change in the references FB_B and FB_C

```
{attribute 'call_on_type_change' := 'FB_B, FB_C'}
METHOD METH_react_on_type_change : INT
VAR
```

Attribute "const_replaced", attribute "const_non_replaced"

The effect of the pragma {attribute "const_replaced"} is that the compiler option **Replace constants** is explicitly activated for all global constants containing this attribute in the constant declaration.

Insert the pragma {attribute "const_non_replaced"} accordingly to explicitly disable the compiler option **Replace constants**. This can be desirable, for example, to provide a constant in the symbol configuration.

The option **Replace constants** in the dialog **Project settings**, category **Compiler options** is preset for the entire project.

Syntax:

```
{attribute "const_replaced"}
{attribute "const_non_replaced"}
```

Insertion location: Line above the declaration line of the global variables.

Example

The constants iTestCon and bTestCon are available in the symbol configuration, as the option **Replace constants** is disabled.

```
VAR_GLOBAL CONSTANT
  {attribute 'const_non_replaced'}
  iTestCon : INT := 12;
  {attribute 'const_non_replaced'}
  bTestCon : BOOL := TRUE;
  rTestCon : REAL := 1.5;
END_VAR

VAR_GLOBAL
  iTestVar : INT := 12;
  bTestVar : BOOL := TRUE;
END_VAR
```

Also refer to

- [chapter "Project settings" - "Compiler options" dialog](#) on page 715
- [chapter 10.2 "Symbol configuration"](#) on page 156

Attribute "dataflow"

With this pragma, control the data flow in the processing of function blocks in the FBD/LD/IL editor. The attribute specifies at which input or output of a function block the forwarding is applied to the next or previous function block.

Only one input and one output is provided with the attribute in the declaration of a function block.

Syntax:

```
{attribute "dataflow"}
```

Insertion location: Line above the line with the declaration of the corresponding variables.

In case of function blocks without the attribute "dataflow", IndraLogic determines the data flow as follows: First of all, the connection is established between an output and an input of same data type. The highest input variable or output variable of the function blocks is always used. If there are no variables of a corresponding data type, IndraLogic connects the uppermost output to the uppermost input of the neighboring function blocks.

Example

The connection between FB and the preceding function block is established via the input variable `i1`. The connection between FB and the following function block is established via the output variable `outRes1`.

```
FUNCTION_BLOCK FB
VAR_INPUT
    r1 : REAL;
    {attribute 'dataflow'}
    i1 : INT;
    i2 : INT;
    r2 : REAL;
END_VAR

VAR_OUTPUT
    {attribute 'dataflow'}
    outRes1 : REAL;
    out1 : INT;
    g1 : INT;
    g2 : REAL;
END_VAR
```

Also refer to

- [chapter "Programming in Function Block Diagram \(FBD\)" on page 92](#)

Attribute "displaymode"

With this pragma, define the display mode of an individual variable.

This definition overwrites the global setting for the representation of the monitoring variable via **Debug ► Display mode**.

Syntax:

```
{attribute 'displaymode':=<displaymode>}
```

The following definitions are possible

- Binary format
 - {attribute 'displaymode':='bin'}
 - {attribute 'displaymode':='binary'}
- Decimal format
 - {attribute 'displaymode':='dec'}
 - {attribute 'displaymode':='decimal'}
- Hexadecimal format
 - {attribute 'displaymode':='hex'}
 - {attribute 'displaymode':='hexadecimal'}

Insertion location: Line above the line with the declaration of the corresponding variables.

Example

```
VAR
{attribute 'displaymode':='hex'}
  dwVar1: DWORD;
END_VAR
```

Also refer to

- [chapter "Command 'Display mode'" on page 636](#)

Attribute 'estimated-stack-usage'

The pragma provides an estimated value for the stack size requirement.

Methods with recursive calls cannot pass a stack check because stack usage cannot be determined. As a result, a warning is issued. To prevent this warning, you can give the method an estimated value (in bytes) for the stack size requirement. Then the method passes the stack check successfully.

Syntax

```
{attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}
```

Example

```
{attribute 'estimated-stack-usage' := '127'} // 127 bytes
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
END_VAR
```

Insertion location: First line above the declaration section of the method.

The "Method call" chapter includes an example that uses this pragma, see [chapter 9.18.4 "Method call" on page 150](#).

Recursive method call

Within an implementation, a method can call itself: Either directly by using the `THIS` pointer by using a local variable for the assigned function block.



Use recursions mainly for processing recursive data types such as linked lists.

In general, we recommend to be careful when using recursion, as unexpectedly deep recursions can cause stack overflow and machine downtime.

Example: Calculation of the factorial

The following program `PLC_PRG` calculates the factorial of a number in the `FB_Factorial` function block in a different way, each in its own method.

- Method `m_Iterative`: Iterative
- Method `m_Pragmaed`: Recursive with warning suppression
- Method `m_Recursive`: Recursive
- Method `m_Temp`: Temporary with warning suppression

A warning is issued for the `m_Recursive` method only

Reference "Programming"

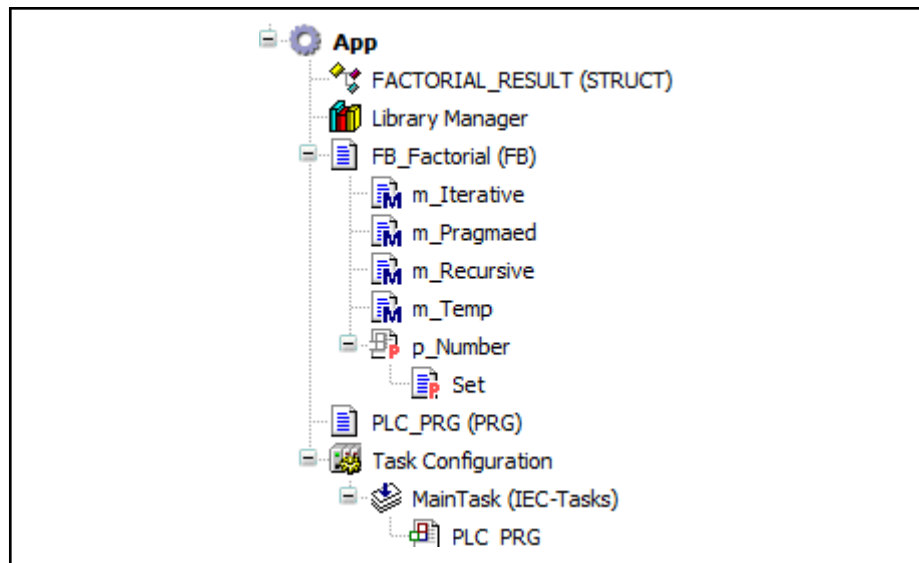


Fig. 17-76:

Program:

```
// Contains the data of the factorial calculation of uiNumber
TYPE FACTORIAL_RESULT :
STRUCT
    uiNumber : UINT;
    udiIterative : UDINT;
    udiRecursive : UDINT;
    udiPragmaed : UDINT;
    udiTemp : UDINT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
    fb_Factorial_A : FB_Factorial;
    factorial_A : FACTORIAL_RESULT := (uiNumber := 9, udiIterative := 0, udiRecursive := 0, udiPragmaed := 0 );
END_VAR
fb_Factorial_A.p_Number := factorial_A.uiNumber;
factorial_A.udiIterative := fb_Factorial_A.m_Iterative();
factorial_A.udiRecursive := fb_Factorial_A.m_Recursive(uiN := factorial_A.uiNumber);
factorial_A.udiPragmaed := fb_Factorial_A.m_Pragmaed(uiN := factorial_A.uiNumber);
factorial_A.udiTemp := fb_Factorial_A.m_Temp(uiN := factorial_A.uiNumber);

//Factorial calculation in different ways
FUNCTION BLOCK FB_Factorial
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    uiN : UINT;
    udiIterative : UDINT;
    udiPragmaed : UDINT;
    udiRecursive : UDINT;
END_VAR

// Iterative calculation
METHOD PUBLIC m_Iterative : UDINT
VAR
    uiCnt : UINT;
END_VAR
m_Iterative := 1;
IF uiN > 1 THEN
    FOR uiCnt := 1 TO uiN DO
        m_Iterative := m_Iterative * uiCnt;
    END FOR;
    RETURN;
ELSE
    RETURN;
END_IF;
```

```
//Recursive calculation with suppressed warning
{attribute 'estimated-stack-usage' := '99'}
METHOD PUBLIC m_Pragmaed : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
VAR
END_VAR
m_Pragmaed := 1;
IF uiN > 1 THEN
    m_Pragmaed := uiN * THIS^.m_Pragmaed(uiN := (uiN - 1));
    RETURN;
ELSE
    RETURN;
END_IF;

//Recursive calculation
METHOD PUBLIC m_Recursive : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
VAR
END_VAR
m_Recursive := 1;
IF uiN > 1 THEN
    m_Recursive := uiN * THIS^.m_Recursive(uiN := (uiN - 1));
    RETURN;
ELSE
    RETURN;
END_IF;

// Called by temporary FB instance
{attribute 'estimated-stack-usage' := '99'}
METHOD PUBLIC m_Temp : UDINT
VAR_INPUT
    uiN : UINT;
END_VAR
VAR
    fb_Temp : FB_Factorial;
END_VAR
m_Temp := 1;
IF uiN > 1 THEN
    m_Temp := uiN * fb_Temp.m_Temp(uiN := (uiN - 1));
    RETURN;
ELSE
    RETURN;
END_IF;

PROPERTY p_Number : UINT
uiN := p_Number; //Setter method
```

Only the `m_Recursive` issues a warning when the program is executed.

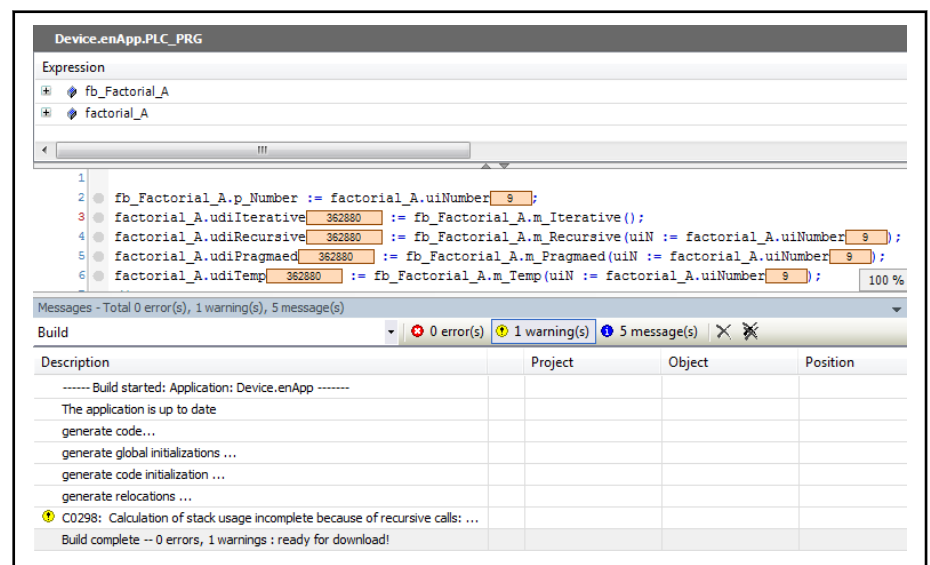


Fig. 17-77:

Also refer to:

- [chapter 9.18.4 "Method call" on page 150](#)
- [chapter 9.18 "Object-oriented programming" on page 145](#)
- [chapter "Method" object " on page 532](#)
- [chapter "Object 'Property'" on page 537](#)

Attribute "ExpandFully"

The effect of this pragma is that the components of an array used as an input variable for referenced visualizations become visible in the "Properties" dialog of the visualization.

Syntax:

```
{attribute "ExpandFully"}
```

Insertion location: The line above the line with the declaration of the array.

Example

Visualization **visu** is to be inserted into a frame in the visualization **visu_main**.

arr is defined as an input variable in the interface editor of **visu** and will thus be available later for assignments in the "Properties" dialog of the frames in **visu_main**. To get the individual components of **arr** as well in this "Properties" dialog, insert the attribute "ExpandFully" directly in front of **arr** in the interface editor of **visu**.

Declaration in the interface editor of visu:

```
VAR_INPUT
{attribute 'ExpandFully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```

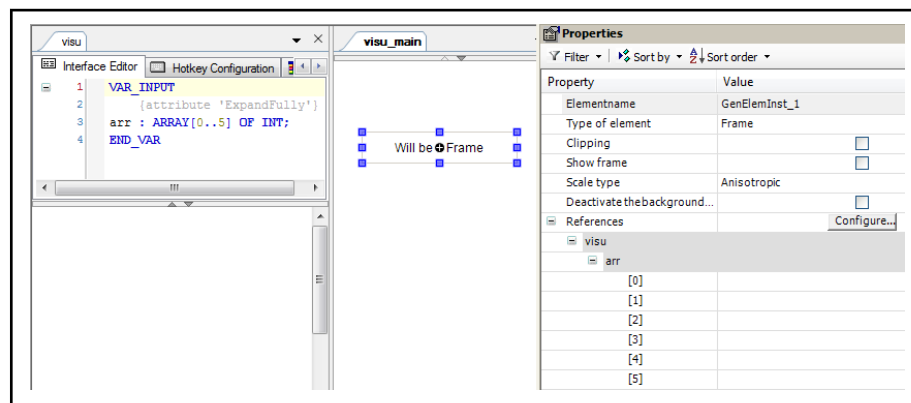


Fig. 17-78:

Attribute "global_init_slot"

This pragma defines the initialization order of function blocks and Global Variable Lists.

The sequence for the initialization of variables from global variable lists is not specified from the very beginning. Sometimes, however, specifying such a sequence is necessary for example if the variables of a list of variables depend on another list.

In this case, the pragma for specifying the sequence during the global initialization can be used.

Syntax:

```
{attribute 'global_init_slot' := '<value>'}
```

The placeholder <value> has to be replaced by an integer value determining the significance in the initialization sequence.

The default value for a GVL is 49990.

The default value for a POU (program, function block) is 50,000.

A lower value causes an earlier initialization. If several function blocks or Global Variable Lists have the same value for the attribute "global_init_slot", the order of their initialization remains undetermined.



This attribute cannot be used for single variables.

Example: The example project contains two global variable lists, GVL_1 and GVL_2:



Fig. 17-79: Two global variable lists

The global variable "A" is part of the global variable list GVL_1:

```
{attribute 'global_init_slot' := '300'}
VAR_GLOBAL
  A: INT:= 1000;
END_VAR
```

The initialization value of the variable "B" from GVL_2 depends on the variable "A" from GVL_1.

Initialization values of the variables "B" of GVL_2:

```
{attribute 'global_init_slot' := '350'}
VAR_GLOBAL
  B : INT:=GVL_1.A + 10;
END_VAR
```

If the attribute "global_init_slot" of the global variable list GVL_1 is set to 300 - that is the lowest initialization value in the example - it is ensured that the expression `GVL_1.A+10` is well defined when "B" is initialized.

The program PlcProg uses the two variables "GVL_1.A" and "GVL_2.B".

Its initialization point in time has to be later than the initialization of both the Global Variable Lists (thus, 'global_init_slot' has to be greater than 350; in the example 10,000. It also works with the default value 50,000, thus, "global_init_slot").

Use of variables

```
{attribute 'global_init_slot' := '10000'}
PROGRAM PlcProg
VAR
  ivar_1:INT:= GVL_1.A;
  ivar_2:INT:= GVL_2.B;
END_VAR
```

Attribute 'hide'

The pragma 'hide' prevents variables or POUs from being shown in the IndraLogic interface.

Purpose: You can intentionally hide variables or POUs from the user using this pragma. As a result, the variables or function blocks are not visible in the "Library manager" and in the input assistant and they are not provided "List components" function.

In online mode, the variables or function blocks are not visible in the declaration part and debug functionalities can also not be applied to them.

Syntax:

```
{attribute "hide"}
```

Insertion location: For variables, above the line with the declaration of the variables. For POU's, in the first line.

Example

The function block **myPOU** uses the attribute {attribute "hide"}:

```
FUNCTION_BLOCK myPOU
VAR_INPUT
  a:INT;
{attribute 'hide'}
  a_invisible: BOOL;
  a_visible: BOOL;
END_VAR
VAR_OUTPUT
  b:INT;
END_VAR
```

Two instances of the function block **myPOU** are defined in the main program.

Program:

```
PROGRAM PlcProg
VAR
  POU1, POU2: myPOU;
END_VAR
```

While one input value is now implemented for POU1, the "List components" function, which opens when typing POU1. (in the implementation section of **PlcProg**), displays the variables a, a_visible and b, but not the hidden variable a_invisible.



Use the pragma `hide_all_locals` to hide all local variables of a declaration.

Also refer to

- [chapter "Attribute "hide_all_locals" on page 390](#)

Attribute "hide_all_locals"

This pragma prevents all local variables of a signature from being visible in the display of the "List components" function, in the input assistant or in the declaration section in online mode.

Syntax:

```
{attribute "hide_all_locals"}
```

Insertion location: First line above the declaration section of the POU.

Example

The function block **myPOU** uses the attribute :

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
  a:INT;
END_VAR
VAR_OUTPUT
  b:BOOL;
END_VAR
VAR
```

```
c,d:INT;
END_VAR
```

Two instances of the function block **myPOU** are defined in the main program.

Program:

```
PROGRAM PlcProg
VAR
    POU1, POU2: myPOU;
END_VAR
```

While an input value for **POU1** is now implemented, the "List components" function, which opens when typing **POU1.** in the implementation section of **PlcProg**, displays the variables **a** and **b**, but not the hidden local variables **c** or **d**.

Also refer to

- [chapter "Attribute 'hide'" on page 389](#)

Attribute "initialize_on_call"

This pragma can be applied to input variables. It causes the input variables of a function block to be initialized upon each function block call. If an input expects a pointer and this pointer has been removed during an online change, this input is set to **zero**.

Syntax:

```
{attribute "initialize_on_call"}
```

Insertion location: Line above the line with the declaration of the corresponding input variables.

Attribute "init_namespace"

The effect of this pragma is that a variable of the type **STRING** or **WSTRING**, which is declared in a library function block with this pragma, is initialized when used in the project with the current namespace of the library.

Syntax

```
{attribute "init_namespace"}
```

Insertion location: The line above the line with the declaration of the variables in a library function block.

Example

The function block **POU** is provided with the necessary attributes:

```
FUNCTION_BLOCK POU
VAR_OUTPUT
{attribute 'init_namespace'}
    myStr: STRING;
END_VAR
```

An instance **fb** of the function block **POU** is defined in the main program **PlcProg**:

```
PROGRAM PlcProg
VAR
    fb:POU;
    newString: STRING;
END_VAR
newString := fb.myStr;
```

The variable **myStr** is initialized with the current namespace, e.g. **MyLib**.

This value is assigned to **newString** in the main program.

Also refer to

- [chapter 18.1.14 "Object 'Library manager'" on page 516](#)

Attribute "init_on_onlchange"

The effect of this pragma is that the variable to which the pragma is applied, is initialized upon each online change.

Syntax:

```
{attribute "init_on_onlchange"}
```

Insertion location: The line above the line with the declaration of the variables.



IndraWorks 14VRS and later are provided with a faster version of the online change for small changes:

If only code was changed, only the affected POU's are compiled and subsequently loaded. More code is not executed.

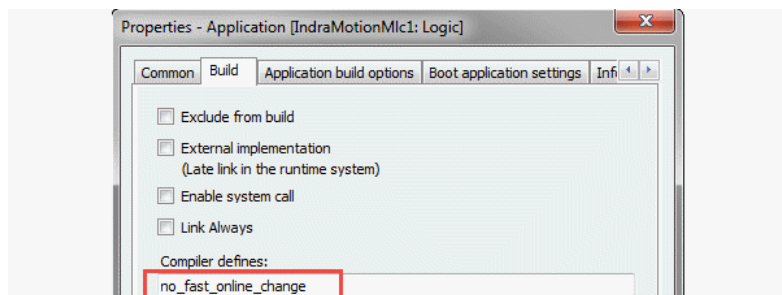
No initialization code is executed.

This means that also no code is generated when variables are initialized with the `init_on_onlchange` attribute.

As a rule, this has no effect, as the attribute is primarily used for initializing variables with addresses. However, a variable cannot change its address during an online change.



To ensure the effect of the `init_on_onlchange` attribute in the entire application code, disabled the fast online change in general for the application by using the compiler definition `no_fast_online_change`.



Insert the definition into the application "Properties" ("Build" tab).

Attribute "instance-path"

This pragma can be applied to a local STRING variable and causes this local STRING variable to be initialized in sequence with the Project Explorer path of the POU to which it belongs.

This can be useful for error messages.

The application of the pragma requires the application of the attribute `"reflection"` to the associated POU as well as the application of the additional attribute `"noinit"` to the STRING variable.

Syntax:

```
{attribute "instance-path"}
```

Insertion location: The line above the line with the declaration of the STRING variable.

Example

The following function block contains the attributes "reflection", "instance-path" and "noinit".

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
  {attribute 'instance-path'}
  {attribute 'noinit'}
  str: STRING;
END_VAR
```

An instance **myPOU** of the function block **POU** is defined in the main program **PlcProg**:

```
PROGRAM PlcProg
VAR
  myPOU:POU;
  myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

After initializing the instance **myPOU**, the path of the instance **myPOU** is assigned to the string variable **str**, in the example **PLCWinNT.Application.PlcProg.myPOU**. This path is assigned to the variable **myString** in the main program.



The length of a string can be freely defined (even >255), but note that the string is truncated at the end if it is assigned to a variable whose data type is too small.

Also refer to

- [chapter "Attribute "reflection" on page 409](#)
- [chapter "Attribute "noinit" on page 400](#)

Attribute 'io_function_block', 'io_function_block_mapping'

With the `io_function_block` attribute, you mark a function block in order to prepare it for the assignment to a channel in the I/O mapping of the device configuration. It is then shown in the "Select function block" dialog, see [chapter 18.3.1 "Dialog "Select function block"" on page 700](#).

With the `io_function_block_mapping` attribute, you mark a parameter that should be used when mapping the FB to a device channel in this kind of function block. You can provide the attribute to multiple parameters of the function block. For I/O mapping, the first one is used automatically whose type matches the channel (input, output, data type).

Syntax:

Program:

```
{attribute 'io_function_block'}

{attribute 'io_function_block_mapping'}
```

Insertion location: In the declaration above the function block, or the line above the parameter declaration.

Example

```
{attribute 'io_function_block'}
```

```

FUNCTION_BLOCK Scale_Output_Int
VAR_INPUT
    iInput : INT;
    iNumerator : INT;
    iDenominator : INT := 1;
    iOffset : INT := 0;
END_VAR
VAR_OUTPUT
    {attribute 'io_function_block_mapping'}
    iOutput : INT;
END_VAR
VAR

```

Also refer to [chapter 18.3.1 "Dialog "Select function block"" on page 700](#).

Attribute "linkalways"

The effect of this pragma is that the associated object is marked with the compiler and is thus always in the compiler information.

This means that the object is always compiled and loaded to the PLC.

The pragma is applied to for POU and GVLs located below an application or in libraries below an application or in the "General module folder".

The **Always link** option in the **Compile** category of the object properties has the same effect.

Syntax:

```
{attribute "linkalways"}
```

Insertion location: The line above the line with the keyword `VAR_GLOBAL` in the declaration section of POU and GVLs.

When using the symbol configuration editor, IndraLogic uses the marked POU as basis for the selectable variables of the symbol configuration used.

Example

The implementation of the Global Variable List **GVLMoreSymbols** uses the attribute "linkalways":

```

{attribute 'linkalways'}
VAR_GLOBAL
    g_iVar1: INT;
    g_iVar2: INT;
END_VAR

```

With this code, the variables of **GVLMoreSymbols** are provided as selectable symbols.

Also refer to

- [chapter 10.2 "Symbol configuration" on page 156](#)

Attribute 'monitoring'

The effect of this pragma is that you can monitor values of properties or function calls in the online view of the IEC editor or in a monitoring list.

There are two possible attribute values for this: 'variable' and 'call'

Syntax

Program:

```

{attribute 'monitoring' := 'variable'}
{attribute 'monitoring' := 'call'}

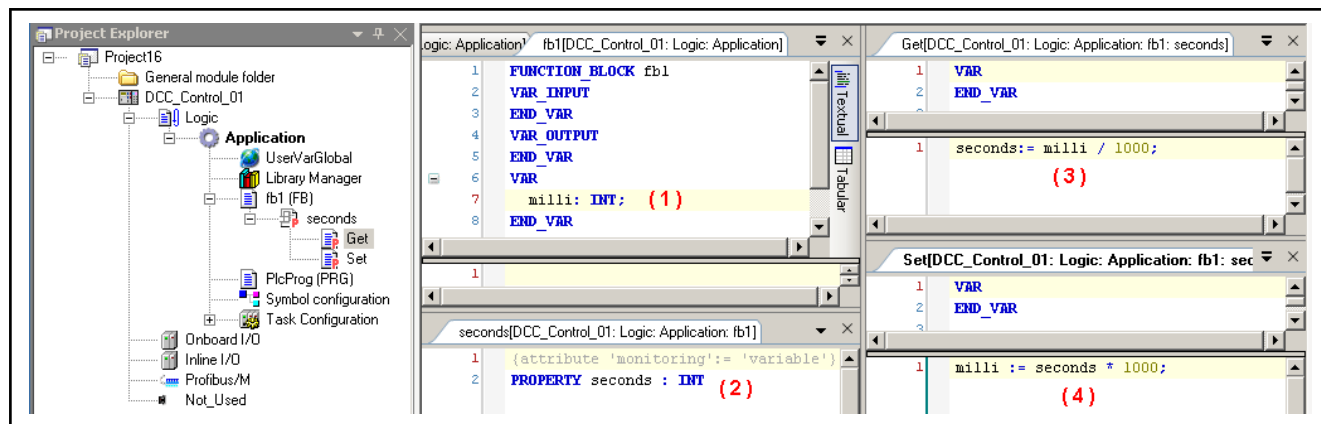
```

Monitoring properties

The pragma has to be inserted into the line above the property definition ([chapter "Object 'Property'" on page 537](#)).

Subsequently, name, type and value of the variables of the property are displayed in the online view of the function block using the property or in a monitoring list ([chapter 13.1.2 "Using watch lists" on page 196](#)).

Values to force the property variables can also be prepared in the online view or the watch list.



- (1) Function block "fb1" with the local variable 'milli'
- (2) Property 'seconds' with attribute pragma
- (3) "Get" of the 'seconds' property
- (4) "Set" of the 'seconds' property

Fig. 17-80: Example of the 'seconds' property prepared for monitoring

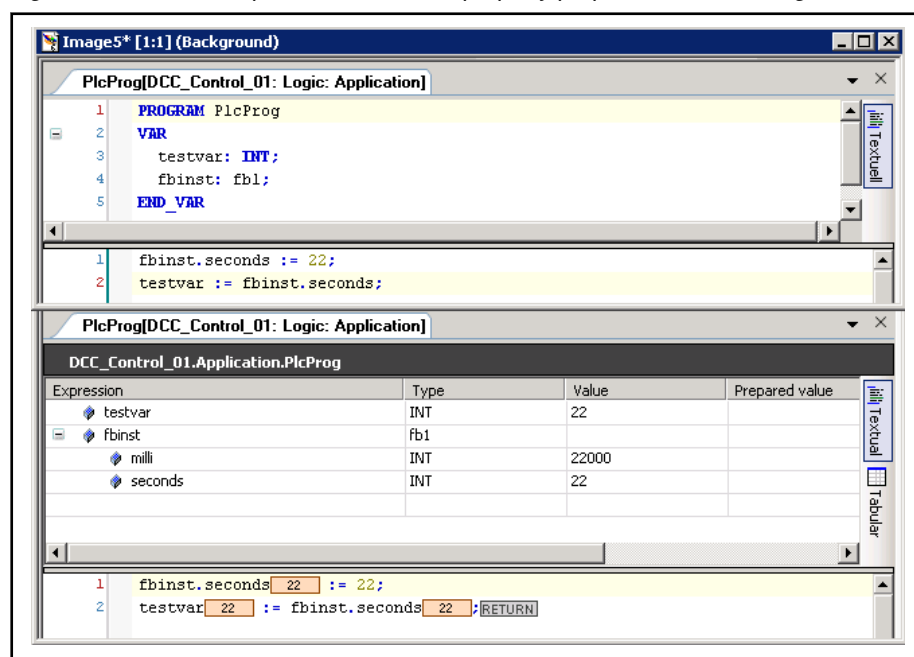


Fig. 17-81: Example of a monitoring view with the 'seconds' property

There are two different ways to display the current value of the "properties" variables.

Decide for each case which attribute pragma is suitable to display the desired value. The decision depends on whether further operations with the variables are implemented in the property.

1. Pragma {attribute 'monitoring':='variable'}

An implicit variable is created for the property that is assigned the current properties value whenever the application calls either the "Set" or "Get" method. The value stored last in this variable is displayed in the monitoring.

2. Pragma {attribute 'monitoring':='call'}

This attribute can only be used for properties that return only simple data types or pointers, but no structured types:

The value to be monitored is read or written by calling the property directly, i.e. the monitoring service of the runtime system executes the `Get` or the `Set` method of the property function.



If this monitoring type is selected instead of using an implicit variable (as for {attribute "monitoring":="variable"}), consider possible side effects that can occur if additional operations are implemented in the "Property" function.



The pragma `monitoring` is also evaluated for the symbol configuration. Only read access is possible for the `variable` value.



With the context menu command **Add to monitoring list**, a variable on which the cursor is presently positioned is adopted directly into the monitoring list in online mode.



Forcing or writing functions is not supported. However, you can implicitly implement forcing by adding an additional input parameter for the respective function, which serves as an internal force flag.



Function monitoring is not possible in the compact runtime system.

Monitoring results and function calls:

The monitoring of function blocks can be used for every constant value that can be interpreted as a 4-byte numeric value (e.g., INT, SINT, DINT).

For all other input parameters (e.g., BOOL), use a variable instead of the constant parameter. The pragma {attribute 'monitoring':='call'} has to be inserted in the line above the function declaration.

In the online view of a POU, in which the result of a function block is assigned to a variable, this variable is monitored in the text editor view. The variable can also be added to a watch list for the same purpose ([chapter 13.1.2 "Using watch lists" on page 196](#)). Note that the "Watch list" command can be used to obtain the variable directly in a watch list

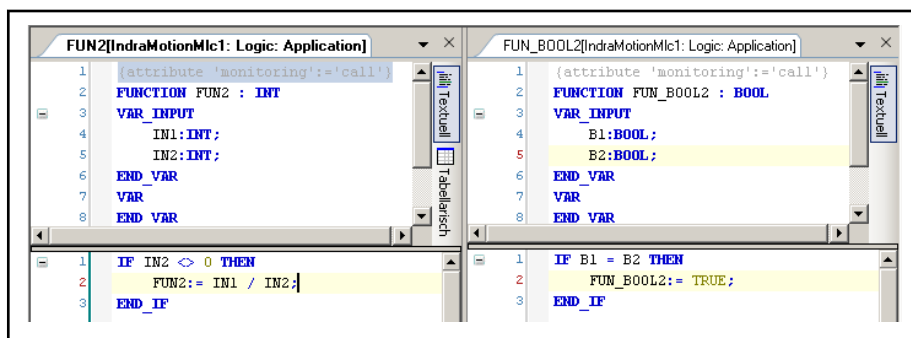


Fig. 17-82: Example of the functions `FUN2` and `FUN_BOOL2` with 'monitoring' attribute

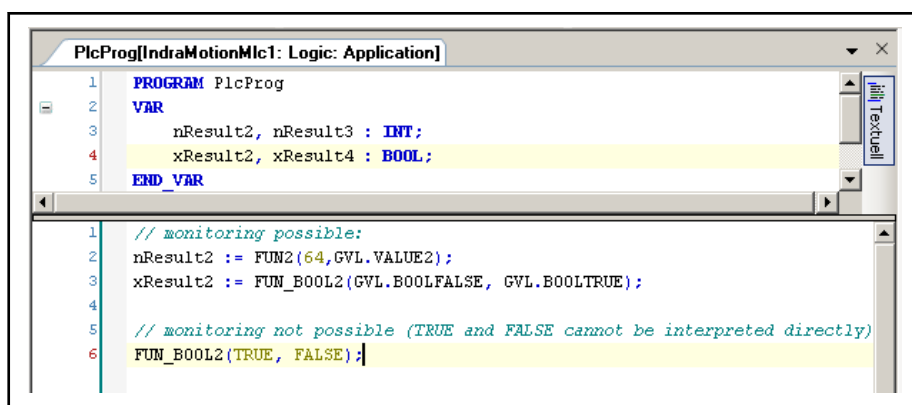


Fig. 17-83: Example of calling the functions FUN2 and FUN_BOOL2 in a program POU

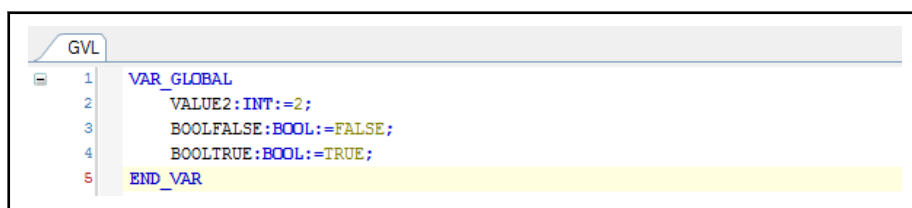


Fig. 17-84: Constants of the Global Variable List (GVL)

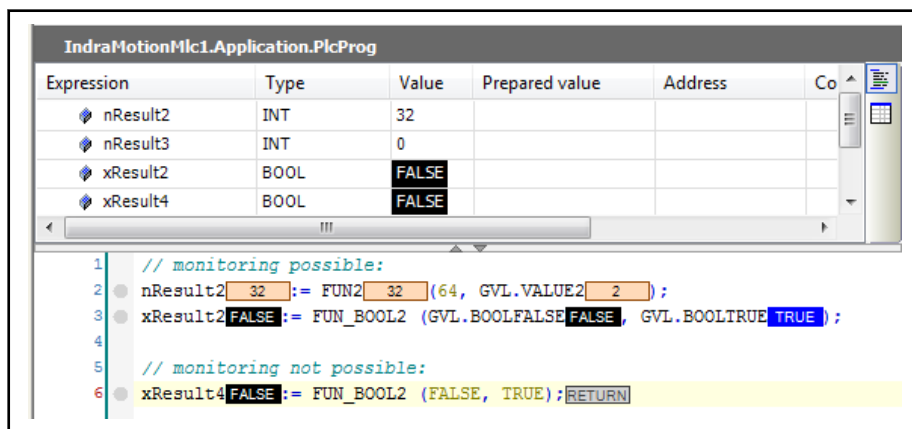


Fig. 17-85: Example of function calls in online mode

Also refer to

- [chapter "Object 'Property'" on page 537](#)

Attribute "namespace"

Together with the attribute "symbol", the pragma allows {attribute "namespace"} to redefine the namespace of project variables ([Attribute "symbol" on page 410](#)).

It can be used for complete POU's such as GVLs or programs, but not for individual variables. The respective variables are exported to the symbol file with the new namespace definition. They are then available after download to the control. This allows to also access variables of function blocks or visualizations originally provided with a different namespace. Thus, an IndraLogic 1.x visualization can also run in the 2G environment.

For information on the "symbol configuration", refer to ["Symbol configuration" object on page 553](#).

A new symbol file is created when downloading or changing a project online. It is loaded together with the application onto the control.

Syntax:

```
{attribute 'namespace' := '<namespace>'}
```

Example of a namespace violation for the variables of a program

```
{attribute 'namespace':='prog'}
PROGRAM PLC_PRG
VAR
{attribute 'symbol' := 'readwrite'}
  iVar:INT;
  bVar:BOOL;
END_VAR
```

If iVar was for example accessed previously via "App1.PLC_PRG.iVar", it can now be triggered via "prog.iVar".

More replacement examples

Original namespace	Variable	Namespace substitution	Variable access in the current project
App1.Lib2.GVL2	Var07	{attribute 'namespace' := ""}	.Var07
App1.GVL2	Var02	{attribute 'namespace' := 'Ext'}	Ext.Var02
App1.GVL2.FB1	Var02	{attribute 'namespace' := 'App1.GVL2'}	App1.GVL2.Var02

The replacements in the table cause the following entries in the symbol file:

```
<NodeList>
  <Node name="">
  <Node name="Var07" type="T_INT" access="ReadWrite">
</Node> </NodeList>

<NodeList>
  <Node name="Ext">
  <Node name="Var02 " type="T_INT" access="ReadWrite">
</Node> </Node> </NodeList>

<NodeList>
  <Node name="App1">
  <Node name="GVL2">
  <Node name="Var02 " type="T_INT" access="ReadWrite">
</Node> </Node> </Node> </NodeList>
```

Attribute 'no_assign', attribute 'no_assign_warning'

The pragma `no_assign` results in compiler errors being displayed if an instance of the function block is assigned to another instance of the same function block. Such assignments are often to be avoided if the function block contains pointers and pointers cause problems, as they are copied as well during the value assignment.

The pragma `no_assign_warnig` results in the same as for the pragma `no_assign` with compiler warnings instead of compiler errors.

Syntax:

```
{attribute "no_assign"}
```

Insertion location: First line in the declaration section of a function block.

Example

Assignment of function block instances containing pointers.

In this example, the value assignment of the function block instances cause problems while executing `fb_exit`:

```

VAR GLOBAL
inst1 : TestFB;
awsBufferLogFile : ARRAY [0..9] OF WSTRING(66); (* Area: 0, Offset: 0x1304 (4868)*)
LogFile : SEDL.LogRecord := (sFileName := 'LogFile.log', pBuffer := ADR(awsBufferLogFile),
    udiMaxEntriesFile := UDINT#10000, udiMaxBuffered := UDINT#10,
    uiLineSize := UINT#64, wsSep := " ", xCircular := TRUE,
    siDateFormat := SINT#0, siTimeFormat := SINT#0);

END VAR
PROGRAM PlcProg
VAR
    inst2 : TestFB := inst1;
    LogFileNew
END VAR

```

In this case, `LogRecord` manages a list of pointers, for which various actions are executed if `fb_exit`. Problems result due to the assignment, as `fb_exit` is executed twice. Prevent this by adding the attribute `"no_assign"` to the declaration of the function block **TestFB**:

```

{attribute 'no_assign'}
FUNCTION_BLOCK TestFB
VAR_INPUT
...

```

The following compile errors are output:

```

C0328: Assignment not allowed for type TestFB
C0328: Assignment not allowed for type LogRecord

```

If the pragma `no_assign_warning` is used instead of the pragma `no_assign` for the function block `TestFB`, then the C0328 message is issued as compiler warning, not as a compiler error.

Attribute "no_check"

This pragma prevents the check function from being called for the POU (POUs for implicit checks).

Since the check functions can affect the processing speed of the program, it can be useful to apply the attribute to function blocks that have already been checked or that are frequently called.

Add the pragma to the declaration of a POU.

Syntax:

```
{attribute "no_check"}
```

Insertion location: First line in the declaration section of the POU



The attribute also affects the child objects of a POU automatically!

Example:

If the attribute is entered into a program, check functions are also not executed for actions assigned to this program.

Also refer to

- [chapter 18.1 "Objects" on page 479](#)

Attribute "no_copy"

In general, an online change requires a re-allocation of instances, for example of a POU. The value of the variable in the instance is copied.

The pragma prevents the value of the variable in the instance from being copied during an online change. The variable is re-initialized during an online change instead.

This can be useful for a local pointer variable that points to a variable that has just been shifted by the online change and thus has a changed address.

Insert the attribute into the declaration section above the line of the declaration of the variables concerned.

Syntax:

```
{attribute "no_copy"}
```

Attribute 'no-exit'

This attribute suppresses the call of the FB_exit method of a function block for one of its instances. Insert the attribute in the line in front of the declaration of the function block instance.

Syntax:

```
{attribute "no-exit"}
```

Example

The method **FB_exit** is added to the function block **POU_ex**. Two instances of the function block **POU_ex** are created in the main program **PlcProg**.

Program:

```
PROGRAM PlcProg
VAR
POU1 : POU_ex;
{attribute 'no-exit'}
POU2 : POU_ex;
END_VAR
```

POU 1: FB_exit is called, POU 2: FB_exit is not called.

Also refer to

- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)

Attribute "noinit"

This pragma is applied to variables that should not be initialized implicitly.

Syntax:

```
{attribute "no_init"}
```

```
{attribute "no-init"}
```

```
{attribute "noinit"}
```

Insertion location: Line above the declaration line of the respective variables concerned in the declaration section.

Example

```
PROGRAM PlcProg
VAR
  A : INT;
  {attribute 'no_init'}
  B : INT;
END_VAR
```

When the associated application is reset, the integer variable **A** is implicitly re-initialized with 0, whereas the variable **B** retains its current value.

Attribute "no_instance_in_retain"

Use this attribute to impede that the instance of a certain function block is stored in the retain area.

Syntax:

```
{attribute "no_instance_in_retain"}
```

Insertion location:

Line above the `FUNCTION_BLOCK` declaration in the declaration section of the function block.

An error message is displayed if a function block instance is declared as RETAIN variable.

Also refer to

- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)

Attribute "no_virtual_actions"

This pragma is used for function blocks that are derived from a function block implemented in SFC and that use the fundamental SFC sequence of this basic class. The actions called from there shows the same virtual response as the methods. This means that the implementations of the actions in the basic class can be replaced by the derived class with its own specific implementations.

If you apply the pragma to the basic class, its actions are protected against overloading.

Syntax:

```
{attribute "no_virtual_actions"}
```

Insertion location: Top line in the declaration section of the function block

Example:

The following example shows the function block `POU_SFC` that creates the basic class for the derived function block `POU_child`.

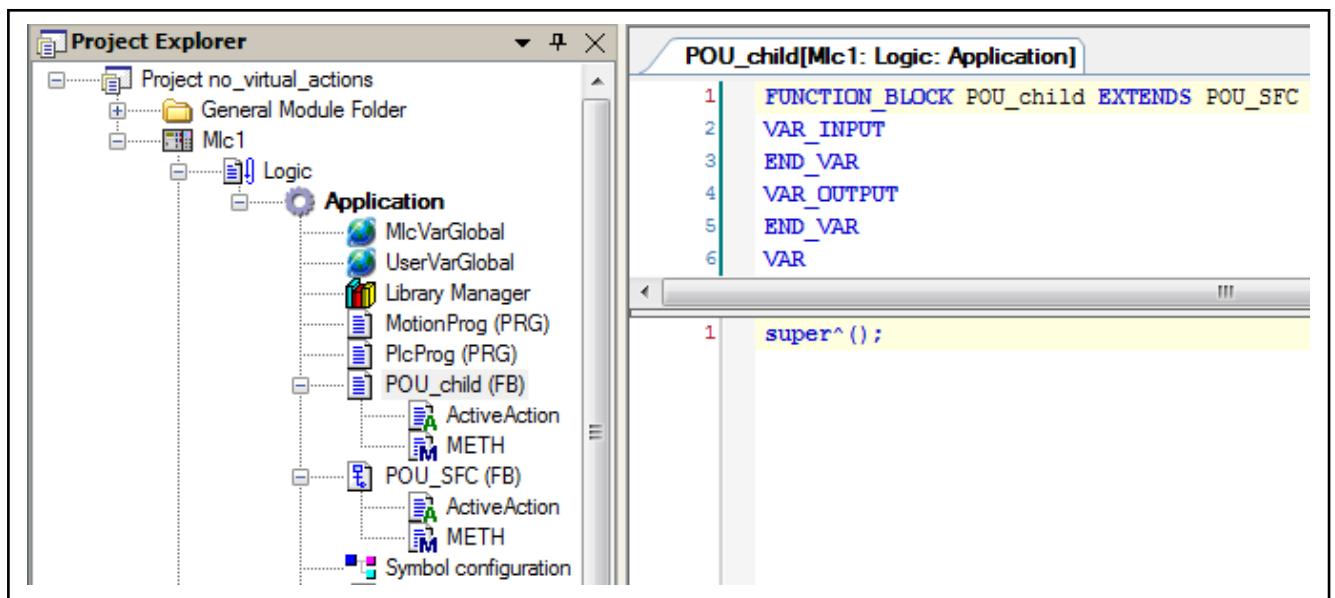


Fig. 17-86: FB `POU_child` extends FB `POU_SFC`

Using the keyword `SUPER`, the derived class (FB) `POU_child` calls the sequence of the basic class (basis FB) written in SFC.

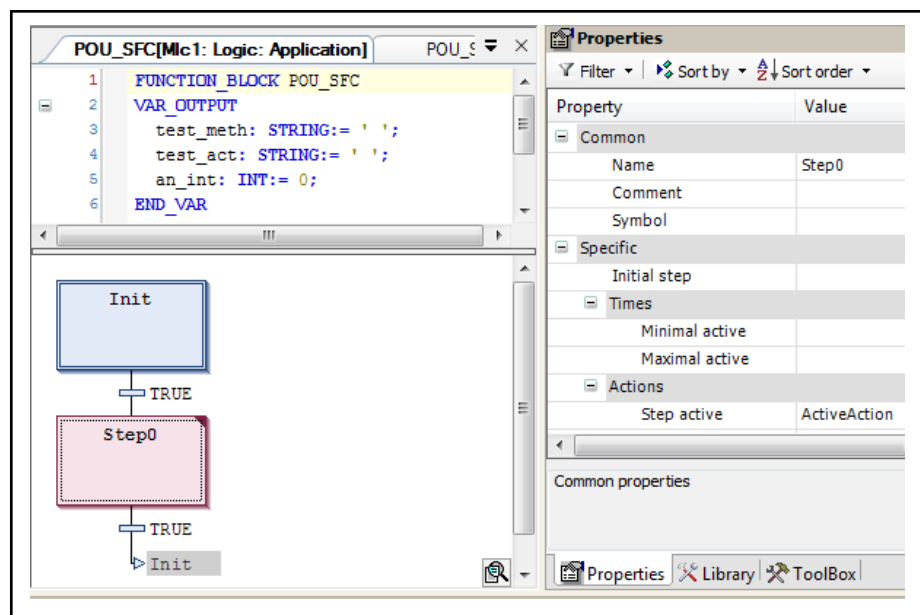


Fig. 17-87: Function block POU_SFC with the action "ActiveAction"

The exemplary implementation of this sequence is limited to the initial step followed by only one step "Step0" with the linked step action "ActiveAction" which assigns the output variables:

ActiveAction (POU_SFC)

```

an_int:=an_int+1; // counting the action calls
test_act:='father_action'; // writing test_act
METH(); // Calling method METH for writing test_meth

```

METH (POU_SFC)

```

METHOD PUBLIC METH
VAR_INPUT
END_VAR
test_meth:= 'father_meth';

```

For the derived class POU_child, the step action is replaced by a special implementation of "ActiveAction" only differing from the original by assigning the "child_action" string instead of "father_action" to the "test_act" variable.

Action and method have to be assigned manually to POU_child and with the same name.

ActiveAction (POU_child)

```

an_int:=an_int+1; // counting the action calls
test_act:='child_action'; // writing test_act
METH(); // Calling method METH for writing test_meth

```

The "METH" method, assigning the "father_method" string to the "test_meth" variable in the basic class, is overwritten in such a way that "test_meth" gets the "child_method" value.

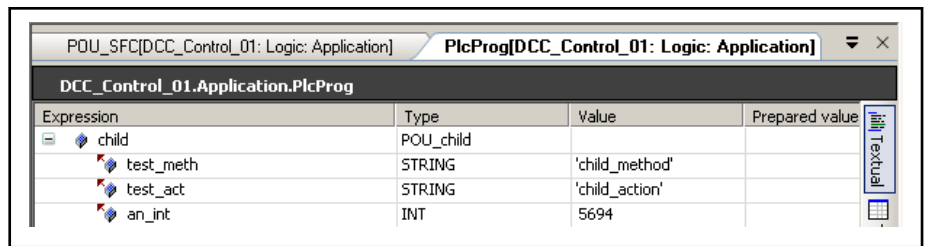
METH (POU_child)

```

METHOD PUBLIC METH
VAR_INPUT
END_VAR
test_meth:= 'child_meth';

```

The main program "PlcProg" calls an instance of the function block "POU_child" called "child". As expected, the value of the strings reflects the call of action and method of the derived class:



Expression	Type	Value	Prepared value
child	POU_child		
test_meth	STRING	'child_method'	
test_act	STRING	'child_action'	
an_int	INT	5694	

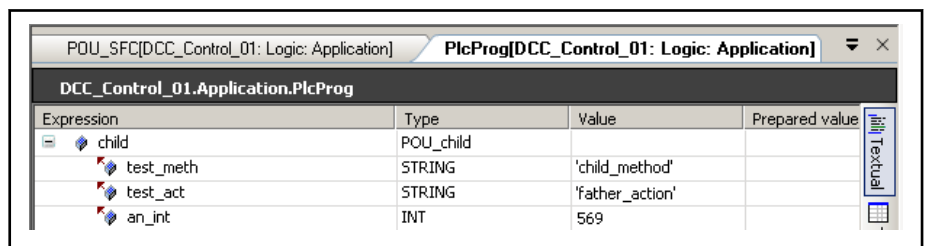
Fig. 17-88: Main program "PlcProg"

If the "no_virtual_actions" attribute precedes the basis, another behavior can be observed:

POU_SFC with {attribute 'no_virtual_actions'}

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK POU_SFC
VAR_OUTPUT
    test_meth: STRING:= ' ';
    test_act: STRING:= ' ';
    an_int: INT:= 0;
END_VAR
```

The implementation of the derived class is used for the "METH" method. Calling the step action now results in calling the "ActiveAction" of the basic class. Thus, "test_act" obtains the value 'father_action':



Expression	Type	Value	Prepared value
child	POU_child		
test_meth	STRING	'child_method'	
test_act	STRING	'father_action'	
an_int	INT	569	

Fig. 17-89: Main program "PlcProg" online, using the attribute "no_virtual_actions" for FUNCTION_BLOCK POU_SFC

Attribute 'pingroup'

The effect of this pragma is that the input pins or output pins (parameters) are grouped in the declaration of a function block.

In the FBD/LD editor, a pin group defined like this can be displayed as an enlarged or reduced unit on the inserted function block.

Multiple groups are possible and are distinguished by their names.

IndraLogic saves the respective state (reduced) per function block with the project options.

Syntax:

```
{attribute "pingroup" := "<group name>"}
```

Insertion location: Line above the declaration of the input or output variables concerned in the declaration section of a function block.

Example

Two groups are defined: General (i1, out1) and group1 (i2, g1). r1, r2, outRes1 and g2 are always displayed

```
FUNCTION_BLOCK FB
VAR_INPUT
    r1 : REAL;
    {attribute 'pingroup' := 'general'}
    i1 : INT;
    {attribute 'pingroup' := 'group1'}
```

```

i2 : INT;
r2 : REAL;
END_VAR
VAR_OUTPUT
  outRes1 : REAL;
  {attribute 'pingroup' := 'general'}
  out1 : INT;
  {attribute 'pingroup' := 'group1'}
  g1 : INT;
  g2 : REAL;
END_VAR

```

Attribute 'pin_presentation_order_inputs/outputs'

The pragmas defined the displayed sequence of the inputs/outputs of a function block.

Syntax *Program:*

```

{attribute 'pin_presentation_order_inputs' := '<First_Input_Name>, (<Next_Input_Name>, )*
  ( *, )? (<Next_Input_Name>)* <Last_Input_Name>'}
{attribute 'pin_presentation_order_outputs' := '<First_Output_Name>, (<Next_Output_Name>, )*
  ( *, )? (<Next_Output_Name>)* <Last_Output_Name>'}

```

- *

The terminal character serves as a wildcard for all inputs/outputs that are not specified in the display order. If the terminal character is missing, then the missing inputs/outputs are appended at the end.

- (...)?

The content of the parenthesis is optional.

- Insertion location: First line in the declaration section of a function block



This pragma is not evaluated when pragma {attribute 'pingroup' := '<Group_Name>'} is used.

Example

```

{attribute 'pin_presentation_order_inputs' := 'input_2,*,input_1'}
{attribute 'pin_presentation_order_outputs' := 'output_2, output_1'}
FUNCTION_BLOCK POU_BASE
VAR_INPUT
  input_1 : BOOL;
  input_2 : INT;
  input_3 : INT;
  input_4 : INT;
END_VAR

VAR_OUTPUT
  output_1 : BOOL;
  output_2 : INT;
  output_3 : INT;
  output_4 : BOOL;
END_VAR

FUNCTION_BLOCK PLC_PRG
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  pouBase_A: POU_BASE;
END_VAR

```

In the representation of function module instance `pouBase_A`, the pragmas result in the following arrangement of input and output pins:

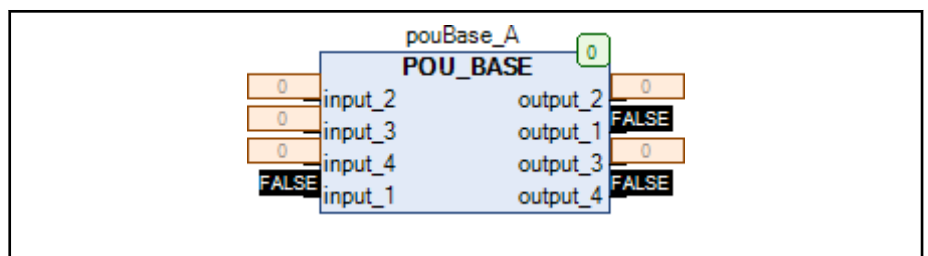


Fig. 17-90:

Refer to [chapter "Attribute 'pingroup'"](#) on page 403

Attribute "obsolete"

The effect of this pragma is that a defined warning is displayed for a data type definition during compilation if the data type (structure, function block, etc.) is used in the project. Thus, attention can be drawn for example to the fact that a data type is no longer valid, as an interface has changed and this should also be implemented into the project.

In contrast to a message pragma, this warning is defined centrally for all instances of a data type.

Syntax:

```
{attribute "obsolete" := "user defined text"}
```

Insertion location: Line of the data type definition or in a line above it.

Example

The pragma is inserted in the definition of the function block fb1:

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
  i:INT;
END_VAR
```

When using use fb1 as data type, for example in `fbinst:fb1;`, the following warning is displayed when compiling the project: "datatype fb1 not valid".

Also refer to

- [chapter 17.6.1 "Message pragmas" on page 378](#)

Attribute "pack_mode"

This pragma defines how a data structure is packed during the allocation.

The attribute has to be inserted above the data structure and affects the zip-ping of the entire structure.

Syntax:

```
{attribute "pack_mode" := "<Value>"}
```

Insert location is above the declaration of the data structure

Possible values for <Value>

pack_mode	Associated packing method	Description
0	Aligned	All variables are allocated to byte addresses. There are no memory gaps.
1	1-byte-aligned	

pack_mode	Associated packing method	Description
2	2-byte-aligned	<i>There are</i> • 1-byte variables at byte addresses • 2 byte variables at even addresses. A maximum gap of 1 byte results. • 4 byte variables at addresses that can be divided by 2. A maximum gap of 1 byte results. • 8 byte variables at addresses that can be divided by 2. A maximum gap of 1 byte results. • Strings are always at byte addresses. No gaps result.
4	4-byte-aligned	<i>There are</i> • 1-byte variables at byte addresses • 2 byte variables at even addresses. A maximum gap of 1 byte results. • 4 byte variables at addresses that can be divided by 4. A maximum gap of 3 byte results • 8 byte variables at addresses that can be divided by 4. A maximum gap of 3 byte results • Strings are always at byte addresses. No gaps result.
8	8-byte-aligned	<i>There are</i> • 1-byte variables at byte addresses • 2 byte variables at even addresses. A maximum gap of 1 byte results. • 4 byte variables at addresses that can be divided by 4. A maximum gap of 3 byte results • 8 byte variables at addresses that can be divided by 8. A maximum gap of 7 byte results • Strings are always at byte addresses. No gaps result.



Depending on the structure, there may be no difference in the memory mapping of the individual modes. Therefore, the memory allocation of a structure with `pack_mode = 4` can correspond to that of `pack_mode = 8`.



Arrays of structures:

If the structures are combined in arrays, then bytes are added at the end of the structure so that the next structure is aligned.

Example 1

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
  STRUCT
    Enable: BOOL;
    Counter: INT;
    MaxSize: BOOL;
    MaxSizeReached: BOOL;
  END_STRUCT
END_TYPE
```

The memory range for a variable of the data type myStruct is allocated "aligned":

If the storage address of your Enable component is 0x0100 for example, the component Counter follows at the address 0x0101, MaxSize at the address 0x0103 and MaxSizeReached at the address 0x0104.

Use "pack_mode":=2 and Counter would be at 0x0102, MaxSize at 0x0104 and MaxSizeReached at 0x0106.

Example 2

```
STRUCT
  Var1 : BOOL := 16#01;
  Var2 : BYTE := 16#11;
  Var3 : WORD := 16#22;
  Var4 : BYTE := 16#44;
  Var5 : DWORD := 16#88776655;
  Var6 : BYTE := 16#99;
  Var7 : BYTE := 16#AA;
  Var8 : DWORD := 16#AA;
END_TYPE
```

	pack_mode= 0		pack_mode= 1		pack_mode= 2		pack_mode= 4		pack_mode= 8	
	Variable	Value	Variable	Value	Variable	Value	Variable	Value	Variable	Value
0	Var 1	01	Var 1	01	Var 1	01	Var 1	01	Var 1	01
1	Var 2	11	Var 2	11	Var 2	11	Var 2	11	Var 2	11
2	Var 3	22	Var 3	22	Var 3	22	Var 3	22	Var 3	22
3	...	00	...	00	...	00	...	00	...	00
4	Var 4	44	Var 4	44	Var 4	44	Var 4	44	Var 4	44
5	Var 5	55	Var 5	55						
6	...	66	...	66	Var 5	55				
7	...	77	...	77	...	66				
8	...	88	...	88	...	77	Var 5	55	Var 5	55
9	Var 6	99	Var 6	99	...	88	...	66	...	66
10	Var 7	AA	Var 7	AA	Var 6	99	...	77	...	77
11	Var 8	AA	Var 8	AA	Var 7	AA	...	88	...	88
12	...	00	...	00	Var 8	AA	Var 6	99	Var 6	99
13	...	00	...	00	...	00	Var 7	AA	Var 7	AA
14	...	00	...	00	...	00				
15					...	00				
16							Var 8	AA	Var 8	AA
17							...	00	...	00
18							...	00	...	00
19							...	00	...	00

Example 3

```
STRUCT
  Var1 : BYTE := 16#01;
  Var2 : LWORD := 16#11;
  Var3 : BYTE := 16#22;
  Var4 : BYTE := 16#44;
  Var5 : DWORD := 16#88776655;
  Var6 : BYTE := 16#99;
  Var7 : BYTE := 16#AA;
  Var8 : WORD := 16#AA;
END_TYPE
```

Reference "Programming"

	pack_mode= 0		pack_mode= 1		pack_mode= 2		pack_mode= 4		pack_mode= 8	
	Variable	Value	Variable	Value	Variable	Value	Variable	Value	Variable	Value
0	Var 1	01	Var 1	01	Var 1	01	Var 1	01	Var 1	01
1	Var 2	11	Var 2	11						
2	...	00	...	00	Var 2	11				
3	...	00	...	00	...	00				
4	...	00	...	00	...	00	Var 2	11		
5	...	00	...	00	...	00	...	00		
6	...	00	...	00	...	00	...	00		
7	...	00	...	00	...	00	...	00		
8	...	00	...	00	...	00	...	00	Var 2	11
9	Var 3	22	Var 3	22	...	00	...	00	...	00
10	Var 4	44	Var 4	44	Var 3	22	...	00	...	00
11	Var 5	55	Var 5	55	Var 4	44	...	00	...	00
12	...	66	...	66	Var 5	55	Var 3	22	...	00
13	...	77	...	77	...	66	Var 4	44	...	00
14	...	88	...	88	...	77			...	00
15	Var 6	99	Var 6	99	...	88			...	00
16	Var 7	AA	Var 7	AA	Var 6	99	Var 5	55	Var 3	22
17	Var 8	AA	Var 8	AA	Var 7	AA	...	66	Var 4	44
18	...	00	...	00	Var 8	AA	...	77		
19					...	00	...	88		
20							Var 6	99	Var 5	55
21							Var 7	AA	...	66
22							Var 8	AA	...	77
23							...	00	...	88
24									Var 6	99
25									Var 7	AA
26									Var 8	AA
27									...	00

Attribute "qualified_only"

The effect of this pragma is that variables of a Global Variable List are only addressed by specifying the global variable name, for example `gvl.g_var`. This also applies to variables of the type "Enumeration" and can be useful to avoid confusion with the local variables.

Syntax:

```
{attribute "qualified_only"}
```

Insertion location: Line above `VAR_GLOBAL` in a GVL

Example**Global Variable List GVL:**

```
{attribute 'qualified_only'}
VAR_GLOBAL
    iVar:INT;
END_VAR
```

Within a POU, for example **PlcProg**, the global variable `iVar` can only be addressed using the prefix `GVL`:

```
GVL.iVar:=5;
```

However, the following incomplete call of the variable generates an error:

```
iVar:=5;
```

Attribute "reflection"

This purpose of this pragma is to identify function blocks in which IndraLogic is to search for local STRING variables to which the attribute "instance-path" is applied.

Syntax:

```
{attribute "reflection"}
```

For an example, refer to the description of the attribute "instance-path"

Also refer to

- [chapter "Attribute "instance-path" on page 392](#)

Attribute 'to_string'

The pragma affects how the result of converting an enumeration component with the `TO_STRING` operator is output. If the enumeration declaration has the pragma, then the name of the enumeration component appears as a string instead of the numeric value.

Syntax

```
{attribute 'to_string'}
```

Insertion location

First line above the declaration section of the enumeration.

Example

Declaration of the enumeration `color`:

```
{attribute 'to_string'}
TYPE color :
(
    red := 0,
    blue := 1,
    green := 2
);
END_TYPE
```

Conversion with TO_STRING:

```
PROGRAM PLC_PRG
VAR
    i_color: Color;
    s_show_color: STRING;
END_VAR
i_color := 1;
s_show_color := TO_STRING(i_color);
```

In this case, `str_show_color` gets the value 'blue' instead of '1' as the conversion result.

Also refer to [chapter 17.5.17 "Enumerations" on page 374](#)

Attribute "strict"

The pragma is saved and ensures the correct use of enumeration types.

As of IndraLogic 14VRS, each enumeration newly added to a project is automatically be provided with the "strict" attribute in the line above the TYPE declaration.

For compiler versions $\geq 3.5.7.0$, this causes compilation errors in the following cases:

- Arithmetic operation with variables of the enumeration type
- Assignment of a constant value - which does not correspond to an enumeration value - to a variable of the enumeration type
- Assignment of a non-constant value - with a data type than of the enumeration type - to a variable of the enumeration type.

The attribute can also be added explicitly or removed.

Syntax:

```
{attribute "strict"}
```

Attribute "subsequent"

This pragma is used to allocate variables to a memory space directly behind one another. If the list changes, the complete variable list is allocated at a new memory location.

This pragma is used in programs and Global Variable Lists.

Syntax:

```
{attribute "subsequent"}
```



`VAR_TEMP` in a program with the attribute "subsequent" causes a compiler error.



If one variable is in the 'RETAIN' list, the complete list is saved as 'RETAIN'.

Attribute "symbol"

The pragma `{attribute "symbol"}` defines which variables of a program or of a Global Variable List are to be included into the symbol configuration. This means that the variables are exported as symbols to a symbol list.

This symbol list is then available on the target system for external access as an XML file in the project directory and as a file invisible to the user.

For example, the symbol list can then be accessed by an OPC server. IndraLogic loads the variables provided with a symbol to the control, even if they are not explicitly configured or visible in the editor of the symbol configuration.



Note that the symbol configuration has to be created as an object below the respective application in the Project Explorer.



Note that only symbols from programs or global variable lists can be accessed. To access a symbol, the symbol name has to be entered completely.

Syntax:

```
{attribute "symbol" := "none"|"read"|"write"|"readwrite"}
{attribute "symbol" := "<access possibilities>"}
```

<access possibilities>: none, read, write, readwrite.

The default value `readwrite` applies if no parameter is specified.

Insertion location:

- To apply only to an individual variable, place the pragma in the line in front of the variable declaration.
- To apply to all variables in the declaration section of a program, place the pragma in the first line of the declaration editor. In this case, statements can still be set explicitly for individual variables explicitly in the respective line.

The parameter determines the possible **access** to a symbol. Possible specifications: 'none', 'read', 'write' or 'readwrite'. If no parameter is specified, the default value 'readwrite' applies.

Single variable example:

The variables "A" and "B" are exported with read-only and write access using the following configuration. Variable "D" is exported with read-only access.

Examples:

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PlcProg
VAR
  A : INT;
  B : INT;
  {attribute 'symbol' := 'none'}
  C : INT;
  {attribute 'symbol' := 'read'}
  D : INT;
END_VAR
```

Structural element example:

To export a structural variable to the symbol file, the access can be defined for each variable in the structure.

Type definition of a structure

```
TYPE ST_Symbols :
STRUCT
  {attribute 'symbol' := 'read'}
  iReadOnly: INT;
  {attribute 'symbol' := 'write'}
  iWriteOnly: INT;
  {attribute 'symbol' := 'readwrite'}
  iReadWrite: INT;
  {attribute 'symbol' := 'none'}
  iNoAccess: INT;
END_STRUCT
END_TYPE
```

When the access rights are defined in the structure, these access rights overwrite the access rights of the variable (stTest) using this structure.

The attribute of the variable "stTest" is of no importance (attribute "read", "write" or "readwrite").

Declaration of the structural instance

```
VAR
  {attribute 'symbol' := 'read'}
```

```
stTest: ST_Symbols;
END_VAR
```

Also refer to

- [chapter 9.6 "Using pragmas" on page 110](#)
- [chapter 10.2 "Symbol configuration" on page 156](#)

Attribute "warning disable", attribute "warning restore"

This pragma causes certain warnings to be suppressed. With the `warning restore` pragma, a suppressed message is enabled again.

Syntax:

```
{warning disable <compiler ID>}
```

```
{warning restore <compiler ID>}
```

<Compiler ID>: ID located at the beginning of an error message or a warning.

Example

Compiler message:

```
typify code ...
C0196: Implicit conversion from unsigned Type 'UINT' to signed Type 'INT' : possible change of sign
Compile complete -- 0 errors
```

Applying the pragma to a variable declaration:

```
VAR
    {warning disable C0195}
    test1 : UINT := -1;
    {warning restore C0195}
    test2 : UINT := -1;
END_VAR
```

test1 does not generate an error message, test2 generates an error message.

Attribute 'enable_dynamic_creation'

The `enable_dynamic_creation` pragma is required to use the operator `__NEW` in function blocks.

Syntax: {attribute 'enable_dynamic_creation'}

Insertion location: First line in the function block declaration.

Also refer to [chapter 17.3.58 "Operator '__NEW'" on page 327](#)

Attribute 'ProcessValue'

With the `ProcessValue` attribute, a component of a structure is structured. In the CFC editor, use the "Use attributed component as input" command to connect this structure to an input of scalar type.

Syntax {attribute 'ProcessValue'}

Insertion location: Line above the affected structure variable.

Example *Program:*

```
TYPE QINT :
STRUCT
    Status : STRING;
    {attribute 'ProcessValue'}
    Value1 : INT;
    Value2 : INT;
END_STRUCT
END_TYPE
```

Also refer to

- [chapter "Command 'Use attributed member as input'" on page 659](#)

17.6.3 Conditional pragmas

Conditional pragmas are used to affect the code generation during precompilation or compilation. The ST implementation language supports these pragmas.



They use conditional pragmas in the implementations of POU's. IndraLogic does not evaluate these conditional pragmas if used in the declaration section!

Use conditional pragmas to affect whether implementation code is taken into account for the compilation. This can for example depend on whether a certain variable is declared, whether a certain function block is present, etc.

Pragma	Description
<code>{define <identifier> <string>}</code>	The value can be queried and compared later with <code>hasvalue</code> .
<code>{undefine <identifier>}</code>	The <code>{define}</code> statement of the identifier <code><identifier></code> is cancelled. The identifier is now "undefined" again. If the currently specified identifier is not defined at all, the pragma is ignored.
<code>{IF <expr>}... {ELSIF <expr>}... {ELSE}... END_IF}</code>	These Pragmas are for the conditional compilation. The specified expressions <code><expr></code> have to be constant at the time of compilation. They are evaluated in the order in which they are displayed until one of the expressions indicate a non-zero value. The text linked to the statement is compiled. The other lines are ignored. The order of the sections is specified. The <code>ELSIF</code> and <code>ELSE</code> sections are optional. The <code>ELSIF</code> sections may occur any number of times. Multiple conditional compilation operators can be used within the constants <code><expr></code> .
<code><expr>}</code>	One or more operators can be used within the constant expression <code><expr></code> within the conditional compilation pragma <code>{IF}</code> or <code>{ELSIF}</code> .



Expressions and `define` definitions can be entered as **compiler definitions** in the **Compile** tab in the "Properties" dialog of POU's. When entering `define` definitions into the "Properties" dialog, omit the term `{define}` in contrast to the definition in the implementation code. In addition, multiple `define` definitions can be specified in the "Properties" dialog separated by commas.

Also refer to

- [chapter 9.3.3 "Structured Text \(ST\) and Extended Structured Text \(ExST\)" on page 101](#)

Operator defined (<Identifier>)

This operator causes that the expression obtains the value `TRUE`. It is required that the identifier `<Identifier>` was defined using a `{define}` statement and not undefined again afterwards with an `{undefine}` statement. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications `App1` and `App2`. Variable `pdef1` is defined by a `{define}` statement in `App1`, but not in `App2`.

```
{IF defined (pdef1)}
(* This code is processed in App1 *)
{info 'pdef1 defined'}
    hugo := hugo + SINT#1;
{ELSE}
(* the following code is only processed in App2 *)
{info 'pdef1 not defined'}
    hugo := hugo - SINT#1;
{END_IF}
```

There is an additional example of a message pragma: Only the message `pdef1 defined` is displayed in the message window when the application is compiled, as `pdef1` is actually defined. The message `pdef1 not defined` is displayed if `pdef1` is not defined.

**Operator defined
(variable: <Variable>)**

With this operator, the expression becomes `TRUE` if the variable <variable> is declared within the current scope of validity. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications `App1` und `App2`. Variable `g_bTest` is declared in `App1`, but not in `App2`.

```
{IF defined (variable: g_bTest)}
(* the following code is only processed in App2*)
    g_bTest := x > 300;
{END_IF}
```

**Operator defined
(type: <Identifier>)**

With this operator, the expression becomes `TRUE` if a data type is declared with the identifier <identifier>. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications `App1` und `App2`. Data type `DUT` is declared in `App1`, but not in `App2`.

```
{IF defined (type: DUT)}
(* the following code is only processed in App1*)
    bDutDefined := TRUE;
{END_IF}
```

**Operator defined
(pou: <POU name>)**

With this operator, the expression becomes `TRUE` if a function block or an action with name <pou-name> is present. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications `App1` and `App2`. The function block `CheckBounds` exists in `App1`, but not in `App2`.

```
{IF defined (pou: CheckBounds)}
(* the following code is only processed in App1 *)
    arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
{ELSE}
(* the following code is only processed in App2 *)
    arrTest[i] := arrTest[i]+1;
{END_IF}
```

**Operator defined
(task: <Identifier>)**

Not yet implemented!

With this operator, the expression becomes `TRUE` if a task is defined with the name <identifier>. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications `App1` and `App2`. Task `PlcProg_Task` is defined in `App1`, but not in `App2`.

```
IF defined (task: PLC_PRG_Task)}
(* the following code is only processed in App1 *)
```

```

    erg := plc_prg.x;
{ELSE}
(* the following code is only processed in App2 *)
    erg := prog.x;
{END_IF}

```

Operator defined
(resource: <Identifier>)

Not yet implemented!

With this operator, the expression becomes **TRUE** if there is a resource object with the name <identifier> for the application. Otherwise, **FALSE** is returned.

Example:

Prerequisite: There are the two applications App1 and App2. There is a resource object glob_var1 of the Global Variable List App1, but not for App2.

```

{IF defined (resource:glob_var1)}
(* the following code is only processed in App1 *)
    gvar_x := gvar_x + ivar;
{ELSE}
(* the following code is only processed in App2 *)
    x := x + ivar;
{END_IF}

```

Operator defined
(IsSimulationMode)

With this operator, the expression becomes **TRUE** if the application runs on a simulated device, i.e. in simulation mode.

Also refer to

- [chapter 12.1 "Testing in simulation mode" on page 179](#)

Operator defined (IsLittleEndian)

With this operator, the expression becomes **FALSE** if the CPU is "BigEndian (Motorola byte order)".

Operator defined
(IsFPUSupported)

If the expression returns the value **TRUE**, then the code generator produces an FPU code (for the floating-point unit processor) when calculating with **REAL** values. Otherwise, IndraLogic emulates FPU operations. However, this is much slower.

Operator "hasvalue"
(RegisterSize, "<RegisterSize>")

<RegisterSize>: Size of a CPU register in bits

With this operator, the expression becomes **TRUE** if the size of a CPU register is equal to <register size>.

Possible values for <RegisterSize>

- 16 for 186 and C16x,
- 64 for X86-64 bit
- 32 for X86-32 Bit

Operator "hasvalue"
(PackMode, "<PackModeValue>")

The checked pack mode depends on the device description, not on the pragma that can be specified for individual DUTs.

Operator "hasattribute"
(pou: <POU name>, "<Attribute>")

This operator causes the expression to become **TRUE** if the attribute <attribute> is specified in the first line of the declaration section of the function block pou-name, otherwise **FALSE** is returned.

Example:

Prerequisite: There are the two applications App1 and App2. A function fun1 is defined in App1 and App2, but in App1, the attribute vision is assigned to it in addition.

In "App1":

```

{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
    i : INT;
END_VAR

```

```
VAR
END_VAR
```

In App2:

```
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

Pragma statement:

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in App1 *)
    ergvar := fun1(i);
{END_IF}
```

Also refer to

- [chapter "User-defined attributes" on page 379](#)

Operator "hasattribute"
(variable: <Variable>,
"<Attribute>")

This operator causes the expression to become **TRUE** if the attribute "<attribute>" is assigned to the variable using the statement {attribute "<attribute>"} in the line before the variable declaration. Otherwise, **FALSE** is returned

Example:

Prerequisite: There are the two applications App1 and App2. Variable g_globalInt is used in App1 and App2, but in App1, the attribute "DoCount" is assigned to it in addition.

Declaration of g_GlobalInt in App1

```
VAR_GLOBAL
{attribute 'DoCount'}
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

Declaration of g_GlobalInt in App2

```
VAR_GLOBAL
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

Pragma statement:

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
(* the following code is only processed in App1 *)
    g_globalInt := g_globalInt + 1;
{END_IF}
```

Also refer to

- [chapter "User-defined attributes" on page 379](#)

Operator "hastype"
(variable: <Variable>,
<TypeSpec>)

With this operator, the expression becomes **TRUE** if the variable <variable> is of the data type <type-spec>. Otherwise, **FALSE** is returned.

Possible data types for <TypeSpec>:

- LREAL
- REAL
- LINT

- DINT
- INT
- SINT
- ULINT
- UDINT
- UINT
- USINT
- TIME
- LWORD
- DWORD
- WORD
- BYTE
- BOOL
- STRING
- WSTRING
- DATE_AND_TIME
- DATE
- TIME_OF_DAY

Example:

Prerequisite: There are the two applications App1 and App2. Variable `g_multitype` is declared in App1 with the data type `LREAL` and in App2, with the data type `STRING`.

```
{IF (hastype (variable: g_multitype, LREAL))}
(* the following code is only processed in App1 *)
  g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
(* the following code is only processed in App2 *)
  g_multitype := 'this is a multitalent';
{END_IF}
```

Operator "hasvalue"
 (<DefineIdent>, "<CharString>")

With this operator, the expression becomes `TRUE` if a variable is defined with the identifier <define-ident> and has the value <char-string>. Otherwise, `FALSE` is returned.

Example:

Prerequisite: There are the two applications App1 and App2. The variable `test` is used in the applications App1 and App2. In App1, it obtains 1, in App2, it obtains 2.

```
{IF hasvalue(test, '1')}
(* the following code is only processed in App1 *)
  x := x + 1;
{ELSIF hasvalue(test, '2')}
(* the following code is only processed in App2 *)
  x := x + 2;
{END_IF}
```

Operator hasconstantvalue
 (<variable>, <literal expression>)

You can use this operator to query the declared value of a constant.

Example:

Prerequisite:

```
{IF hasconstantvalue(test,'1')}
(* the following code is only processed in App1 *)
  x := x + 1;
{ELSIF hasconstantvalue(test,'2')}
(* the following code is only processed in App2 *)
  x := x + 2;
{END_IF}
```

Operator "NOT" <Operator>

The expression becomes TRUE if the reciprocal of <operator> returns TRUE. <Operator> can be one of the operators described in this chapter.

Example:

Prerequisite: There are the two applications App1 and App2. PlcProg1 is in App1 and App2, POU CheckBounds is only in App1.

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
(* the following code is only processed in App2 *)
  bANDNotTest := TRUE;
{END_IF}
```

Operator <Operator> AND <Operator>

The expression becomes TRUE if both operators specified return TRUE. <Operator> can be one of the operators described in this chapter.

Example:

Prerequisite: There are two applications, App1 and App2.

PLCPRG1 exists in App1 and App2, the POU CheckBounds only in App1.

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
(* the following code is only processed in App1 *)
  bANDTest := TRUE;
{END_IF}
```

Operator <Operator> OR <Operator>

The expression returns TRUE if one of the specified operators returns TRUE. <Operator> can be one of the operators described here.

Example:

Prerequisite: There are two applications, App1 and App2.

PLCPRG1 exists in App1 and App2, the POU CheckBounds only in App1.

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
(* the following code is only processed in App1 and in App2 *)
  bORTest := TRUE;
{END_IF}
```

Operator (<Operator>)

() parenthesizes the operators.

Also refer to

- [chapter 9.6 "Using pragmas" on page 110](#)
- [chapter "User-defined attributes" on page 379](#)

17.6.4 Region pragma

The pragma is used to summarize multiple lines to one block in a text editor. The block can be provided with a name. Region pragmas can be nested among each other.

Code with region pragma: Expanded and collapsed view

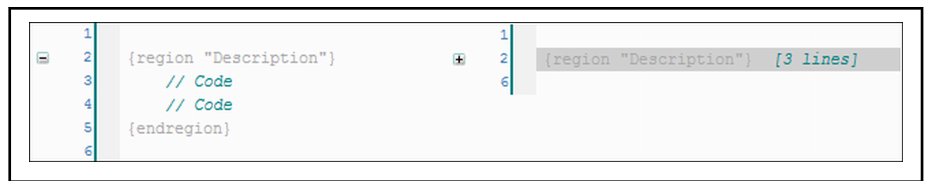


Fig. 17-91:

The pragma can be used with all declaration editors in the ST editor. Syntax highlighting can be adjusted in the options.

Also refer to:

[chapter "'Expand all' command" on page 584](#)

[chapter "Command 'Expand all'" on page 584](#)

17.7 Identifier

Rules for identifier specification

Rules for identifiers of a variable

- An identifier may not contain any spaces or special characters
- This means that IndraLogic is case-insensitive. VAR1 and var1 refer to the same variable.
- IndraLogic identifies underscores. For example, A_BCD and AB_CD are handled as two different identifiers.
Multiple consecutive underscores are not permitted.
- The length of an identifier is unlimited

Rules for multiple use of identifiers (namespaces)

- An identifier must not be declared twice locally
- An identifier may not be identical to a keyword (e.g., variable type).
- An identifier may be used multiple times. If a local variable has the same name as a global variable, the local variable has priority within the POU.
- A variable defined in a Global Variable List can have the same name as a variable defined in another GVL.

IndraLogic provides standard-extending functionalities with regard to the namespace and the scope of variables:

- Global namespace operator: An instance path that begins with a dot always opens a global namespace. If there is a local variable (e.g. `ivar`) that has the same name as a global variable, address the global variable with `.ivar`.
- The name of a Global Variable List can define the namespace uniquely for the included variables. Thus, variables with the same name can be declared in different Global Variable lists, but still be uniquely addressed by preceding the list name.

For example, `globlist1.ivar := globlist2.ivar; (* ivar from GVL globlist2 is copied to ivar in GVL globlist1 *)`

- Variables defined in the Global Variable List of a library integrated into the project can be addressed uniquely based on the syntax `<NamespaceLibrary>.<NameOfGVL>.<VariableName>`.

For example, `globlist1.ivar := lib1.globlist1.ivar (* ivar from GVL globlist1 in library lib1 is copied to ivar in GVL globlist1 *)`

- When inserting a library using the library manager, a namespace is also defined.

Thus, uniquely address library function blocks or a library variable via `<Namespace library>.<Function block name|Variable name>`.

Note that when libraries are nested, the namespaces of all libraries involved, have to be specified successively.

Example: If Lib1 is referenced by Lib0, the function block func in Lib1 is addressed using Lib0.Lib1.fun:

```
ivar := Lib0.Lib1.fun(4, 5); (* return value from
func is copied to variable ivar in the project *)
```

Also refer to

- [chapter 17.5 "Data types" on page 351](#)
- [chapter 17.2 "Variables" on page 280](#)

Recommendation

We recommend to apply the following rules in addition to the items considered specifically for the variables declaration. Thereby, the name specification is standardized the best possible.

Recommendations for variable names

The naming of variables should be related to the Hungarian notation.

For each variable, find a short, meaningful, English description for the basic name. The first letter of words in the base name is uppercase and all other letters are lowercase (for example, `FileSize`).

According to the data type of the variables, attach prefix(es) in lowercase letters in front of the basic name:

Data type	Prefix	Comment
BOOL	x*	We intentionally recommend x as the prefix for Boolean variables in order to distinguish them from identifiers of the data type BYTE. The prefix indicates the view of an IEC programmer.
	b	Reserved
BYTE	by	Bit string, not for arithmetic operations
WORD	w	Bit string, not for arithmetic operations
DWORD	dw	Bit string, not for arithmetic operations
LWORD	lw	Bit string, not for arithmetic operations
SINT	si	
USINT	usi	
INT	i	
UINT	ui	
DINT	di	
UDINT	udi	
LINT	li	
ULINT	uli	

Data type	Prefix	Comment
REAL	r	
LREAL	lr	
STRING	s	
WSTRING	ws	
TIME	tim	
LTIME	ltim	
TIME_OF_DAY	tod	
DATE_AND_TIME	dt	
DATE	date	
ENUM	e	
POINTER	p	
ARRAY	a	
Enumeration	e	

*: x is used intentionally as prefix for Boolean variables to differentiate them from BYTE and to accommodate IEC programmers.

Examples

```
VAR
    bySubIndix: BYTE;
    xFlag: BOOL;
    udiCounter: UDINT;
END_VAR
```

Identifier for	Description	Example
Nested declarations	The prefixes are attached in the order of declaration:	pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
function block instance Variable of user-defined data type	Prefix: Abbreviation for the name of the function block or data type.	cansdoReceivedTelegram: CAN_SDOTelegram; TYPE CAN_SDOTelegram: (* prefix: sdo *) STRUCT wIndex: WORD; bySubIndex: BYTE; byLen: BYTE; aby: ARRAY [0..3] OF BYTE; END_STRUCT END_TYPE
Local constant Local constant variable	Prefix: c_ followed by type prefix and variable name	VAR CONSTANT c_uiSyncID: UINT := 16#80; END_VAR

Reference "Programming"

Identifier for	Description	Example
Global variable	An additional prefix is added to the library prefix: <code>g_</code>	<pre>VAR_GLOBAL CAN_g_iTest: INT; END_VAR</pre>
Global constants Global constant variable	An additional prefix is added to the library prefix: <code>gc_</code>	<pre>VAR_GLOBAL CONSTANT CAN_gc_dwExample: DWORD; END_VAR</pre>

Recommendations for variable names in IndraLogic 2G libraries

Identifier for	Description	Example
Variable names	Corresponds to the previous description on variable names, with the exception that global variables and constants do not require library prefixes, as the namespace replaces the function.	<pre>g_iTest: INT; // Declaration CAN.g_iTest; // Implementation; call in the program</pre>

Recommendations for identifiers for user-defined data types (DUT)

Identifier for	Description	Example
Structures	Library prefix followed by an underscore and a short, informative description of the structure. The corresponding prefix for variables created with this structure should follow as a comment directly after the colon.	<pre>TYPE CAN_SDOTelegram : (* prefix: sdo *) STRUCT wIndex: WORD; bySubIndex: BYTE; byLen: BYTE; abyData: ARRAY [0..3] OF BYTE; END_STRUCT END_TYPE</pre>
Enumerations	Library prefix followed by an underscore and the identifier in uppercase letters. Please note: In past IndraLogic versions, ENUM values > 16#7FFF caused errors, as they were not automatically converted into INT. Therefore, ENUMs should always be defined with correct INT values.	<pre>TYPE CAL_Day : (CAL_MONDAY, CAL_TUESDAY, CAL_WEDNESDAY, CAL_THURSDAY, CAL_FRIDAY, CAL_SATURDAY, CAL_SUNDAY); Declaration: eToday: CAL_Day;</pre>

Recommendations for identifiers for user-defined data types (DUT) in IndraLogic 2G libraries

Identifier for	Description	Example
DUT names in IndraLogic 2G libraries	The namespace replaces the function of the library prefix. Therefore, it is omitted. Enumeration values are also defined without a library prefix.	Library with namespace "CAL" <pre> TYPE Day : (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY) ; Declaration: eToday: CAL.Day; Use in the application IF eToday = CAL.Day.MONDAY THEN </pre>

**Recommendations for identifiers
for POU: Functions, function
blocks and programs**

Identifier for	Description	Example
POUs: Functions, function blocks and programs	Library prefix followed by an underscore and a short, informative POU name. As for variables, the first letter of each word should be in uppercase letters, all others, in lowercase letters. It is recommended to use a combination of a verb and a noun in the POU name. For function blocks, the respective prefix for created instances should follow the name as a comment.	<pre> FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *) </pre>
Actions	Only actions called by the function block beginning with <code>prv_</code>. Otherwise, actions do not contain prefixes.	

**Recommendations for identifiers
for IndraLogic 2G libraries**

Identifier for	Description	Example
POUs:	The namespace replaces the function of the library prefix. Therefore, it is omitted.	FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *)
Methods	Only methods called by the function block beginning with <code>prv_</code> . Otherwise, methods do not contain prefixes.	
Interface Interface	Beginning with "I"	ICANDevice

Recommendations for identifiers for visualizations



Ensure that a visualization is not named like another function block in the project, as problems can result when changing visualizations.

Also refer to

- [chapter 9.2 "Declaring variables " on page 83](#)
- [chapter 17.5 "Data types" on page 351](#)
- [chapter 17.2 "Variables" on page 280](#)

17.8 Shadowing rules

In IndraLogic, it is generally allowed to use the same identifier for different elements. For example, a POU and a variable can be named the same. However, you should avoid this practice in order to prevent confusion.

Negative example

In the following code section, a local function block instance has the same name as a function:

Program:

```
FUNCTION YYY : INT
;
END_FUNCTION

FUNCTION_BLOCK XXX
;
END_FUNCTION_BLOCK

PROGRAM PLC_PRG
VAR
    YYY : XXX;
END_VAR
YYY();
END_PROGRAM
```

In such a case as this, it is unclear whether the instance or the function is called in the program.

To make sure that names are always unique, you should follow naming conventions, such as certain prefixes for variables. Rules for assigning identifiers can be found in the "Identifiers" chapter of the help.

Naming conventions can be checked automatically using the static code analysis of IndraLogic. Static code analysis could also detect the duplicate use of the name YYY and report it as an error.

The consistent use of the attribute `qualified_only` for enumerations and global variable lists and the use of qualified libraries can also prevent ambiguous situations.

shadowing The compiler does not report any errors or warnings if the same identifier is used for different elements. Instead, the compiler searches the code in a specific order for the declaration of the identifier. If a declaration is found, then the compiler does not search for any other declarations elsewhere. If other declarations do exist, then they are "shadowed" for the compiler. The following section describes the shadowing rules (that is, the search order that the compiler uses when searching for the declaration for identifiers). The section "Ambiguous access and qualified access" provides ways to prevent ambiguous access and bypass shadowing rules.

Search order in the application When the compiler encounters a single identifier in the code of an application, it searches for the corresponding declaration in the following order:

1. Local variables of a method
2. Local variables in the function block, program, or function, and in any basic function blocks
3. Local methods of the POU
4. Global variables in the application, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
5. Global variables in a parent application, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
6. Global variables in referred libraries when neither the library nor the variable list requires qualified access
7. POU or type names of the application (i.e. Names of global variable lists, function blocks, etc.)
8. POU or type names of a parent application
9. POU or type names of a library
10. Namespaces of locally referred libraries and libraries that are published by libraries
11. Global variables in the `POUs` view, unless the `qualified_only` attribute is set in the variable list where they are declared
12. POU or type names of the `POUs` view (i.e. Names of global variable lists, function blocks, etc.)



Libraries that are inserted in the Library Manager of the `POUs` view are mirrored in the Library Manager in all applications in the project with the appropriate placeholder resolution. These libraries then form a common namespace with the libraries in the application. Therefore, there is no shadowing of libraries in the pool by libraries in the application.

Search order in the library When the compiler encounters a single identifier in the code of a library, it searches for the corresponding declaration in the following order:

1. Local variables of a method
2. Local variables in the function block, program, or function, and in any basic function blocks
3. Local methods of the POU
4. Global variables in the local library, if the `qualified_only` attribute is not set in the variable list where the global variables are declared
5. Global variables in referenced libraries when neither the library nor the variable list requires qualified access

6. POU or type names of the local library (i.e. Names of global variable lists, function blocks, etc.)
7. POU or type names of a referenced library
8. Namespaces of locally referenced libraries and libraries that are published by locally referenced libraries

Ambiguous access and qualified access

Despite the mentioned search orders, ambiguous access can still occur. For example, this is the case when a variable with the same name exists in two global variable lists that do not require qualified access. Such a case is reported by the compiler as an error (for example: ambiguous use of the name `XXX`).

This kind of ambiguous use can be made unique by means of qualified access, for example by accessing via the name of the global variable list (example: `GVL.XXX`).

Qualified access can also always be used to avoid shadowing rules.

Examples:

- The name of the global variable list can be used to uniquely access a variable in the list.
- The name of a library can be used to uniquely access elements in the library.
- The `THIS` pointer can be used to uniquely access variables in a function block, even if a local variable with the same name exists in a method of the function block.

To find the declaration location of an identifier at any time, use the command **Edit ► Browse ► Go to Definition**. This procedure can be especially helpful if the compiler produces an apparently obscure error message.

Searching in instance paths

The search orders described above do not apply to identifiers that exist as components in an instance path or to identifiers that are used as inputs in calls.

For access of the following type `yy.component`, it depends on the entity described by `yy` where the declaration of `component` is searched for.

If `yy` denotes a variable with a structured data type (that is, type `STRUCT` or `UNION`), then `component` is searched for in the following order:

- Local variables of the function block
- Local variables of the base function block
- Methods of the function block
- Methods of the base function block

If `yy` denotes a global variable list or a program, then `component` is searched for in this list only.

If `yy` denotes a namespace of a library, then `component` is searched for in this library exactly as described in the section above "Search order in the library".

Only in the second instance does the compiler decide whether access to the found element is granted (that is, whether the variable is only locally accessible, or whether a method is private). If access is not allowed, an error is issued.

Also refer to:

- [chapter 17.7 "Identifier" on page 419](#)
- [chapter "Attribute "qualified_only" on page 408](#)

- [chapter 17.2.15 "THIS" on page 293](#)

17.9 Keywords

In all editors, you must capitalize keywords that for example denote scopes, data types, or operators.

Keywords cannot be used as variable names.

Examples

Program:

```
VAR
END_VAR
BOOL_TO_INT
IF
THEN
ELSE
LTIME
MUL
XOR
PERSISTENT
PROGRAM
```

IndraLogic checks the correct use of keywords automatically and underlines errors immediately after the input.



If IndraLogic generates implicit code, variables and functions are generally provided with a name containing "__" (mostly starting with it).

The use of double underscores is automatically avoided in implementations.

Thus, no conflicts can arise between system-internal identifiers and identifiers specified by the programmer.

The following keywords are used in the IndraLogic export format. Thus, they must not be used as identifiers:

- ACTION
- END_ACTION
- END_FUNCTION
- END_FUNCTION_BLOCK
- END_PROGRAM

The following keywords also apply:

- VAR_ACCESS
- READ_ONLY
- READ_WRITE
- PARAMS

17.10 'FB_Init', 'FB_Reinit' and 'FB_Exit' methods

These two methods can be used to influence the initialization of function block variables and to influence the behavior before exiting function blocks.



The type of the return value for the implicit methods is `BOOL`. The value is not evaluated by the system, but the type should not be changed.

`FB_Init` is always implicitly available and is always called during the initialization. To specifically influence the initialization, declare the `FB_Init` explicitly and supplement the existing standard initialization code.

`FB_Reinit` has to be implemented explicitly. If this method is available, it is called after the instance of the respective function block was copied. This is done during an online change, after changes to the function block declaration (signature change) to reinitialize the new instance module. When re-initializing the basic implementation of the function block, call `FB_Reinit` explicitly for this function block.

`FB_Exit` has to be implemented explicitly. If an implementation is available, the method is called before the function block instance code is removed from the control (implicit call).

In the following, several use cases of this method are described for different operations.

Operation "First download"

When downloading an application to the control with factory settings, the storage places of all variables have to be set to the required initial state. Thus, the desired values are assigned to data ranges of function blocks. By explicitly implementing `FB_Init` for function blocks, it is possible to respond specifically to that situation in the application code. This operating case can be detected by assigning the method parameters `bInCopyCode` (`FALSE`) and `bInitRetains` (`TRUE`). (See Operation "Online change" and Operation "New download".)

Operation "Online change"

In the online change, the methods `FB_Exit`, `FB_Init` and `FB_Reinit` can be used to influence the initialization of function block instances. In the online change, the changes made at the application in offline mode are identified on the running control. Thus, the "old" instances of the function blocks are replaced as quickly as possible by their "new" siblings. If no changes were made to the declaration part of a function block, but only to the implementation part before logging in to an application, data areas are not replaced. Only code blocks are replaced. The methods `FB_Exit`, `FB_Init` and `FB_Reinit` are thus not called!



If changes were made in the declaration of a function block causing the copying previously described, the message on "possibly unintended effects" can be output during online change. A list of all instances to be copied are given in the "Details" message box.

Whether an online change is currently executed can be detected in the code of the methods `FB_Init` by evaluating the values of the parameters `bInCopyCode` (`TRUE`).

There are the following subsequent calls during an online change:

1. `FB_exit`

```
old_inst.fb_Exit(bInCopyCode := TRUE);
```

`FB_Exit` can be used to trigger cleanup before copying when exiting the "old" instance. Data can be prepared for subsequent copying and the state of the "new" instance can be affected. Other parts of the application can be informed on the upcoming change in position in the memory. Especially note variables of the types `POINTER` and `REFERENCE`. It is possible that they do not reference anymore to the required memory locations after an online change. Interface variables (`INTERFACE`) are handled separately by the compiler and adjusted during the online change. External resources such as sockets, files or other handles can

possibly applied unchanged by the new instance. They do not have to be handled separately during the online change. (See "Operation "New download"")

2. FB_Init

```
new_inst.FB_Init(bInitRetains := FALSE, bInCopy-
Code := TRUE);
```

FB_Init is called before the copy operation and can be used in order to execute specific operations for the online change. For example, variables can thus be initialized accordingly at the "new" position in the memory or other parts of the application can be informed about the new position of specific variables in the memory.

3. Copy operation copy

```
copy(&old_inst, &new_inst);
```

Existing values are retained. They are copied from the old instance to the new instance.

4. FB_Reinit

```
new_inst.FB_Reinit();
```

This method is called after copying is completed and sets the variables of the instance to defined values. For example, variables can thus be initialized accordingly at the "new" position in the memory or other parts of the application can be informed about the new position of specific variables in the memory. The implementation has to be independent of the online change. This method can be called any time from the application if a function block instance should be returned to its original state.



Use the {attribute 'no_copy'} attribute to prevent copying of a single variable of the function block during online change. The variable always retains the initial value.

Operation "New download"

When downloading the application, it is possible that an already existing application is replaced on the control. Thus, the memory space has to be controlled and released for the existing function blocks. Use the FB_Exit method to implement the required steps. External resources (such as SOCKET or FILE handles) can thus go into a defined state.

Detect this operating case by verifying if the parameter bInCopyCode = FALSE in the FB_Exit method.

Operation "Application start"

Before the first cycle of the tasks of an application, the initial assignments are processed.

Example

```
T1 : TON := (PT:=t#500ms);
```

Such assignments are only executed after calling FB_Init. To check the effects of these assignments, a function block or a message of a function block may be provided with the {attribute 'call_after_init'}. The attribute has to be inserted via the declaration part of the function block body as well as via the declaration part of the respective method. A function block extending another function block that uses the {attribute 'call_after_init'} has to be provided with that attribute as well. For reasons of comprehensibility, it is recommended to overwrite the corresponding method with the same name, the same signature and the same attribute. SUPER^.MyInit has to be called. The name of the method can be freely selected. (Exceptions: FB_Init, FB_Reinit and FB_Exit). The method is

called after processing the initial assignments and before starting the tasks of an application and can thus respond to the specifications of the user.

However, note the following when using `FB_Init` or `{attribute 'call_after_init'}`: Finding errors in the methods `FB_Init` or in methods with the `{attribute 'call_after_init'}` is difficult, as the setting of breakpoints might not have the desired effect.



If the execution reaches the explicitly-defined initialization code, the function block instance is already completely initialized via the implicit initialization code. No `SUPER^.FB_Init` call must be executed



`FB_Init` replaces the `INI` operator used in IndraLogic 1.x. This method cannot be compared with a construct of a constructor known from C#, C++ or Java. This has consequences for function blocks that extend other function blocks. (See below "Derived function blocks")

Interface of the `FB_Init` method

```
METHOD FB_Init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // TRUE: the Retainvariables are initialized
    (reset warm / reset cold)
    bInCopyCode : BOOL; // TRUE: the instance will be copied to the
    copy-code afterward(online change)
END_VAR
```

Additional function block inputs can be declared in an `FB_Init` method. Set these inputs in the declaration of the function block instance.

Example

`FB_Init` method for a function block `serialdevice`

```
METHOD PUBLIC FB_Init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // initializing of retain variable
    bInCopyCode : BOOL; // instance is copied to copy code
    iCOMnum : INT; // additional input: number of the COM interface,
    that is to be observed
END_VAR
```

Instantiation of the function block `serialdevice`:

```
com1 : serialdevice(iCOMnum:=1);
com0 : serialdevice(iCOMnum:=0);
```

Interface of the `FB_Reinit` method

```
METHOD FB_Reinit : BOOL
```

Interface of the `FB_Exit` method

There is a mandatory parameter `bInCopyCode`.

```
METHOD FB_Exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // TRUE: the exit method is called in order to leave the
    instance which will be copied afterwards (online change).
END_VAR
```

Behavior in derived function blocks

If a function block is derived from another function block, the `FB_Init` method of the derived function block has to define the same parameters as the `FB_Init` method of the basic function block. However, further parameters can be added to set up a special initialization for the instance.

Example

The MainFB, SubFB and SubSubFB function blocks derive from each other. The following applies: SubFB EXTENDS MainFB and SubSubFB EXTENDS SubFB.

Call sequence of the methods FB_Exit and FB_Init:

1. fbSubSubFb.FB_Exit (...);
2. fbSubFb.FB_Exit (...);
3. fbMainFb.FB_Exit (...);
4. fbMainFb.FB_Init (...);
5. fbSubFb.FB_Init (...);
6. fbSubSubFb.FB_Init (...);

Also refer to

- [chapter ""Method" object " on page 532](#)
- [chapter "Attribute 'call_after_init'" on page 380](#)
- [chapter "Attribute 'no_copyon page 399](#)
- [chapter 17.2.14 "SUPER" on page 291](#)

17.11 Compiler error messages

[Compiler errors C0001](#)

[Compiler errors C0004](#)

[Compiler errors C0007](#)

[Compiler errors C0010](#)

[Compiler errors C0015](#)

[Compiler errors C0022](#)

[Compiler errors C0027](#)

[Compiler errors C0032](#)

[Compiler errors C0038](#)

[Compiler errors C0041](#)

[Compiler errors C0045](#)

[Compiler errors C0048](#)

[Compiler errors C0061](#)

[Compiler errors C0066](#)

[Compiler errors C0070](#)

[Compiler errors C0075](#)

[Compiler errors C0080](#)

[Compiler errors C0084](#)

[Compiler errors C0089](#)

[Compiler errors C0094](#)

[Compiler errors C0098](#)

[Compiler errors C0104](#)

[Compiler errors C0117](#)

[Compiler errors C0002](#)

[Compiler errors C0005](#)

[Compiler errors C0008](#)

[Compiler errors C0011](#)

[Compiler errors C0018](#)

[Compiler errors C0023](#)

[Compiler errors C0030](#)

[Compiler errors C0035](#)

[Compiler errors C0039](#)

[Compiler errors C0043](#)

[Compiler errors C0046](#)

[Compiler errors C0049](#)

[Compiler errors C0062](#)

[Compiler errors C0068](#)

[Compiler errors C0072](#)

[Compiler errors C0076](#)

[Compiler errors C0081](#)

[Compiler errors C0086](#)

[Compiler errors C0090](#)

[Compiler errors C0096](#)

[Compiler errors C0101](#)

[Compiler errors C0114](#)

[Compiler errors C0118](#)

[Compiler errors C0003](#)

[Compiler errors C0006](#)

[Compiler errors C0009](#)

[Compiler errors C0013](#)

[Compiler errors C0020](#)

[Compiler errors C0026](#)

[Compiler errors C0031](#)

[Compiler errors C0037](#)

[Compiler errors C0040](#)

[Compiler errors C0044](#)

[Compiler errors C0047](#)

[Compiler errors C0050](#)

[Compiler errors C0064](#)

[Compiler errors C0069](#)

[Compiler errors C0074](#)

[Compiler errors C0077](#)

[Compiler errors C0082](#)

[Compiler errors C0087](#)

[Compiler errors C0091](#)

[Compiler errors C0097](#)

[Compiler errors C0102](#)

[Compiler errors C0116](#)

[Compiler errors C0119](#)

Reference "Programming"

Compiler errors C0120	Compiler errors C0122	Compiler errors C0125
Compiler errors C0126	Compiler errors C0130	Compiler errors C0131
Compiler errors C0132	Compiler errors C0136	Compiler errors C0139
Compiler errors C0140	Compiler errors C0141	Compiler errors C0142
Compiler errors C0143	Compiler errors C0144	Compiler errors C0145
Compiler errors C0149	Compiler errors C0161	Compiler errors C0162
Compiler errors C0168	Compiler errors C0169	Compiler errors C0174
Compiler errors C0175	Compiler errors C0177	Compiler errors C0178
Compiler errors C0182	Compiler errors C0185	Compiler errors C0189
Compiler errors C0190	Compiler errors C0195	Compiler errors C0196
Compiler errors C0197	Compiler errors C0198	Compiler errors C0199
Compiler errors C0201	Compiler errors C0203	Compiler errors C0204
Compiler errors C0205	Compiler errors C0206	Compiler errors C0208
Compiler errors C0211	Compiler errors C0212	Compiler errors C0216
Compiler errors C0217	Compiler errors C0218	Compiler errors C0219
Compiler errors C0221	Compiler errors C0222	Compiler errors C0224
Compiler errors C0227	Compiler errors C0228	Compiler errors C0234
Compiler errors C0235	Compiler errors C0236	Compiler errors C0237
Compiler errors C0238	Compiler errors C0239	Compiler errors C0240
Compiler errors C0241	Compiler errors C0242	Compiler errors C0243
Compiler errors C0380	Compiler errors C0509	Compiler errors C0511

Tab. 17-15:

17.11.1 Compiler errors C0001

Message:	Constant <constant value> too large for type <data type>
Possible error cause:	A "typed" constant is too large for the specified data type or a constant is too large for each possible data type.
Error correction:	Use smaller constants or an appropriate data type for a typed constant.
<i>Example of the error:</i>	

```

PROGRAM PLC_PRG
VAR
    test1: INT;
    test2: INT;
    test3: LREAL;
END_VAR

test1 := 12345678912345566991923939292939911;
test2 := INT#123456;
test3 := 10E500;

--> C0001: Constant '12345678912345566991923939292939911' is too big for type 'ANY_INT'
--> C0001: Constant 'INT#123456' is too big for type 'INT'
--> C0001: Constant '10E500' is too big for type 'ANY_REAL'

```

17.11.2 Compiler errors C0002

Message:	<operator 1> or <operator 2> expected instead of <tag>
Possible error cause:	Syntax error

Error correction: Use the correct syntax

Example of the error:

```
PROGRAM PLC_PRG
Fun(1;
```

```
--> C0002:  ',' or ')' expected instead of ';'

```

17.11.3 Compiler errors C0003

Message: <value> is not a valid bit number for <variable>

Possible error cause: Trying to access a bit outside the data type range

Error correction: Use a bit value for the bit access that is smaller than the number of bits in the data type of the variable

Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: WORD;
    test2: BOOL;
END_VAR
```

```
test1 := test2.17;
```

```
--> C0003:  '17' is not a valid bit number for 'w'

```

17.11.4 Compiler errors C0004

Message: <variable> is not a component of <structure>

Possible error cause: Component access with "." to a variable that is not a structured value or does not exist as a component of the structure

Error correction: Access a defined component or change the definition in the data type that is accessed. The input assistance "List components" provides all valid accesses to this position.

Example of the error:

```
TYPE DUT:
STRUCT
    x, y : INT;
END_STRUCT
END_TYPE
```

```
PROGRAM PLC_PRG
VAR
    test1 : DUT;
    test2 : INT;
END_VAR
```

```
test2 := test1.z;
```

```
--> C0032:  Type 'Unknown type: 'test1.z'' cannot be converted to type 'INT'
--> C0004:  'z' is to a component of 'DUT'

```

17.11.5 Compiler errors C0005

Message: Constant overflow in address <address>

Possible error cause: At least one component in the address is not compatible with a 32 bit integer value

Error correction: Use a valid address expression

Example of the error:

```
PROGRAM PLC_PRG
VAR
```

```

      X: BYTE;
END_VAR

X := %QB555555555555;

--> C0005: Constant overflow in address '%??'
```

17.11.6 Compiler errors C0006

Message: <operator> instead of <token>
Possible error cause: Syntax error
Error correction: Use the correct syntax
Example of the error:

```

PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF bTest
    x := 9;
END_IF

--> C0006: 'THEN' expected instead of 'x'
```

17.11.7 Compiler errors C0007

Message: Expression expected instead of <token>
Possible error cause: Syntax error
Error correction: Use the correct syntax
Example of the error:

```

PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF THEN
    x := 9;
END_IF

--> C0007: Expression expected instead of 'THEN'
```

17.11.8 Compiler errors C0008

Message: Unexpected "end-of-file" found: <operator 1>, <operator 2> or <operator 3> expected
Possible error cause: Syntax error
Error correction: Use the correct syntax
Example of the error:

```

PROGRAM PLC_PRG
VAR
    x: INT;
    bTest : BOOL;
END_VAR

IF bTest THEN
    x := 9;

--> C0008: Unexpected end-of-file found: 'ELSIF', 'ELSE' or 'END_IF' expected
```

17.11.9 Compiler errors C0009

Message: Unexpected token <token> found
Possible error cause: Syntax error
Error correction: Use the correct syntax
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

END_FOR;

--> C0009: Unexpected token 'END_FOR' found
```

17.11.10 Compiler errors C0010

Message: Unexpected end-of-file found: <token> expected
Possible error cause: Syntax error
Error correction: Use the correct syntax
Example of the error:

```
PROGRAM PLC_PRG
VAR
  i: INT;
END_VAR

FOR i := 0 TO 2 DO
;

--> C0010: Unexpected end-of-file 'END_FOR' found
```

17.11.11 Compiler errors C0011

Message: No CASE label found
Possible error cause: Syntax error in a CASE statement. A statement in a CASE statement is not assigned to a CASE label.
Error correction: Insert a CASE label
Example of the error:

```
PROGRAM PLC_PRG
VAR
  i: INT;
  x: INT;
END_VAR

CASE i OF
  x := 9;
END_CASE

--> C0011: No 'CASE' label found
```

Error correction:

```
CASE i OF
0:
  x := 9;
END_CASE
```

17.11.12 Compiler errors C0013

Message: At least one statement is expected

- Possible error cause:** At least one statement is expected at some positions in the code.
For example, in the `THEN` and in the `ELSE` part of an `IF` statement, or in the body of a `FOR` loop.
- Error correction:** Insert at least one statement at the selected position.
It is enough to write a blank statement `;`.
- Example of the error:*

```
PROGRAM PLC_PRG
VAR
    bTest: BOOL;
END_VAR

IF bTest THEN
END_IF

--> C0013: At least one statement is expected
```

17.11.13 Compiler errors C0015

- Message:** Counter initialization expected
- Possible error cause:** Syntax error in a `FOR` loop
The counter variable is not initialized correctly.
- Error correction:** Ensure the correct syntax of the `FOR` loop:
Program:

```
FOR i := 0 TO 10 DO
;
END_FOR
```

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
END_VAR

FOR i TO 10 DO
;
END_FOR

--> C0015: Counter initialization expected
```

Error correction:

```
FOR i := 0 TO 10 DO
;
END_FOR
```

17.11.14 Compiler errors C0018

- Message:** `<expression>` is not a valid assignment target
- Possible error cause:** An expression with no write permission is on the left side of an assignment, e.g. a constant
- Error correction:** Assign only to variables that have write access
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i: INT;
END_VAR
VAR CONSTANT
    j: INT := 0;
```

```
END_VAR
j := i;
--> C0018: 'j' is not a valid assignment target
```

17.11.15 Compiler errors C0020

Message: <statement> is not a valid statement
Possible error cause: Syntax error: For example, a character was omitted or a character too many
Error correction: Ensure that the syntax is correct
Example of the error:

```
PROGRAM PLC_PRG
VAR
    x : INT;
END_VAR
x = 2;
--> C0020: '(x = 2); ' is no valid statement
```

Error correction:

Example:
x := 2;

17.11.16 Compiler errors C0022

Message: <operator> requires exactly <number of operands> operands
Possible error cause: Too many or too few operands are assigned to an operator
Error correction: Assign the required number of operands to the operator
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    pt: POINTER TO INT;
END_VAR
pt := ADR(i,1);
--> C0022: 'ADR' needs exactly '1' operands
```

Error correction:

Example:
pt := ADR(i);

17.11.17 Compiler errors C0023

Message: <operator> requires at least <number of operands> operands
Possible error cause: Too few operands are assigned to an operator
Error correction: Assign the required number of operands to the operator
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i := MUX(30,40);
```

--> C0023: 'MUX' needs at least '3' operands

Error correction:

Example:
i := MUX(30,40,50);

17.11.18 Compiler errors C0026

Message: Identifier expected instead of <invalid identifier>

Possible error cause: An invalid identifier is transmitted to a method

Error correction: Use valid identifiers

Example of the error:

```
METHOD 123
VAR_INPUT
END_VAR
```

--> C0243: The name used in the signature is not identical to the object name
--> C0026: Identifier expected instead of '123'

Error correction:

Example:
METHOD METH123

17.11.19 Compiler errors C0027

Message: Character string length expected after "("

Possible error cause: The length of the string is not specified

Error correction: Specify a string length in parentheses

Example of the error:

```
PROGRAM PLC_PRG
VAR
    str : STRING();
END_VAR
```

--> C0027: size of string expected after '('
--> C0006: ';;, :=, REF=, (or [' expected instead of ')'

Error correction:

Example:
str : STRING(100);

17.11.20 Compiler errors C0030

Message: Direct address expected after "AT" instead of <identifier>

Possible error cause: Either an invalid address or no address is assigned after "AT"

Error correction: Specify a valid address

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i AT ABC : INT;
END_VAR
```

--> C0030: Direct Address expected after 'AT' instead of 'ABC'

Error correction:

Example:
i AT %IW0 : INT;

17.11.21 Compiler errors C0031

Message: Type definition expected instead of <no data type>
Possible error cause: An invalid type definition is assigned to the identifier
Error correction: Specify a valid type definition
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : 0;
END_VAR
```

--> C0031: Type definition expected instead of '0'

Error correction:

Example:
i : INT;

17.11.22 Compiler errors C0032

Message: Unable to convert type <type 1> to type <type 2>
Possible error cause: A variable is assigned to another variable with an incompatible type
Error correction: Use a type conversion
Example of the error:

```
PROGRAM PLC_PRG
VAR
    test1: INT;
    test2: STRING;
END_VAR
```

```
test1 := test2;
```

-->C0032: Type 'STRING' cannot be converted to type 'INT'

Error correction:

Example:
test1 := TO_INT(test2);

17.11.23 Compiler errors C0035

Message: Program name, function or function block instance expected instead of <invalid function>
Possible error cause: A function is called that does not exist
Error correction: Ensure that only existing program names, functions, and function block instances are called
Example of the error:

```
PROGRAM PLC_PRG
VAR
```

Reference "Programming"

```

END_VAR

PLC_PRG.METH1();

METHOD METH
VAR_INPUT
END_VAR

```

```

--> C0004: 'METH1' is no component of 'PLC_PRG'
--> C0035: Program name, function or function block instance expected instead of 'PLC_PRG.METH1'

```

Error correction:

Example:

```

PLC_PRG.METH();

```

17.11.24 Compiler errors C0037

Message: <invalid input> is not an input of <function name>
Possible error cause: A local variable is defined in a function call
Error correction: Declare the variable as an input parameter
Example of the error:

```

PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i := TEST(iVar := 1);

FUNCTION TEST : INT
VAR
    iVar : INT;
END_VAR

```

```

--> C0037: 'iVar' is no input of 'TEST'

```

Error correction:

Example:

```

VAR_INPUT
    iVar : INT;
END_VAR

```

17.11.25 Compiler errors C0038

Message: <invalid output> is not an output of <function name>
Possible error cause: A local variable is handled as an output in a function call
Error correction: Declare the variable as an output parameter
Example of the error:

```

PROGRAM PLC_PRG
VAR
    i : INT;
    x : INT;
END_VAR
i := TEST(iVar => x);

FUNCTION TEST : INT
VAR
    iVar : INT;
END_VAR

```

```

--> C0038: 'iVar' is no output of 'TEST'

```


Error correction:

```
Example:
VAR OUTPUT
  _iVar : INT;
END_VAR
```

17.11.26 Compiler errors C0039

- Message:** VAR_IN_OUT <invalid variable> has to be assigned in the <function block name> call
- Possible error cause:** No IN_OUT variable is transmitted to a function block, requiring an IN_OUT variable
- Error correction:** Assign the IN_OUT variable

Example of the error:

```
PROGRAM PLC_PRG
VAR
  inst : FB;
END_VAR
inst();

FUNCTION_BLOCK FB
VAR_IN_OUT
  _inout : INT;
END_VAR
```

```
--> C0039: VAR_IN_OUT 'inout' must be assigned in call of 'FB'
```

Error correction:

```
Example:
inst(inout := i);
```

17.11.27 Compiler errors C0040

- Message:** Function <function name> requires exactly <number of inputs> inputs
- Possible error cause:** Too many or too few parameters are transmitted to the called function
- Error correction:** Transmit exactly as many parameters to the function as are expected

Example of the error:

```
PROGRAM PLC_PRG
VAR
  i : INT;
END_VAR
i := TEST(1,2);

FUNCTION TEST : INT
VAR_INPUT
  _IN: INT;
END_VAR
```

```
--> C0040: Function 'TEST' requires exactly '1' inputs
```

Error correction:

```
Example:
i := Test(1);
```

17.11.28 Compiler errors C0041

Message: VAR_IN_OUT parameter <parameter name> of <function name> requires a variable with write access as input

Possible error cause: The transmitted parameter is not a variable with write access but a constant, for example

Error correction: Transmit a VAR_IN_OUT parameter with write access to the function

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    x : INT;
END VAR
i := Test(31415);
```

```
FUNCTION TEST : INT
VAR_IN_OUT
    in_out: INT;
END_VAR
```

```
--> C0041: VAR_IN_OUT' parameter 'in_out' of 'TEST' needs variable with write access as input
```

Error correction:

Example:
i := Test(x);

17.11.29 Compiler errors C0043

Message: Incorrect formal parameter: <parameter name> expected at this position

Possible error cause: The parameters are assigned to the function explicitly in the wrong order

Error correction: Specify the parameters in the correct order

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END VAR
i := Test(iPar2 := 2, 5);
```

```
FUNCTION Test : INT
VAR_INPUT
    iPar1 : INT;
    iPar2 : INT;
END_VAR
```

```
--> C0043: Wrong formal parameter: 'iPar1' expected in this place
--> C0412: Multiple input assignments for parameter ''
```

Error correction:

Example:
i := Test(5, iPar2 := 2);

17.11.30 Compiler errors C0044

Message: Assignment to input missing for parameter <input variable name> in <function block name> call

Possible error cause: A parameter is transmitted although no input variable is declared

Error correction: Declare an input variable

Example of the error:

```

PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
inst(1);

```

```

FUNCTION BLOCK FB
VAR_INPUT
END_VAR

```

```
--> C0044: Assignment to input missing for parameter '1' in call of 'FB'
```

Error correction:

```

Example:
VAR_INPUT
    in : INT;
END_VAR

```

17.11.31 Compiler errors C0045

- Message:** Use of `THIS` is not valid this context
- Possible error cause:** `THIS` can be used only in a method, action, transition, or in the body of a function block This error message appears for all other positions.
- Error correction:** Use `THIS` in an allowed context only

Example of the error:

```

PROGRAM PLC_PRG
VAR
    test1: INT;
END_VAR

THIS^.test1 := 19;

--> C0018: 'THIS^.test1' is not a valid assignment target
--> C0062: 'THIS^' is not a structure variable
--> C0045: Use of 'THIS' is not allowed in this context

```

17.11.32 Compiler errors C0046

- Message:** Identifier <identifier name> not defined
- Possible error cause:** An identifier is used that is not declared
- Error correction:** Declare the variables you want to use

Example of the error:

```

PROGRAM PLC_PRG
VAR
END_VAR

i := 1;

--> C0018: 'i' is no valid assignment target
--> C0046: Identifier 'i' not defined

```

Error correction:

```

Example:
VAR

```

```

    i : INT;
END_VAR

```

17.11.33 Compiler errors C0047

- Message:** Cannot apply indexing with [] to an expression of type <data type>
- Possible error cause:** A data type that is not an array is indexed with []
- Error correction:** Index data types with [] only if they are declared as arrays
- Example of the error:*

```

PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i[1];

```

```
--> C0047: Cannot apply indexing with '[' to an expression of type 'INT'
```

17.11.34 Compiler errors C0048

- Message:** Data arrays require exactly <number> indices
- Possible error cause:** Too many or too few indexes are specified when using an array
- Error correction:** Specify as many indices as there are dimensions assigned to the array
- Example of the error:*

```

PROGRAM PLC_PRG
VAR
    arr1 : ARRAY[1..2,1..3] OF INT;
END_VAR
arr1[1] := 5;

```

```
--> C0048: Array requires exactly 2 indexes
```

Error correction

Example:

```
arr1[1,2] := 5;
```

17.11.35 Compiler errors C0049

- Message:** The constant index <index value> is outside the range from <start index> to <end index>
- Possible error cause:** An index is specified that is outside the size of the array
- Error correction:** Use only indexes that are within the size of the array
- Example of the error:*

```

PROGRAM PLC_PRG
VAR
    arr1 : ARRAY[1..2] OF INT;
END_VAR
arr1[3] := 1;

```

```
--> C0049: The constant index '3' is not within the range from '1' to '2'
```

Error correction

Example:

```
arr1[2] := 1;
```

17.11.36 Compiler errors C0050

Message: Bit access requires literal or symbolic integer constant
Possible error cause: No literal or an integer constant is specified in a bit access
Error correction: Use a literal or an integer constant

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    x : INT;
END_VAR

i.x := FALSE;

--> C0018: 'i.x' is no valid assignment target
--> C0050: Bitaccess requires literal or symbolic integer constant
```

Error correction

Example:
i := Test(x);

17.11.37 Compiler errors C0061

Message: Bit access to a function call is not permitted
Possible error cause: A bit access is executed to a function
Error correction: Use bit access only for supported data types

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

Test().2;

FUNCTION Test : INT
VAR_INPUT
END_VAR

--> C0061: Bitaccess on function call is not allowed
```

17.11.38 Compiler errors C0062

Message: <variable name> is not a structured variable
Possible error cause: A variable that is not a structure variable is treated like a structure variable
Error correction: Ensure that the variable is a structure variable

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pt : PUNKT;
    i : INT;
END_VAR

i.x := 1024;

TYPE Punkt :
STRUCT
    x : REAL;
    y : REAL;
END_STRUCT
```

Reference "Programming"

END_TYPE

--> C0018: 'i.x' is no valid assignment target
 --> C0062: 'Variable' is no structured variable

Error correction:

Example:
 pt.x := 1024;

17.11.39 Compiler errors C0064

Message: Dereferencing requires a pointer

Possible error cause: A variable that is not a pointer variable is "dereferenced"

Error correction: "Dereference" only variables that are pointer variables

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    pi : POINTER TO INT;
END_VAR
i^:=1;
```

--> C0018: 'i^' not a valid assignment target
 --> C0064: Dereferencing requires a pointer

Error correction:

Example:
 pi := ADR(i);
 pi^ := 1;

17.11.40 Compiler errors C0066

Message: Cannot compare type <data type> to type <data type>

Possible error cause: Two data types are compared which cannot be compared with each other

Error correction: Compare only data types that can be compared with each other

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    re : REAL;
    str: STRING;
    b : BOOL;
END_VAR
```

b := i > str;

--> C0066: Cannot compare type 'INT' with type 'STRING'

Error correction:

Example:
 b := i > re;

17.11.41 Compiler errors C0068

Message: Comparison for objects of type <data type> not possible

Possible error cause: Objects are being compared in which a comparison is not possible

Error correction: Compare only data types in which a comparison is possible, e.g. INT, REAL, etc

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    arr1 : ARRAY [1..2] OF INT;
    arr2 : ARRAY [1..2] OF INT;
END_VAR

b := arr1 > arr2;
```

--> C0068: Compare not possible on objects of type 'ARRAY [1..2]'

17.11.42 Compiler errors C0069

Message: Compare not possible on objects of type <data type> or <data type>

Possible error cause: Two different objects are being compared in which a comparison is not possible

Error correction: Compare only data types in which a comparison is possible, e.g. INT, REAL, etc

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    arr1 : ARRAY [1..2] OF INT;
    arr2 : ARRAY [1..3] OF INT;
END_VAR

b := arr1 > arr2;
```

--> C0069: Compare not possible on objects of type 'ARRAY [1..2]' or 'ARRAY [1..3]'

17.11.43 Compiler errors C0070

Message: INI operator requires function block instance or data unit type instance

Possible error cause: Neither a function block instance nor a DUT instance is applied to the INI operator

Error correction: Transmit only function block instances or DUT instances to the INI operator

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BOOL;
    inst : FB;
END_VAR

b := INI(b, TRUE);

FUNCTION_BLOCK FB
VAR
END_VAR
```

--> C0070: 'INI' operator needs function block instance or data unit type instance

Error correction:

Example:
b := INI(inst, TRUE);

17.11.44 Compiler errors C0072

Message: Operator <operator name> cannot be used for type <data type>
Possible error cause: An operator is applied to an incompatible type
Error correction: Apply operators only to compatible data types

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    str : STRING;
END_VAR

str := ABS(str);

--> C0072: Operator 'Abs' is not possible on type 'STRING'
```

Error correction:

Example:
i := ABS(i);

17.11.45 Compiler errors C0074

Message: Unexpected array initialization
Possible error cause: Syntax error during array initialization
Error correction: Correct the syntax

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : INT := [1,2,3,4,5,6];
END_VAR

--> C0074: Unexpected array initialisation
--> C0032: Cannot convert type 'Unknown type: [1,2,3,4,5,6]' to type 'INT'
```

Error correction:

Example:
arr1 : ARRAY [1..6] OF INT := [1,2,3,4,5,6];

17.11.46 Compiler errors C0075

Message: Too many initializers for array
Possible error cause: Too many values are specified for the size of the array
Error correction: The number of assigned values must correspond to the size of the array

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5,6];
END_VAR

--> C0075: Unexpected array initialisation
```

Error correction:

Example:
 arr1 : ARRAY [1..6] OF INT := [1,2,3,4,5,6];

17.11.47 Compiler errors C0076

Message: Unexpected structure initialization
Possible error cause: Syntax error in the structure initialization
Error correction: Ensure that the syntax is correct

Example of the error:

```
PROGRAM PLC_PRG
VAR
    st1 : INT := (p1 := 1);
END_VAR
```

```
--> C0076: Unexpected structure initialisation
--> C0032: Cannot convert type 'STRUCT(p1:=1)' to type 'INT'
--> C0046: Identifier 'p1' not defined
--> C0018: 'p1' is no valid assignment target
```

Error correction:

Example:
 st1 : STRUCT1 := (p1:=1,p2:=10);

17.11.48 Compiler errors C0077

Message: Unknown type: <data type>
Possible error cause: Invalid data type in the declaration (maybe a syntax error)
Error correction: Specify valid data types only

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INTEGER;
END_VAR
```

```
--> C0077: Unknown type: 'INTEGER'
```

Error correction:

Example:
 arr1 : ARRAY[1..2] OF STRUCT1 := (p1:=1,p2:=10);

17.11.49 Compiler errors C0080

Message: Function block <function block name> must be instantiated for access
Possible error cause: Missing function block instantiation
Error correction: Instantiate the function block

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

FB();
```

```
FUNCTION_BLOCK FB
VAR
END_VAR
```

```
--> C0080: Functionblock 'FB' must be instantiated to be accessed
```

Error correction:

```
Example:
VAR
    inst : FB;
END_VAR
inst();
```

17.11.50 Compiler errors C0081

Message: Unexpected pragma: <pragma name> without corresponding IF condition

Possible error cause: The IF condition is missing when using the pragma

Error correction: Complete the IF condition of the pragma

Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR

i := 5;
{END_IF}
```

```
--> C0081: Unexpected Pragma: 'END_IF' found without matching 'if'
```

Error correction:

```
Example:
{IF <expression>}
i := 5;
{END_IF}
```

17.11.51 Compiler errors C0082

Message: <invalid pragma> is not a valid pragma condition

Possible error cause: When using a pragma, an invalid expression is used in the IF condition

Error correction: Use valid pragma conditions

Example of the error:

```
VAR
    i : INT;
END_VAR

{IF abc}
i := 5;
{END_IF}
```

```
--> C0082: '!!!ERROR!!!' is no valid condition for pragma
```

Error correction:

```
Example:
{IF defined (abc)}
```

17.11.52 Compiler errors C0084

Message: <pragma operand> is not a valid pragma operand

Possible error cause: Syntax error
Error correction: Use a valid pragma operand
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

{IF defined(0)}
{END_IF}

--> C0084: 'defined(null)' is no valid operand for pragma
```

Error correction:

```
Example:
{IF defined (abc)}
```

17.11.53 Compiler errors C0086

Message: No definition found for <interface name> interface
Possible error cause: An undefined interface is used
Error correction: Define the interface
Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
END_VAR

--> C0086: No definition found for interface 'XY'
```

Error correction:

```
Example:
INTERFACE XY
```

17.11.54 Compiler errors C0087

Message: There is no implementation for method <method name> of the interface <interface name>
Possible error cause: One of the methods specified by the interface has not been provided by the implementing function block
Error correction: Implement all methods that are specified by the interface
Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

INTERFACE XY
METHOD METH1
VAR_INPUT
END_VAR

FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
```

```

END VAR
METHOD METH2
VAR INPUT
END_VAR

```

--> C0087: There is no implementation for method 'METH1' defined in interface 'XY'

17.11.55 Compiler errors C0089

- Message:** Interface of overridden method {0} of interface {1} does not match the declaration
- Possible error cause:** The signature of the implemented method does not match the signature of the method in the interface
- Error correction:** Make sure that the same return types and parameters are declared
- Example of the error:*

```

PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

INTERFACE XY
METHOD METH1
VAR INPUT
    iPar : INT;
END_VAR

FUNCTION_BLOCK FB IMPLEMENTS XY
VAR
END_VAR
METHOD METH1
VAR INPUT
END_VAR

```

--> C0089: Interface of overridden method 'METH1' of interface 'XY' does not match declaration

17.11.56 Compiler errors C0090

- Message:** No definition found for base class <function name>
- Possible error cause:** The function block specified as the base does not exist or is not a function block
- Error correction:** Use a function block as the base
- Example of the error:*

```

PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

FUNCTION_BLOCK FB EXTENDS POU
VAR
END_VAR

FUNCTION POU
VAR
END_VAR

```

--> C00090: No definition found for base class 'POU'

17.11.57 Compiler errors C0091

- Message:** Recursion in base function block list: <function name>
- Possible error cause:** A base function block is extended by itself
- Error correction:** Recursion in base function block lists is not possible

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
```

```
FUNCTION_BLOCK FB EXTENDS FB
VAR
END_VAR
```

```
--> C00091: Recursion in base function block list: FB -> FB
```

17.11.58 Compiler errors C0094

- Message:** Interface of overridden method <method name> of <function block name> does not match the declaration
- Possible error cause:** The signature of the method of the first interface does not match the signature of the method in the second interface, which is extended by the first.
- Error correction:** Adjust the signatures

Example of the error:

```
PROGRAM PLC_PRG
VAR
    xyz : XY2;
END_VAR
```

```
FUNCTION_BLOCK XY
METHOD METH1
VAR_INPUT
END_VAR
```

```
FUNCTION_BLOCK XY2 EXTENDS XY
METHOD METH1
VAR_INPUT
    iPar : BOOL;
END_VAR
```

```
--> C00094: Interface of the overridden method METH1 of interface XY doesn't match declaration
```

17.11.59 Compiler errors C0096

- Message:** Only one base function block must be defined in the EXTENDS list
- Possible error cause:** Two or more base function blocks are defined in the EXTENDS list
- Error correction:** Define only one base function block in the EXTENDS list

Example of the error:

```
PROGRAM PLC_PRG
VAR
    fb : FB;
END_VAR
```

```
FUNCTION_BLOCK FB EXTENDS FB2, FB3
VAR
END_VAR
```

```
FUNCTION_BLOCK FB2
VAR
END_VAR
```

```
FUNCTION_BLOCK FB3
VAR
END_VAR
```

```
--> C00096: Only one base function block may be defined in EXTENDS-list
```

17.11.60 Compiler errors C0097

Message: Multiple definition of variable <variable name> in the <function block name> function block and in the <base function block name> base

Possible error cause: A variable is declared with the same name in a function block and its base

Error correction: Use different variable names

Example of the error:

```
PROGRAM PLC_PRG
VAR
    fb : FB;
END_VAR
```

```
FUNCTION_BLOCK FB EXTENDS FB2
VAR
    i : INT;
END_VAR
```

```
FUNCTION_BLOCK FB2
VAR
    i : INT;
END_VAR
```

```
--> C00097: Duplicate definition of variable 'i' in function block 'FB' and in base 'FB2'
```

17.11.61 Compiler errors C0098

Message: The keyword "FUNCTIONBLOCK" is no longer supported
Instead, use "FUNCTION_BLOCK"

Possible error cause: Syntax error

Error correction: Use the keyword "FUNCTION_BLOCK"

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
```

```
FUNCTIONBLOCK FB
VAR
    END_VAR
```

```
--> C00098: The keyword "FUNCTIONBLOCK" is no longer supported.
--> Use "FUNCTION_BLOCK" instead.
```

17.11.62 Compiler errors C0101

Message: Data recursion: <recursion>

Possible error cause: Recursive data initialization over two function blocks

Error correction: Avoid recursions for data initialization

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst0 : FB1;
END_VAR
```

```
FUNCTION_BLOCK FB1
VAR
    inst1 : FB2;
END_VAR
```

```
FUNCTION_BLOCK FB2
VAR
    inst2 : FB1;
END_VAR
```

```
--> C0101: Data Recursion: FB1->FB2->FB1.
```

17.11.63 Compiler errors C0102

- Message:** Outside the retain memory: Variable <variable name>, <byte size> bytes
- Possible error cause:** More retain memory is used than is available on the PLC.
It is also possible that the retain memory is too fragmented due to incremental builds.
- Error correction:** Use the "Clean" command for fragmenting the memory.
This will force the reallocation of all data at the next build.

17.11.64 Compiler errors C0104

- Message:** "Insufficient global data memory: Variable <variable name>, <byte size> bytes
- Possible error cause:** More memory for data is used than is available on the PLC.
It is also possible that the memory is too fragmented due to incremental builds.
- Error correction:** Use the "Clean" command for fragmenting the memory.
This will force the reallocation of all data at the next build.

17.11.65 Compiler errors C0114

- Message:** Invalid target <jump label> for JMP
- Possible error cause:** Syntax error or typographical error in the JMP target
- Error correction:** Correct the typographical or syntax error
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP 0;
```

```
--> C0114: Invalid destination 0 for 'JMP'
```

17.11.66 Compiler errors C0116

- Message:** Label <jump label> defined multiple times
- Possible error cause:** A jump label is defined multiple times
- Error correction:** Define each label one time only once
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP label;
label:
```

```
label:
```

```
--> C0116: The label 'LABEL' is a duplicate
```

17.11.67 Compiler errors C0117

Message: The label <jump label> is not defined within the JMP statement range
Possible error cause: A jump is made to a label that does not exist
Error correction: Define the jump label specified as target

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
JMP A;
```

```
--> C0117: No such label 'A' within the scope of the 'JMP' statement
```

17.11.68 Compiler errors C0118

Message: The jump label <jump label> has not been referenced
Possible error cause: A jump label is defined that is not referenced
Error correction: Remove the unused jump labels

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR
LABEL:
```

```
--> C0118: The label ' LABEL' has not been referenced
```

17.11.69 Compiler errors C0119

Message: An FB_init method of a function block or a structure requires two inputs
bInitRetains and bInCopyCode of type BOOL
Possible error cause: One or several Boolean inputs are missing:

- bInitRetains
- bInCopyCode

Error correction: Define the missing inputs

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
```

```
FUNCTION BLOCK FB
METHOD FB_init
VAR_INPUT
END_VAR
```

```
--> C0119: An 'FB init'-Method of a functionblock or struct needs two inputs
--> 'bInitRetains' and 'bInCopyCode' of type BOOL
```

Error correction:

```
METHOD FB_init
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode : BOOL;
END_VAR
```

17.11.70 Compiler errors C0120

Message: An FB_Exit method of a function block or struct needs an input bInCopyCode of type BOOL

Possible error cause: The input bInCopyCode of type BOOL is missing

Error correction: Define the input

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
```

```
FUNCTION BLOCK FB
METHOD FB_exit
VAR_INPUT
END_VAR
```

```
--> C0120: An 'FB_Exit'-Method of a functionblock or struct needs an input
--> 'bInCopyCode' of type BOOL.
```

Error correction:

```
METHOD FB_exit
VAR_INPUT
    bInCopyCode : BOOL;
END_VAR
```

17.11.71 Compiler errors C0122

Message: Expression SUPER^ is not allowed in this context

Possible error cause: SUPER^ is used outside of derived function blocks

Error correction: Use SUPER^ in function blocks only

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

SUPER^.METH(TRUE, TRUE);
```

```
--> C0122: Expression 'SUPER' is not allowed in this context
```

17.11.72 Compiler errors C0122

Message: "Initialization" is no valid initialization for an enumeration

Possible error cause: A data type that is not ANY_INT is used for the enum initialization

Error correction: Use only ANY_INT for enum initializations

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT;
END_VAR

TYPE DUT :
(
    enum_member := 1.5
) DWORD;
END_TYPE
```

```
--> C0032: Cannot convert type 'LREAL' to type 'DUT'
--> C0124: 'Initialization' is no valid initialization for an enumeration
```

17.11.73 Compiler errors C0125

Message: The constant <constant value> is assigned to more than one enumeration

Possible error cause: The same value is assigned to two or more enumerations

Error correction: Assign different values to the enumerations

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT;
END_VAR

TYPE DUT :
(
    enum_member := 0,
    enum_member2 := 0
);
END_TYPE
```

```
--> C0125: The constant 0 is assigned to more than one enumeration
```

17.11.74 Compiler errors C0126

Message: Variable of type <data type> requires several indices

Possible error cause: Multiple indices are assigned to a variable with one index

Error correction: Assign only one index

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pi : POINTER TO INT;
END_VAR
pi[0,1] := 0;
```

```
--> C0126: Variable of type 'POINTER TO INT' requires exactly 1 Index
```

Error correction:

Example:
pi[0] := 0;

17.11.75 Compiler errors C0130

Message: <object> '<object name>' homed without brackets ()

Possible error cause: A method is referenced without parentheses

Error correction: Always reference methods by means of parentheses

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR
inst.METH1
```

```
--> C0130: METHOD 'METH1' referenced without parentheses '()'
```

Error correction:

Example:
 inst.METH1();

17.11.76 Compiler errors C0131

Message: <value> is not permitted as operand for ADR
Possible error cause: A constant is transmitted as operand to the operator ADR
Error correction: Only use valid operands for ADR
Example of the error:

```
PROGRAM PLC_PRG
VAR
  i : INT;
  pt : POINTER TO INT;
END_VAR

pt := ADR(1);
```

--> C0131: '1' is not allowed as operand for 'ADR'

Error correction:

Example:
 pt := ADR(i);

17.11.77 Compiler errors C0132

Message: No enclosing loop statement applied to EXIT
Possible error cause: EXIT expected outside of a loop
Error correction: Use EXIT only within loops
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

EXIT
;
```

--> C0132: No enclosing loop of which to EXIT

17.11.78 Compiler errors C0136

Message: Ambiguous use of name <variable name>
Possible error cause: A variable is declared in multiple GVLs
Error correction: Qualify the variable with the desired GVL
Example of the error:

```
PROGRAM PLC_PRG
VAR
  j : INT := g_i;
END_VAR

GVL1:
VAR_GLOBAL
  g_i : INT;
END_VAR

GVL2:
VAR_GLOBAL
```

Reference "Programming"

```

      g_i : INT;
END_VAR

--> C0136: ambiguous use of name 'g_i'

```

Error correction:

Example:

```

j : INT := GVL1.g_i;

```

17.11.79 Compiler errors C0139

Message: Code <code> does not have any effect. Is this intended?

Possible error cause: The written code is syntactically correct but does not have any functionality

Error correction: Write code with functionality

Example of the error:

```

PROGRAM PLC_PRG
VAR
      i : INT;
END_VAR

i;

--> C0139: The code 'i;' has no effect. Is this the intent?

```

17.11.80 Compiler errors C0140

Message: Assigning references is only allowed to variables of Reference types

Possible error cause: Trying to assign a reference value to a variable not defined as a reference type

Error correction: Define the variable as a reference type

Example of the error:

```

PROGRAM PLC_PRG
VAR
      i : INT;
      I_r : INT;
END_VAR

I_r REF= i;

--> C0140: Reference assign is only allowed to variables of Reference type

```

Error correction:

Example:

```

I_r : REFERENCE TO INT;

```

17.11.81 Compiler errors C0141

Message: Reference assignment requires a variable with write access

Possible error cause: A constant is assigned to the reference assignment

Error correction: Assign a writable variable

Example of the error:

```

PROGRAM PLC_PRG
VAR
      i : INT;
      I_r : REFERENCE TO INT;
END_VAR

I_r REF= 314;

```

--> C0141: Reference assign needs variable with write access

Error correction:

Example:
I_r REF= i;

17.11.82 Compiler errors C0142

Message: A local variable with the name <variable name> is already defined in <pou name>
Possible error cause: The same variable name is used twice
Error correction: Use different variable names
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    i : INT;
END_VAR
```

--> C0142: A local variable named 'i' is already defined in 'PLC_PRG'

17.11.83 Compiler errors C0143

Message: The property <property name> cannot be used in this context as it lacks the Get access definition
Possible error cause: The property does not have a Get access
Error correction: Make sure that the property has a Get access definition
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    inst: FB;
END_VAR

i := inst.Prop;

FUNCTION_BLOCK FB
VAR
END_VAR

PROPERTY Prop : INT
Set;
```

--> C0143: The property 'Prop' cannot be used in this context because it lacks the get accessor

17.11.84 Compiler errors C0144

Message: Inheritance is only allowed in function blocks, interfaces and structures
Possible error cause: It is attempted to use inheritance in an object that does not permit inheritance
Error correction: Use EXTENDS in function blocks, interfaces, and structures only
Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : DUT_1;
END_VAR

TYPE DUT:
(
    enum_member := 0
```

```
);
END_TYPE

TYPE DUT_1 EXTENDS DUT:
(
    enum_memberX := 0
);
END_TYPE
```

--> C0144: Inheritance only allowed in Functionblocks, Interfaces and Structures

17.11.85 Compiler errors C0145

Message: Interfaces can only be implemented by function blocks
Possible error cause: Trying to implement an interface outside of a function block
Error correction: Implement interfaces only in function blocks
Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

POU();

FUNCTION POU IMPLEMENTS ITF
VAR
END_VAR
```

--> C0145: Interfaces can only be implemented by Functionblocks

17.11.86 Compiler errors C0149

Message: Variable declarations are not allowed in interfaces
Possible error cause: Trying to define a variable in an interface
Error correction: Do not define variables in interfaces
Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : ITF;
END_VAR

INTERFACE ITF
VAR_INPUT
    i : INT;
END_VAR
```

--> C0149: Variable declarations are not allowed in interfaces

17.11.87 Compiler errors C0161

Message: Array range limit <array bound> is not a constant value
Possible error cause: A variable is specified as an array bound limit
Error correction: Use constants for the array limits
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT := 3;
    arr1 : ARRAY[1..i] OF INT;
END_VAR
```

--> C0161: Border 'i' of array is no constant value

Error correction:

Example:
 arr1 : ARRAY[1..3] OF INT;

17.11.88 Compiler errors C0162

- Message:** Number <number of array values> of array initialization is no constant value
- Possible error cause:** The initialization [value1,number value2(value2)] works only with a constant for number value2
- Error correction:** Use constants only
- Example of the error:*

```
PROGRAM PLC_PRG
VAR
  i : INT := 3;
  arr1 : ARRAY[1..4] OF INT := [1,i(7)];
END_VAR

--> C0162: Number 'i' of array initialisations is no constant value
```

Error correction:

Example:
 arr1 : ARRAY[1..4] OF INT := [1,3(7)];

17.11.89 Compiler errors C0168

- Message:** VAR_CONFIG declaration only allowed in VAR_CONFIG
- Possible error cause:** VAR_CONFIG is used outside of a VAR_CONFIG list
- Error correction:** Use VAR_CONFIG only in VAR_CONFIG lists
- Example of the error:*

```
PROGRAM PLC_PRG
VAR_CONFIG
  i : INT;
END_VAR

--> C0168: 'VAR_CONFIG' declaration only allowed in VAR_CONFIG - list
```

17.11.90 Compiler errors C0169

- Message:** VAR_GLOBAL declaration only allowed in global variable list
- Possible error cause:** VAR_GLOBAL is used outside of global variable lists
- Error correction:** Use VAR_GLOBAL in global variable lists only
- Example of the error:*

```
PROGRAM PLC_PRG
VAR_GLOBAL
  i : INT;
END_VAR

--> C0169: 'VAR_GLOBAL' declaration only allowed in Global variable list
```

17.11.91 Compiler errors C0174

- Message:** VAR_TEMP declaration invalid at this position
- Possible error cause:** VAR_TEMP is used outside of a program or function block

Error correction: Use VAR_TEMP inside of programs and function blocks only

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

FUN();

FUNCTION FUN
VAR_TEMP
END_VAR

--> C0174: 'VAR_TEMP' declaration not allowed in this place
```

17.11.92 Compiler errors C0175

Message: RETAIN or PERSISTENT are invalid at this position

Possible error cause: RETAIN or PERSISTENT is used in a function

Error correction: Use RETAIN or PERSISTENT at the intended positions

Example of the error:

```
PROGRAM PLC_PRG
VAR
END_VAR

POU_1();

FUNCTION POU_1
VAR RETAIN
END_VAR

--> C0175: 'RETAIN' or 'PERSISTENT' not allowed in this place
```

17.11.93 Compiler errors C0177

Message: <object> is of type type and cannot be instantiated

Possible error cause: Trying to instantiate a function

Error correction: Instantiate only objects that can be instantiated

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : POU;
END_VAR

FUNCTION POU
VAR
END_VAR

--> C0177: 'POU' is of type 'FUNCTION' and cannot be instantiated
```

17.11.94 Compiler errors C0178

Message: External access to the VAR_IN_OUT parameter <parameter name> of the <object name>

Possible error cause: It is tried to remotely access a VAR_IN_OUT parameter

Error correction: Do not externally access the VAR_IN_OUT parameters

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
    i : INT;
```



```

END_VAR
i := inst.in_out;

FUNCTION_BLOCK FB
VAR_IN_OUT
  in_out : INT;
END_VAR

```

--> C0178: No external access to 'VAR_IN_OUT' parameter 'in_out' of 'FB'

17.11.95 Compiler errors C0182

Message: Return type is only possible for functions and methods

Possible error cause: It is tried to define a return value in a program

Error correction: Define a return value only in methods and functions

Example of the error:

```

PROGRAM PLC_PRG : BOOL
VAR
END_VAR

```

--> C0182: Return type is only possible for POU's of Type FUNCTION and METHOD

17.11.96 Compiler errors C0185

Message: It is not possible to execute a component access <.>, an index access <[]> or a call <()> to the results of a function call.

Assign the result to a variable.

Possible error cause: Component or index access to the result of a function call

Error correction: Assign the result to a variable for access purposes

Example of the error:

```

PROGRAM PLC_PRG
VAR
  it : ITF;
END_VAR

POU_1() [0].METH1();

FUNCTION POU_1 : ARRAY[0..0] OF ITF

INTERFACE ITF

METHOD METH1

```

--> C0185: It is not possible to perform component access '.', index access '[]' or call '()' on result of function call.

--> Assign result to help variable first.

17.11.97 Compiler errors C0189

Message: <;> expected instead of <token>

Possible error cause: Syntax error

Error correction: Make sure that the syntax is correct

Example of the error:

```

PROGRAM PLC_PRG
VAR
  INT
END_VAR

```

--> C0009: Unexpected Token '<Token>' found

--> C0189: ';' expected instead of 'INT'

17.11.98 Compiler errors C0190

Message: <;> expected instead of end of POU
Possible error cause: Syntax error in POU
Error correction: Make sure that the syntax is correct
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
i := 5
```

--> C0190: ';' expected instead of end of POU

17.11.99 Compiler errors C0190

Message: The INDEXOF operator is not supported anymore. Instead, use ADR.
 If ADR is used on a POU name, a pointer to the POU code is returned.
Possible error cause: The outdated operator INDEXOF is used
Error correction: Use the operator ADR

17.11.100 Compiler errors C0195

Message: Implicit conversion of the signed data type <data type 1> to the unsigned data type <data type 2>: Possible loss of sign
Possible error cause: A sign conflict is not taken into consideration in the implicit conversion
Error correction: Convert only data types with the same sign implicitly
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    b : UINT;
END_VAR
```

```
b := i;
```

--> C0195: Implicit conversion from signed Type 'INT' to unsigned Type 'UINT'
 --> : possible change of sign

17.11.101 Compiler errors C0196

Message: Implicit conversion of the unsigned data type <data type 1> to the signed data type <data type 2>: Possible loss of sign
Possible error cause: A sign conflict may not have been taken into consideration in the implicit conversion
Error correction: Use explicit conversions
Example of the error:

```
PROGRAM PLC_PRG
VAR
    i : INT;
    b : UINT;
END_VAR
```

```
i := b;
```

--> C0196: Implicit conversion from unsigned Type 'UINT' to signed type 'INT'
 --> : possible change of sign

17.11.102 Compiler errors C0197

- Message:** Implicit conversion of <data type 1> to <data type : 2> : Possible loss of data
- Possible error cause:** Trying to implicitly convert a variable of data type DINT or LINT to data type REAL
- Error correction:** For DINT, use the data type LREAL, and when converting from LINT to LREAL make sure that the value of LINT does not exceed the capacity of LREAL
- Example of the error:*

```
PROGRAM PLC_PRG
VAR
    i : DINT;
    b : REAL;
END VAR
b := i;
```

```
--> C0197: Implicit conversion from 'DINT' to 'REAL': possible loss of information
```

17.11.103 Compiler errors C0198

- Message:** String constant <string value> too long for destination type <data type>
- Possible error cause:** The string constant has too many characters
- Error correction:** Use shorter string constants or declare larger strings
- Example of the error:*

```
PROGRAM PLC_PRG
VAR
    str : STRING(4) := '12345';
END VAR
```

```
--> C0198: String constant '12345' too long for destination type 'STRING(4)'
```

17.11.104 Compiler errors C0199

- Message:** Interface <interface name> must be instantiated to be accessed
- Possible error cause:** It is tried to access an interface method without the interface being instantiated
- Error correction:** Instantiate the interface
- Example of the error:*

```
PROGRAM PLC_PRG
VAR
END VAR

ITF.METH();

INTERFACE ITF

METHOD METH
VAR_INPUT
END VAR
```

```
--> C0199: Interface 'ITF' must be instantiated to be accessed
```

Error correction:

```
Example:
itest: ITF;
```

17.11.105 Compiler errors C0201

Message: Type <data type 1> is not equal to type <data type 2> of the VAR_IN_OUT variable

Possible error cause: The data type transmitted as VAR_IN_OUT parameter to the function does not match the data type defined in the function

Error correction: Transmit a variable with the correct data type

Example of the error:

```
PROGRAM PLC_PRG
VAR
    Inst: POU;
    b : BOOL;
END_VAR

inst(in_out := b);

FUNCTION_BLOCK POU
VAR_IN_OUT
    in_out : INT;
END_VAR

--> C0201: Type 'BOOL' is not equal to type 'INT' of VAR_IN_OUT 'Variable'
```

17.11.106 Compiler errors C0203

Message: Only structures and function blocks can contain variables of type BIT.

Possible error cause: It is tried to declare a variable of type BIT outside of structures and function blocks

Error correction: Declare variables of type BIT only in structures and function blocks

Example of the error:

```
PROGRAM PLC_PRG
VAR
    b : BIT;
END_VAR

--> C0203: Only Structures and Function Blocks can contain variables of type BIT.
```

17.11.107 Compiler errors C0204

Message: Variables of type BIT have to be declared within a VAR_INPUT-, VAR_OUTPUT or VAR block

Possible error cause: It is attempted to define a variable of type BIT or as VAR_IN_OUT parameter.

Error correction: Define variables of type BIT only within a VAR_INPUT-, VAR_OUTPUT or VAR block.

Example of the error:

```
PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

FUNCTION_BLOCK FB
VAR_IN_OUT
    b : BIT;
END_VAR

--> C0204: Variables of type BIT must be declared within a VAR_INPUT-,
--> VAR_OUTPUT or VAR-block
```

17.11.108 Compiler errors C0205

Message: POINTER TO BIT is not allowed
Possible error cause: An attempt is made to declare a POINTER TO BIT
Error correction: Do not declare POINTER TO BIT

Example of the error:

```
PROGRAM PLC_PRG
VAR
    pt : POINTER TO BIT;
END_VAR

--> C0205: POINTER TO BIT is not allowed
```

17.11.109 Compiler errors C0206

Message: BIT is not permitted as basic type of an array
Possible error cause: It is tried to declare a BIT array
Error correction: Do not declare BIT arrays

Example of the error:

```
PROGRAM PLC_PRG
VAR
    arr : ARRAY[1..2] OF BIT;
END_VAR

--> C0206: BIT is not allowed as base type of an array
```

17.11.110 Compiler errors C0208

Message: MOD is not defined for REAL
Possible error cause: It is tried to perform a modulo operation with a variable of type REAL
Error correction: Modulo operations are only possible with variables of type ANY_INT

Example of the error:

```
PROGRAM PLC_PRG
VAR
    r1 : REAL;
END_VAR
r1 := r1 MOD 2;

--> C0208: 'MOD' is not defined for 'REAL'
```

17.11.111 Compiler errors C0211

Message: Variable declaration expected instead of <EXPRESSION>
Possible error cause: Syntax error
Error correction: Ensure that the syntax is correct

Example of the error:

```
PROGRAM PLC_PRG
VAR
    VAR

    END_VAR
END_VAR

--> C0211: Variable declaration expected instead of VAR END_VAR
```

17.11.112 Compiler errors C0212

Message: VAR, VAR_INPUT, VAR_OUTPUT or VAR_INOUT expected instead of <expression>

Possible error cause: Syntax error

Error correction: Ensure that the syntax is correct

Example of the error:

```
PROGRAM PLC_PRG
i : INT;
```

```
--> C0212: VAR, VAR_INPUT, VAR_OUTPUT or VAR_INOUT expected instead of i : INT;
```

17.11.113 Compiler errors C0216

Message: Duplicate CASE label

Possible error cause: A CASE label is used multiple times

Error correction: Use each CASE label only once

Example of the error:

```
PROGRAM PLC_PRG
VAR
  i : INT;
END_VAR
```

```
CASE i OF
  1: i := i+1;
  1: i := i+2;
ELSE
  i := i+10;
END_CASE;
```

```
--> C0216: Case label duplicate
```

17.11.114 Compiler errors C0217

Message: The CASE label <case label> is also contained in the <case range begin> ... <case range end> range

Possible error cause: A CASE label is part of the range of another CASE label

Error correction: Make sure that there are no intersections

Example of the error:

```
PROGRAM PLC_PRG
VAR
  i : INT;
END_VAR
CASE i OF
  3..5: i := i+2;
  4: i := i+2;
ELSE
  i := i+10;
END_CASE;
```

```
--> C0217: Case label 4 also contained in range 3 .. 5
```

17.11.115 Compiler errors C0218

Message: CASE label requires literal or symbolic integer constant

Possible error cause: It is tried to use a variable as a CASE label

Error correction: Use only literals and symbolic integer constants

Example of the error:

```

PROGRAM PLC_PRG
VAR
    i : INT;
    a : INT := 2;
END_VAR

CASE i OF
    1: i := i+1;
    a: i := i+2;
ELSE
    i := i+10;
END_CASE;

--> C0218: Case label requires literal or symbolic integer constant

```

17.11.116 Compiler errors C0219

Message: CASE contains overlapping ranges <case range 1 begin> .. <case range 1 end> and <case range 2 begin> .. <case range 2 end>

Possible error cause: Two branches of CASE markers have the same elements or subsets

Error correction: Make sure that there are no intersections

Example of the error:

```

PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
CASE i OF
    3..5: i := i+2;
    1..4: i := i+2;
ELSE
    i := i+10;
END_CASE;

--> C0219: Case contains overlapping range 1 .. 4 and 3 .. 5

```

17.11.117 Compiler errors C0221

Message: Direct address <address> is not represented correctly

Possible error cause: An address is not displayed completely

Error correction: Ensure that the address is displayed correctly

Example of the error:

```

PROGRAM PLC_PRG
VAR
    xVar : BOOL;
END_VAR;
xVar := %IX0;

--> C0221: Direct Address '%IX0' malformed

```

Error correction:

Example:
xVar := %IX0.2;

17.11.118 Compiler errors C0222

Message: Outputs cannot be of type REFERENCE TO

Possible error cause: Trying to define REFERENCE TO as an output parameter

Error correction: Do not use REFERENCE TO as an output parameter

Example of the error:

```

PROGRAM PLC_PRG
VAR
    inst : FB;
END_VAR

FUNCTION_BLOCK FB
VAR_OUTPUT
    re : REFERENCE TO INT;
END_VAR

--> C0222: Outputs can't be of type 'REFERENCE TO'

```

17.11.119 Compiler errors C0224

Message: Call recursion: <recursion>
Possible error cause: A function calls itself
Error correction: Ensure that functions are not recursive

Example of the error:

```

PROGRAM PLC_PRG
VAR
END_VAR

POU();

FUNCTION_POU
VAR
END_VAR

POU();

--> C0224: Call Recursion: POU -> POU

```

17.11.120 Compiler errors C0227

Message: Initialization of constant variable <constant name> not constant
Possible error cause: A constant is initialized with a variable
Error correction: Initialize constants only with constant values

Example of the error:

```

PROGRAM PLC_PRG
VAR
    i : INT;
END_VAR
VAR_CONSTANT
    k : INT := i;
END_VAR

--> C0227: Initialisation of constant variable 'k' not constant

```

17.11.121 Compiler errors C0228

Message: No initial value for constant variable <constant name>
Possible error cause: A constant is not initialized
Error correction: Initialize the constants

Example of the error:

```

PROGRAM PLC_PRG
VAR
END_VAR
VAR_CONSTANT
    k : INT;
END_VAR

```

--> C0228: No initial value for constant variable 'k'

Error correction:

Example:
k : INT := 1;

17.11.122 Compiler errors C0234

Message: First operand of __QueryInterface has be an interface reference or the instance of a function block

Possible error cause: Incorrect operands are transmitted to the __QUERYINTERFACE operator

Error correction: Transmit an interface reference or the instance of a function block

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref, ITFref2 : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(a , ITFref);

INTERFACE ITF EXTENDS __SYSTEM.IQueryInterface
INTERFACE ITF2 EXTENDS ITF

--> C0234: First Operand of __QueryInterface must be an interface,
--> reference or the instance of a function block
```

Error correction:

Example:
__QueryInterface(ITFref2, ITFref);

17.11.123 Compiler errors C0235

Message: The second operand of __QueryInterface has to be an interface reference

Possible error cause: Incorrect operands are transmitted to the __QUERYINTERFACE operator

Error correction: Transmit an interface reference

Example of the error:

```
PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref, ITFref2 : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(ITFref2, a);

INTERFACE ITF EXTENDS __SYSTEM.IQueryInterface
INTERFACE ITF2 EXTENDS ITF

--> C0235: Second Operand of __QueryInterface must be an interface reference
```

Error correction:

Example:
__QueryInterface(ITFref2, ITFref);

17.11.124 Compiler errors C0236

- Message:** Incorrect type definition for VAR_EXTERNAL <constant name>
Possible error cause: The variable is declared in VAR_GLOBAL / VAR_EXTERNAL as different types
Error correction: Use the same type definition in VAR_GLOBAL and VAR_EXTERNAL

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    ig : STRING;
END_VAR
```

```
VAR_GLOBAL
    ig : INT;
END_VAR
```

```
--> C0236: Wrong type definition for VAR_EXTERNAL ig
```

17.11.125 Compiler errors C0237

- Message:** No global definition found for VAR_EXTERNAL <variable name>
Possible error cause: It is tried to declare a variable in VAR_EXTERNAL which does not exist in VAR_GLOBAL
Error correction: Ensure that the identifiers match

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    i : INT;
END_VAR
```

```
VAR_GLOBAL
    ig : INT;
END_VAR
```

```
--> C0237: No global definition found for VAR_EXTERNAL 'i'
```

17.11.126 Compiler errors C0238

- Message:** No initial value allowed for VAR_EXTERNAL <variable name>
Possible error cause: Trying to initialize a variable in VAR_EXTERNAL
Error correction: Do not initialize variables in VAR_EXTERNAL

Example of the error:

```
PROGRAM PLC_PRG
VAR_EXTERNAL
    ig : INT := 2;
END_VAR
```

```
VAR_GLOBAL
    ig : INT;
END_VAR
```

```
--> C0238: No initial value allowed for VAR_EXTERNAL ig
```

17.11.127 Compiler errors C0239

- Message:** Interface <interface name 1> does not extend <interface name 2>
Possible error cause: The used interface does not extend another interface
Error correction: Extend the interface

Example of the error:

```

PROGRAM PLC_PRG
VAR
    ITFref : ITF;
    ITFref2 : ITF2;
END_VAR

__QueryInterface(ITFref2, ITFref);

INTERFACE ITF
INTERFACE ITF2 EXTENDS ITF

--> C0239: Interface ITF__Union does not extend __System.IQueryInterface

```

Error correction:

```

Example:
INTERFACE ITF EXTENDS __System.IQueryInterface

```

17.11.128 Compiler errors C0240

Message: First operand of __QueryPointer has to be an interface reference or the instance of a function block

Possible error cause: Incorrect operands are transmitted to the __QUERYPOINTER operator

Error correction: Transmit an interface reference or the instance of a function block

Example of the error:

```

PROGRAM PLC_PRG
VAR
    a : INT;
    ITFref : ITF;
    pt : POINTER TO FB;
END_VAR

__QueryPointer(a, pt);

--> C0240: First Operand of __QueryPointer must be an interface reference,
--> or the instance of a function block

```

Error correction:

```

Example:
__QueryPointer (ITFref, pt);

```

17.11.129 Compiler errors C0241

Message: Second operand of __QUERYPOINTER has to be pointer

Possible error cause: Incorrect operands are transmitted to the __QUERYPOINTER operator

Error correction: Transmit a pointer

Example of the error:

```

PROGRAM PLC_PRG
VAR
    b : INT;
    ITFref : ITF;
    pt : POINTER TO FB;
END_VAR

__QueryPointer(ITFref, b);

INTERFACE ITF EXTENDS __System.IQueryInterface

--> C0241: Second Operand of __QueryPointer must be pointer

```

Error correction:

Example:
 __QueryPointer (ITFref, pt);

17.11.130 Compiler errors C0242

Message: Operand of __DELETE has to be a POINTER

Possible error cause: An incorrect operand is transmitted to the __DELETE operator

Error correction: Transmit a pointer

Example of the error:

```
PROGRAM PLC_PRG
VAR
  a : INT;
  pt : POINTER TO INT;
END_VAR

__DELETE (a);

--> C0242: Operand of __DELETE must be pointer
```

Error correction:

Example:
 __DELETE (pt);

17.11.131 Compiler errors C0243

Message: The name used in the interface is not identical with the object name

Possible error cause: The object name differs from the name used in the code

Error correction: Make sure that the names are identical

Example of the error:

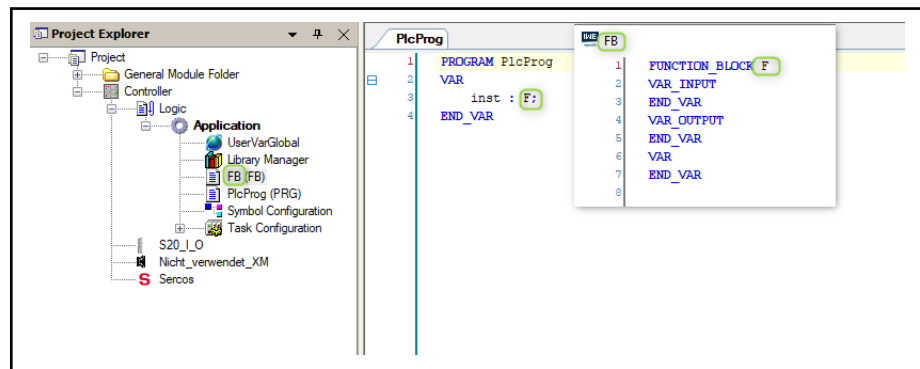


Fig. 17-92:

17.11.132 Compiler errors C0380

Message: The Operators LOWER_BOUND and UPPER_BOUND are only supported for arrays of variable length

Possible error cause: One of the two operators LOWER_BOUND and UPPER_BOUND is not used for an array of variable length

Error correction: Use the operators LOWER_BOUND and UPPER_BOUND only for an array of variable length.

Example of the error:

```
FUNCTION_BLOCK POU
VAR_IN_OUT
  arrIn : ARRAY [*] OF INT;
```

```

END_VAR
VAR
    arrtest : ARRAY [0..5] OF INT;
    test1: DINT;
    test2: DINT;
END_VAR

test1 := UPPER_BOUND(arrin, 0);
test2 := UPPER_BOUND(arrtest, 0);

--> C0380: The operators LOWER_BOUND and UPPER_BOUND
--> are supported only for arrays with variable length.

```

17.11.133 Compiler errors C0509

Message:	Multiple assignments for operator <code>__New</code> not allowed
Possible error cause:	In one line of code, the assignment operator <code>:=</code> is called a multiple number of times with the <code>__New</code> operator
Error correction:	Program the memory allocation with the <code>__New</code> operator in a separate line of code for each pointer that points to dynamically allocated memory
	<i>Example of the error:</i>

```

PROGRAM PLC_PRG
VAR
    pbAlpha : POINTER TO BYTE; // Typed pointer to Alpha
    pbBeta: POINTER TO BYTE;    // Typed pointer to Beta
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pbBeta := pbAlpha := __NEW(BYTE); // Incorrect code for memory allocation
END_IF

pbBeta := pbAlpha := 16#01;

IF (xDelete) THEN
    DELETE(pbAlpha); // Frees memory of pointer
END_IF

--> C0509: Multiple assignments for operator ' __New ' not allowed

```

Error correction:

```

PROGRAM PLC_PRG
VAR
    pbAlpha : POINTER TO BYTE; // Pointer to Alpha
    pbBeta: POINTER TO BYTE;    // Pointer to Beta
    xInit : BOOL := TRUE;
    xDelete : BOOL;
END_VAR

IF (xInit) THEN
    pbAlpha := __NEW(BYTE); // Allocates memory for Alpha
    pbBeta := __NEW(BYTE); // Allocates memory for Beta
END_IF

pbBeta := pbAlpha := 16#01; // Multiple assignment

IF (xDelete) THEN
    DELETE(pbAlpha); // Frees memory of pointer
END_IF

```

17.11.134 Compiler errors C0511

Message:	The function block <function block name> is ABSTRACT and cannot be used as a target for an assignment
Possible error cause:	A value was assigned to an abstract function block. The concrete function blocks may have different types and therefore cannot be copied.

Error correction: In order to copy the data of the function block, concrete function blocks have to be used

Example of the error:

```
PROGRAM PLC_PRG
VAR
refAbstract1 : REFERENCE TO AbstractPOU;
refAbstract2 : REFERENCE TO AbstractPOU;
END_VAR

refAbstract1 := refAbstract2;
--> C0511: The function block 'refAbstract1' is ABSTRACT and
--> cannot be used as a target for an assignment.
```

Error correction: Use the reference assignment REF= to assign the reference refAbstract1 to the same function block as refAbstract2

18 Reference user interface

18.1 Objects

Objects in IndraLogic 2G provide specific functionalities to create your application.

Examples:

Application, program, function, library manager, device, image pool. Objects are managed in the Project Explorer.

Click on **Add** to insert a project into the Project Explorer. The insertion options depend on the position in the Project Explorer.

Properties of each object can be seen and edited via the context menu of the object.

Also refer to

- [chapter 2.2 "Adding objects" on page 8](#)
- [chapter ""Properties" command" on page 596](#)

18.1.1 Object 'General module folder'

Symbol



Call

In the Project Explorer below the project node.

Function

The "general module folder" manages the project settings, see [chapter 18.1.19 ""Project settings" object" on page 545](#).

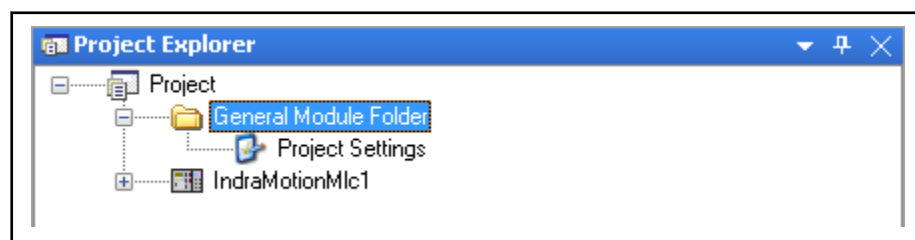


Fig. 18-1: Select the "General module folder" in the Project Explorer

Additionally, project-global objects and instances can be added to the folder via the context menu. The use of project-global instances is used for object-oriented programming, see [chapter 9.18 "Object-oriented programming" on page 145](#)

Project-global objects and instances

Library info...	: For information about properties, meta information and project information, see chapter 18.1.20 ""Library info" object" on page 545 Note: The Library info object cannot be removed after it has been inserted in the project! Only one library info possible in the project.
Library manager...	: Overview of all libraries integrated in the project to create the application, see chapter 18.1.14 "Object 'Library manager'" on page 516 Note: Only one library manager possible in the project!

Image pool...	 : Table in which images are assigned to their IDs, see chapter 18.1.13 "Image pool" object on page 515
Data types...	 : For project-global data types, see "Add DUT" dialog on page 489.
Global Variable List...	 : For declaration, editing and display of global variables, see chapter 18.1.11 "GVL" object - Global Variable List on page 512
Folder...	 : More subfolders can be added to the general module folder, see chapter "Add folder" command on page 605
POU...	 : For project-global program organization units (POU), see chapter 18.1.17 "Object POU" on page 520
Parameter list...	 : For the project-global parameter list, see chapter 18.1.11 "GVL" object - Global Variable List on page 512
Interface...	 : Describes a set of method and property prototypes within the object-oriented programming, see chapter "Interface" object (INTERFACE) on page 529
Text list...	 : For a list to create, manage and compile texts, see chapter 18.1.23 "Text list" object on page 553
Visualization...	 : For IndraLogic visualization, see chapter 20.1 "Visualization overview" on page 757
POU templates	For PLC monitoring functions, see chapter "POU templates" on page 543

Tab. 18-1: General module folder - Supported objects and instances

18.1.2 “Application” object

Symbol: 

It includes objects required for an executable control program. It includes objects required for an executable control program.

The “Application” object is automatically created when creating a control and is thus below the **Logic** node (2).

More objects are automatically created below the “Application” object:

- **Task configuration** (3): Configure which program of the application is called by which task and with which settings.
- **Library manager** (4): Depending on the control hardware, a matching set of libraries is automatically created. More libraries can be added.

Depending on the selected control hardware, further objects are created and provided.

Further objects of your control program such as POU and Global Variable Lists can be added below the application. These objects are only available for this application.

Additionally, the application can also use instances of project-global objects. Project-global objects are managed in the **General module** folder (5). The use of such instances corresponds to the object-oriented programming.

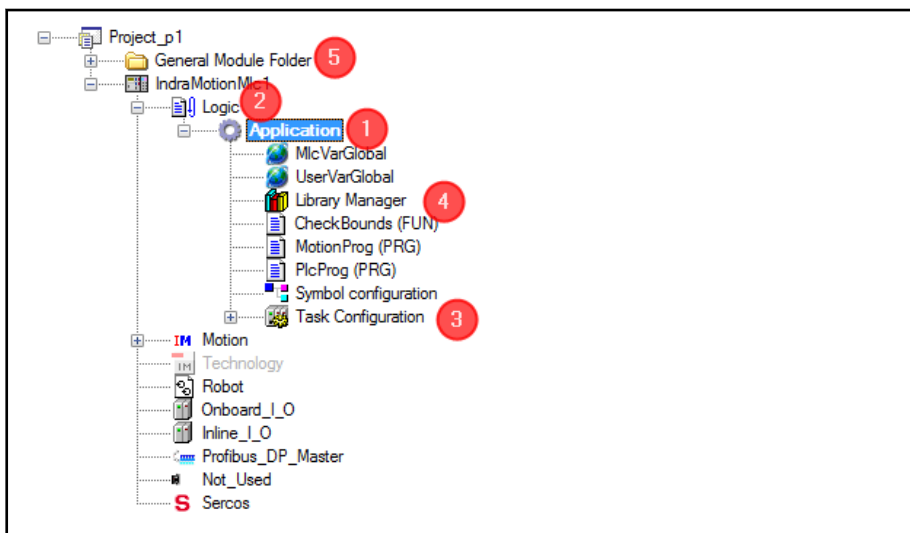


Fig. 18-2:

The application to work with in online mode, has to be set as "active application".

An application can be provided with specific properties in the **Properties** dialog of the application object, in the tab **Application build options**.

Open this dialog via the context menu of the application object.

Example: Enabling dynamic memory allocation.

When downloading the application to the PLC, it is possible to provide information on the application content.

This is also a setting in the **Application build options** tab. The application on the control can subsequently be compared to the active application in IndraLogic.

To provide additional information on the application such as author, version and an individual brief description, modify the information from the **Library info** in the **Properties** dialog under the **Information** tab.

When logging into the target device (PLC or simulation), it is first checked which application is currently on the PLC and whether the application parameters on the control match those in the project configuration. Respective messages show possible mismatches and options for further actions.

Analogously, the application can also be deleted on the PLC.

Go to the **Application** tab of the device editor for the applications currently located on the device. The application can also be deleted there from the target system.

There can also be additional applications in the list without a separate object in the Project Explorer, such as <Application>_symbols.app containing a symbol list created for the application (refer to "Symbol configuration").

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 18.1.25 ""Task configuration" object" on page 562](#)
- [chapter ""Properties" - "Application build options" dialog" on page 707](#)
- [chapter 11 "Transferring applications to the PLC" on page 175](#)

- [chapter "'PLC settings' tab" on page 497](#)
- [chapter 10.2 "Symbol configuration" on page 156](#)
- [chapter 14.1 "Executing an online change" on page 213](#)
- [chapter "'Library info" command" on page 676](#)

18.1.3 "Data source manager" object

Symbol: 

This object is used as a node for managing the data sources below it. At least one data source must be available. An application with the data source manager communicates with remote devices.

Refer to [chapter 10.4.1 "Basics of the data source manager" on page 162](#)

Command "Add object" > "Data source"

Function: This command opens the "Add data source" dialog.

Call:

- Select the data source manager and open the context menu
- Select **Add ► Data source**

Dialog 'Add data source'

Name	Name of the data source to be added
Select data source type	Data source type that fits the control configurations to establish communication. IndraLogic supports the driver "CODESYS ApplicationV3" The communication takes place by means of address monitoring. No other driver can be selected.
Add	Opens the dialog "Initialize data source - Provider settings". The contents of the dialog depend on the selected data source type.



The remote control should be running while initializing the new data source. The remote control application should be loaded and started.

Refer to [chapter 18.1.4 "'Data source" object - "Variables" tab" on page 483](#)

Dialog "Initialize data source wizard - Provider settings" (for data source driver "CODESYS ApplicationV3")



The dialog settings are described in chapter "Object "Data source" - "Communication" tab"

The dialog is used to initially configure the connection for the data source drive "CODESYS ApplicationV3". Communication takes place via address monitoring.

After the configuration settings, press "Finish" to close the dialog. The new data source is created and initialized.

The communication settings can be edited subsequently via the editor of the new data source.

Refer to [chapter 18.1.6 "'Communication" tab" on page 486](#)

"Initialize data source wizard - Browse data source variables" dialog



The dialog settings are described in chapter "Object 'Data source'" under the "'Select variables' dialog".

Function: Option to select the variables of the remote control to be transferred. The transferred data is stored in the data interface. The dialog lists the remote data that are accessed by means of the configured connection.

After the selection, click on "Finish" to close the dialog.

The data source is initialized. The data types and variables (data interface) are declared in the "DataSources_Objects" folder.

The editor for the data source object allows to subsequently modify the settings.

Refer to ["Dialog 'Choose variables'" on page 484](#)

18.1.4 "Data source" object - "Variables" tab

Symbol: 

The object (type "Data source") manages the access to the data of a remote device in the tabs "Variables", "Type mappings", "Communication", and "General and diagnosis". The status bar which is always visible notifies the user about the data source type and the most important communication settings.

Refer to

- [chapter 10 "Working with control networks" on page 153](#)
- [chapter 18.1.5 "'Data source' object - 'Type mappings' tab" on page 485](#)
- [chapter 18.1.6 "'Communication' tab" on page 486](#)
- [chapter 18.1.7 "'Data source' object - 'General and Diagnosis' tab" on page 488](#)

"Variables" tab

The variables for data of remote sources are declared in the Global Variable List <name of data source>. The GVL acts as a data interface for remote control and it is located below the application and folder "DataSources_Objects".

A local application can be developed offline without any connection to the data source on the basis of symbol information.

Refresh variables	Establishes a connection to the remote device and opens the dialog "Choose variables".
Local variable	Access permission of the variables. The respective remote variable has the same access permission. • : Write access. Every time the values changes, the variable is updated on the control. • : Read access. Every time the values changes on the control, the variable is updated in the application. • : Read and write access

Update always	: The control data is updated automatically (by means of the data source). A variable is updated automatically if it is used in the visualization, trend, recipe, or as an alarm. Note: That is the recommended setting type. : The variable is updated in each cycle. Note: Only activate the option if the variable is exclusively used in IEC code. If a variable is used in the visualization code, it is updated automatically. Note: When an instance of a function block or a data type is updated in this way, the instance is always transferred completely.
Create or map	Mapping type how the remote variable is mapped to the local variable. : Mapping to a specific created variable with the data type of the remote variable. The control data is mapped 1:1. This is the recommended mapping type. The variable is declared in the GVL<name of data source>. : Mapping to an existing variable. This requires that the existing variable has the same data type. : Mapping to a created variable with type-conforming data type to the remote data type: remote and local data types are not the same, but compatible. A type-conforming data type can also be available in a library. The variable is declared in the GVL <name of data source>.
Type mapping	Data type of the remote variable. If the variable does not have a scalar type, the type is listed in the "Type mappings" tab.
Remote variable	Variable in the remote control

Dialog 'Choose variables'

Symbol: 

Function: This dialog lists the remote variables that are accessed by means of the configured connection.

Call: Click "Update variables" in the "Variables" tab.

Prerequisite: The remote control is running. The control application is downloaded.

Variables	The remote variables are listed in the tree view. The top node is identified by the remote application name. Its variables are listed below that. Structured data is listed with all of its subordinate elements. Example: appControl_A : The variable is selected for transfer to the local device. If the variable is structured, it is applied with all sub-elements. If the variables themselves are subelements, then only this subelement is accepted without accepting the structure completely. Red font: When a variable is displayed in a red font, the variable is not available (anymore) in the remote control. : The variable is not selected for the transfer.
	The variable has expandable elements. By clicking the symbol, the variable is extended by their elements.
Insert elements structured	: The selected variables are transferred with this structure if they are structured. : The variable is not selected for the transfer.

18.1.5 “Data source” object - “Type mappings” tab

The tab lists the non-scalar data types as they are currently available in the "DataSources_Objects" folder. Edit the data type declaration by selecting the data type and by changing or removing the declared elements in the lower window. Moreover, you can modify the name, reset access rights, map another type, or select another remote variable.

Refer to [chapter 10.4.3 "Editing data source variables" on page 167](#)

“Type mappings” tab

Local type	Data type in the local application.
Create or map	: Mapping to a newly created data type. Declared in the "DataSources_Objects" folder. : Mapping to an existing data type : Mapping to a type-conforming data type. Declared in the "DataSources_Objects" folder.
Mapping name	Name of the data type
Remote type	Data type of the remote control
List with the subordinate elements of the selected data type	
local variable	Local variable name of the element of the selected data type
access rights	Access rights to the element
Type mapping	Data type of the element
Remote variable	Remote variable name of the element of the selected data type

18.1.6 “Communication” tab

This tab includes the communication settings for the remote data source. When adding a data source, a data source type was selected, and depending on the type, the communication settings to the data source were configured. Then communication settings are managed and edited in this tab. The contents of the tab vary according to the data source type and connection.

Refer to

- [chapter 10.4.2 "Adding initial data source" on page 164](#)
- [chapter 18.1.3 ""Data source manager" object" on page 482](#)

“Communication” tab for data source type CODESYS ApplicationV3

Tab of data source type "CODESYS ApplicationV3" to configure the connection settings (communication via address monitoring).

Select project type	• "Current project" The control application is part of the current project. • "Other project": The control application is stored as a separate project and not part of the current project.
Choose file	Name and path of the project that the application contains for the remote control. Example: C:\Data\Project new\Project\IndraLogic\IndraLogic.project Prerequisite: The "Choose project type" is "Other project".
	Window that lists the detected control variables in a tree view. When the control data is displayed here, a connection can be established.

"Target device"

Use device address	The address of the connected data source device is entered as configured in its communication settings. Example: 0000.0A34.864B.C000.2DDC.0A34.906F The following cases lead to an incorrect address: • The gateway used to communicate with the device runs on a computer with multiple IP addresses. • The device where the application is running with the data source manager has multiple IP addresses or other interfaces (USB, serial, etc.) for communication with IndraLogic. • The IndraLogic runtime system and the gateway are not in the same Ethernet subnet.
From device	By clicking "From device", IndraLogic automatically enters the data of the connected data source device as it is configured in its "Communication settings".

Search for the target device using the network scan	Possible search criteria for the device: • Node name • Target type Example: 4096 • Target ID Example: 1001 0114 • Target version Example: 14.20.20.0 • Network path – "Direct child of the data sources PLC": The scanned remote control has an address that is running with the address of the local control (of the data source manager). – "Direct child of the node with address:" Address of the parent node specified – "Direct child of the data sources PLC or of the node with address:" Combination of both the options mentioned above.
Search type:	• "First device found": The first control in the Project Explorer meeting the criteria, is selected • "Exactly found device": The control is selected that fulfills the specified criteria Note: The data source manager waits until the network scan is complete. Normally, it takes 10 seconds.

"Login configuration"

If a user management is configured on the remote control, valid access data is required at login.	
Type	• "Login with the following access data": The access data is hard-coded in the "User name" and "Password" settings. They are used at each connection attempt. • "Login with access data defined at runtime": A dialog prompts for the name and password at runtime.
User name	Example: Jane.Doe Prerequisite: Setting "Type" is "Login with the following access data".
Password	Prerequisite: Setting "Type" is "Login with the following access data".

"Expert settings"

Expert settings	 : The settings for the "Default communication buffer" are visible.

Refer to [chapter 10 "Working with control networks" on page 153](#)

18.1.7 "Data source" object - "General and Diagnosis" tab

The "General and diagnosis" tab provides information on the status of the data source communication.

Refer to [chapter 10.4.5 "Updating data interface" on page 170](#)

"General and Diagnosis" tab

Refreshing the configuration

Refresh rate (ms)	Example: 200

Connection information

Connection status	Example: online
Error information	Example: OK

18.1.8 Object "DUT"

Symbol:  for DUT without text list support,  for DUT of type "Enumeration" with text list support.

A DUT (Data Unit Type) describes a user-specific data type. A **DUT** object can be added below the application or to the "General module folder".

Certain definitions can already be made when adding, see below: **Add DUT** dialog.

Syntax

```
TYPE <Bezeichner> : <DUT Komponentendeklaration>
END_TYPE
```

The component declaration depends on the selected type, structure or enumeration.

Example:

The following shows two DUTs defining the structures `struct1` and `struct2`:

`struct2` supplements `struct1`. This means

that `struct2.a` can be used to access the `a` variable.

```
TYPE struct1 :
STRUCT
  a:INT;
  b:BOOL;
END_STRUCT
END_TYPE

TYPE struct2 EXTENDS struct1 :
STRUCT
  c:DWORD;
  d:STRING;
END_STRUCT
END_TYPE
```

Also refer to

- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)
- [chapter 17.5.17 "Enumerations" on page 374](#)
- [chapter 17.5.6 "'UNION' data type" on page 354](#)

"Add DUT" dialog

Function: Use this dialog to configure a new DUT.

The DUT (Data Unit Type) describes a user-specific data type.

Call: Menu **Add ► DUT**, context menu of the Application object.

Name	DUT name
Data type	
Structure	The DUT consists of a structure of various data types. Advanced : The structure extends an existing structure. Thus, the definitions of the extended DUT are automatically valid in the new DUT.
Enumeration	The DUT consists of a series of integer constants. For the declaration syntax, refer to the help page on "Enumerations". <i>"Add text list support"</i> : The text list support allows the localization of enumeration components. Additionally, the symbolic and not the numeric enumeration values are shown in the text output of a visualization element. If a text list-supported enumeration variable is entered into the property "<i>Text variable</i>" of a visualization element, it gets the addition <EnumerationName>. Use the button at the right edge of the editor to toggle between  <i>Textual view</i> and  <i>Localization view</i> (Text list). The DUT object is shown below the "Application" node or in the "General module folder" with the  symbol. Example: The variable <code>PLC_PRG.enVar</code> of the type <code>myEnum</code> is used. <code>myEnum</code> is a text list-supported DUT. The entry looks as follows in the "Properties" editor: <code>PLC_PRG.enVar <myEnum></code>. If the enumeration type is changed in the application, an input prompt is displayed asking whether IndraLogic is supposed to update the respective visualizations. Note: The text list support can subsequently be added or removed at any time on an existing enumeration object: Use the commands <i>"Add text list support"</i> or <i>"Remove text list support"</i> in the context menu of the object. : An enumeration without text list support is created. The DUT object is shown below the "Application" node or in the "General module folder" with the  symbol
Alias	Definition of an alias for a basic type. Enter the basic type either directly or select it using the input assistant or the array assistant.
Union	All components reserve the same storage space for this data type. The memory requirement of a union is determined by the memory requirement of its "largest" component.

Also refer to

- [chapter 17.5.16 "Data type 'STRUCT'" on page 373](#)
- [chapter 17.5.17 "Enumerations" on page 374](#)
- [chapter 17.5.6 ""UNION" data type" on page 354](#)
- [chapter ""Add text list support" command" on page 688](#)
- [chapter ""Remove text list support" command" on page 688](#)

18.1.9 “Device” object and generic device editor

Symbol: 

A device object represents a certain hardware.

Examples:

Control device, field bus node, bus coupler, drive, I/O module, monitor.

Map the hardware structure by arranging the device objects in the Project Explorer.

The I/Os of the devices can also be linked to project variables in the configuration editors of the device objects.

Add a device object to the Project Explorer with **Add** (or via drag&drop from the IndraWorks library "Drive and Control").

IndraLogic provides the devices matching the insertion position in the Project Explorer for selection.

Double-click on a device object in the Explorer to open the corresponding device editor. It includes generic as well as specific tabs for the device configuration.

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter "Generic device editor" on page 490](#)

Generic device editor

The generic device editor contains tabs for the configuration of a PLC device in IndraLogic. Additionally, there are device-specific tabs. Thus, the configuration editor consists of many different dialogs (depending on the device).

Double-click on the device object in the Project Explorer (**Devices** view) to open the editor.

To change the setting for a device editor, go to **Tools ► Options ► IndraLogic 2G ► General settings** and ... ► **Device editor**.

For example, the tabs of the generic device editor can be shown and hidden.

A device editor has the name of the device.

The following tabs of the generic device editor can be included:

- **Applications:** List of applications on the control.
- **Files:** Configuration of the file transfers between a "host" file system and the device, context menu: **Device data**.
- **Log:** Display of the PLC log file.
- **PLC settings:** Configuration and handling of I/Os: Which application, response in "Stop" state, update, bus cycle options, etc.
- **PLC shell:** Text-based control monitor to query certain information from the control.

- **Users and groups:** User management with regard to permitted access to the device at runtime.
- **Access rights:** Rights to access objects and files on the device.
- **Parameter:** Display or configuration of the device parameters and floating point settings: Response on exception. General response on floating point exceptions.
- **I/O mapping:** Mapping IEC objects
- **Task list:** Overview on all inputs and outputs assigned to tasks (useful for troubleshooting).
- **Status:** Device-specific status messages and diagnostic messages.
- **Information:** General information on the device (name, vendor, version, etc.)

Also refer to:

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 8 "Configuring an I/O link" on page 69](#)

Tab "Parameters"

The device-specific parameters are displayed in a table on this tab of the generic device editor. The device description defines which parameters can be edit in this dialog.

To sort the entries in alphabetically ascending or descending order or in the default order, click on the column header.

Parameters	Parameter name, not editable
Type	Data type of the parameter, not editable
Value	Initially, the default value of the parameter is displayed directly or the corresponding symbol name. Parameters that cannot edited are highlighted light gray. If the parameter can be edited, double-click on the table field to open an input field, a drop-down list or a file selection dialog to change the value.
Default value	Default value of the parameter defined by the device description, not editable
Unit	Measuring unit of the value, e.g. "ms" for milliseconds (cannot be edited)
Description	Brief description of the parameter specified by the device description, not editable

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter "Generic device editor" on page 490](#)

"Applications" tab

The applications on the device are displayed on this tab of the generic device editor. Depending on the system, applications on the device can be deleted or detailed information on the application can be retrieved.

Applications on the PLC	List of the applications found during the last scan of the control device via Refresh list .
Delete Delete all	Deletes the application selected in the list or all listed applications on the control Note: If a safety control is configured under a control, this command can permanently interrupt the communication settings of the safety control to other safety controls (via safety network variables), to field devices and to the development system! The safe field devices or the other safety controls can go to the safe state as reaction. The connection to the development system is only affected if the safety control is connected to the main control via the field bus. Refer to chapter 10.5 "Subordinate safety control" on page 172
Details	Opens the Details dialog. It displays information defined for the application in the Information tab of the Properties dialog.
Content	Prerequisite: The Download the application info option is selected in the Properties of the application object in the Application generation options tab. Thus, information on the application content is additionally loaded to the PLC. The Content button opens a dialog with additional information on the differences between the latest code generated and the application code on the control. The different function blocks are displayed in a comparison view.
Refresh list	The control is scanned for applications and the list is refreshed accordingly

Also refer to

- [chapter 18.1.9 ""Device" object and generic device editor" on page 490](#)
- [chapter 18.1.2 ""Application" object" on page 480](#)
- [chapter "Generic device editor" on page 490](#)
- [chapter ""Information" tab" on page 511](#)
- [chapter ""Properties" - "Application build options" dialog" on page 707](#)
- [chapter "Menu 'Login'" on page 614](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)

Tab "Backup and restore"



This tab of the generic device editor is only available for the control hardware IndraControl VPx in the system IndraMotion MLC.

This tab is used to back up and restore the application-specific files on the PLC. A zipped backup file is generated or reloaded.

Prerequisites:

- The communication settings are correct for connection to the device
- The application to be backed up is available on the control.

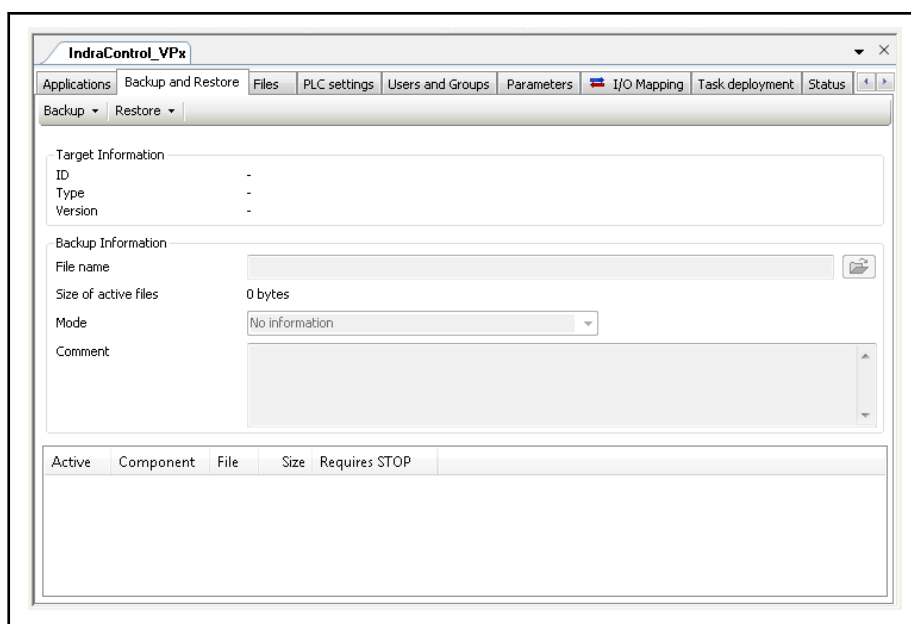


Fig. 18-3: Tab "Backup and restore"

Menu bar

Backup	This button opens a menu with the following commands:  "Read backup information from device": This command searches for application-specific files from the \$PLcLogic\$ directory of the PLC and lists them in the lower part of the tab as table.  "Create backup file and save to disk": Prerequisite: The "Read backup information from device" command was used to determine the backup-related files. These files are located in the lower part of the tab as table. This command compresses the files in the table set as "Active" and the meta.info information file in a backup zip file. The file extension is tbf (= "Target Backup File").  "Save backup file to device": Prerequisite: The backup file has been saved to the disk. This command saves the backup file to the TBF directory of the PLC.
Restore	This button opens a menu with the following commands:  "Load backup file from disk": This command opens the "Open file" dialog to search for a saved backup file in the file system. The included files are listed in the table in the lower part of the tab.  "Load backup file from device": This command generates a list of all backup files found on the PLC. Select one of these files to view its content on the tab as table. For the restore operation, optional components can be disabled and comments can be edited.  "Restore on device": This command is available if at least one component of the backup file just loaded in the tab is set to active. It causes the restoration of the application status on the device. The user interface is blocked during restoration. You can cancel the operation.

Target system information

ID	ID of the PLC, example: 1001 0023
Type	Device type, example: 4096
Version	Device version, example: 14.19.20.0

Backup information

File name	Storage path of the backup file. Use the button to open the file system browser. Example: PlcLogic\$/ Application/Application.crc
Size of active files	Specified in kilobytes: Total of the actively set files in the following table, example: 206 KB (210965 bytes)
Backup mode	Defines the scope of the backup: "Application". The application-related files are zip-ped.
Comment	Optional comment input to be saved in the meta.info file of the backup and that can be read when the files are restored.

Table of files for backup

Active	 : Optional files can be disabled here to exclude them in the backup file. Required components are shown here with a green checkmark (no checkbox).
Component	Affected component, example: File system
File	Name of the component file to back up, example: \$PlcLogic\$/ Application/Application.app
Size	File size in bytes, example: 43280
Requires STOP	 : For the component, the application is to be stopped before backing it up and restoring it. In this case, a warning is issued when the backup or the restoration is started.

"Synchronized files" tab

This tab of the generic device editor lists the files that are downloaded to the PLC when the application is downloaded. For example, these are external files that were added to an application.

Implicit files, such as the source code archive file, are displayed here only if their time of download is configured for this and the "Show implicit files for application download on the editor of a PLC" option is activated in the IndraLogic 2G options (category "Device editor").

File name	Name of the file below the application, or direct name of the implicitly transferred file. Double-click the file name to open the file.
Host path	Storage location or original location of the file Example: D: \Proj1\Files. Double-click the path to open the directory in the file explorer.
Time interval	Time interval of the file update on the PLC Example: After download/online change of the application

Information	Object-dependent additional information Example: Object: external file
Origin	General origin type of the file, Examples: External file object, source code download provider

Also refer to

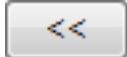
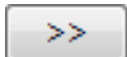
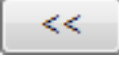
- [chapter "Dialog 'Options' - 'Device editor' " on page 722](#)
- [chapter 18.1.9 ""Device" object and generic device editor" on page 490](#)

"Files" tab

In this tab of the generic device editor, files can be transferred between IndraLogic (host) and the control device.

If the communication settings are correct and the control is running, IndraLogic establishes the connection automatically to the control during the file transfer.

Left dialog section, host

Location	Path in the file system of the host. Set like in the default file manager. Subdirectories and files are shown in the lower part of the window with name, size and change date.
	Opens the dialog for creating a new directory in the set path.
	Deletes the selected files or directories.
	Updates the list of files and directories for the set path (location).
 	Copies the selected files and directories to the other file system. If a file is not already available in the target directory, it is created. If it is already available and not write-protected, it is overwritten. A corresponding message is then output  corresponds to the "Write file to control" command  corresponds to the "Read file from control" command

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter 15 "Copying files to/from PLCs" on page 217](#)

"Log" tab

See the PLC "log" in this tab of the generic device editor. It lists the events recorded on the target system.

This applies to the:

- Events upon the startup and shutdown of the system (components loaded with version)
- Application download and loading of the boot application
- Customized entries
- Log entries from I/O drivers
- Log entries of data sources

Offline recording	☐ : Standard setting ☒ : The PLC also records actions not related to the connection to the control. This is currently only implemented for the IndraLogic safety version.
UTC time	☐ : Standard setting; The time stamp is converted into the local time on the computer (calculated using the time zone of the operating system). ☒ : The time stamp of the runtime system is displayed.
Severity	There are the following categories for the severity of the event: : Information : Warning : Error : Debug : Exception Show or hide each category using the corresponding button in the bar above the list. Each button shows the number of log entries of the respective category.
Time stamp	Date and time, e.g.: 12.01.2007 09:48
Description	Description of the event, e.g.: Import function failed of
Component	Name of the respective runtime system component, e.g. CmpApp
Selection list with component names	The log list displays only events that concern the selected component
Logger	Selection list with all available recordings. The default setting is the <DefaultLogger> specified by the target system, currently identical to "PlcLog" for the IndraLogic runtime system
	Refreshes the log list
	Export of the list content to an xml file. File name and storage directory can be selected.
	Import of a log list from an xml file. The list is displayed in a separate window.
	The displayed log list is emptied, i.e. all entries are deleted.

Tab. 18-2:



Notes on troubleshooting:

For exceptions with the description `*SOURCEPOSITION*`, double-click to open the affected function in the editor or select **Display source code in editor** from the context menu.

The cursor jumps to the line that is causing the error.

This diagnostics can also be executed if the IndraLogic project archive, including the "download information files" and the exported log file are present. If the function concerned is protected, the message `The source code is not available for <FunctionName>` is displayed.



If a `VendorException` is reported, a manufacturer-specific exception occurred in the IndraLogic runtime system.

For more information, please contact the Rexroth service, see [chapter 21 "Service and support" on page 1019](#).

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter 18.1.9 "'Device' object and generic device editor" on page 490](#)
- [chapter 13.5 "Reading PLC logbook" on page 208](#)

'PLC settings' tab

On this tab of the generic device editor, basic settings can be made to configure the PLC, e.g. handling inputs, outputs and the bus cycle task.

Application for I/O handling	Application responsible for the I/O handling.

PLC settings

Refresh I/Os in stop	: IndraLogic refreshes the values of the input channels and output channels even if the PLC is in the "Stop" state. If the watchdog detects a malfunction, the outputs are set to the predefined default values. : IndraLogic does not refresh the values of the input channels and output channels when the PLC is in the "Stop" state. Note: The setting "Update I/O while in stop" refers only to the PLC response. The field bus communication does not depend on this setting, i.e. at a field bus interface, most data is set to "0" or marked as "invalid" (bus-specifically) when the PLC is in stop.
Response of the outputs at stop	<i>Handling the output channels when the control enters the "Stop" state:</i> • Retain values: The current values are retained. Note that the outputs after an IEC exception retain their value from the last cycle. • All outputs to default value: The default values from the I/O mapping are assigned. • Execute program: Handling the output values can be controlled via a program in the project. IndraLogic executes this program at "STOP". Enter the program name in the field on the right.
Always update variables	<i>Global setting, which defines whether IndraLogic updates the I/O variables in the bus cycle task. This setting is only applied to I/O variables of the slaves and modules if "deactivated" is defined in their update settings.</i> • Deactivated (update only if used in a task): IndraLogic updates the I/O variables only if they are used in a task. • Activates 1 (use bus cycle task if not used in another task): IndraLogic updates the I/O variables in the bus cycle task if it is not used in another task. • Activated 2 (always in bus cycle task): IndraLogic updates all variables in each cycle of the bus cycle task, regardless of whether they are used or mapped to an input channel or output channel.

Bus cycle options

Bus cycle task	Task controlling the bus cycle. The task defined by the device description is entered by default. By default, the bus cycle setting of the superordinate bus device (use cycle settings of the superordinate bus) applies, i.e. the Project Explorer is scanned upwards for the next valid bus cycle task definition. Pay strict attention to the following notes!



Before selecting the setting **<Unspecified>** for the bus cycle task, note the following:

<Unspecified> means that the default setting in the device descriptions is applied. Thus, check this description. Use of the task with the shortest cycle time can be defined as the default, but the use of the task with the longest cycle time could also be defined!



A defined cycle grid is required for field buses to ensure a determined response. Therefore, do not use the type "free-running" for a bus cycle task.

Additional settings

Force variables for the I/O mapping	This setting is available only if it is supported by the device. : When compiling the application, IndraLogic creates two global variables for each I/O channel mapped to a variable in the I/O mapping dialog. These variables can be used to force the input value or output value on this channel, e.g. via an HMI visualization.
Enabled diagnostics for devices	: IndraLogic automatically integrates the library <code>CAA Device Diagnosis</code> into the project and creates an implicit function block for each device. If there is already a function block for the device, either an extended function block is used (e.g. with EtherCAT) or another function block instance is added. This contains a general implementation for the device diagnostics. Use function block instances to determine the status of all devices in the application and to analyze errors. In addition, the library contains functions for the programmatic editing of the Project Explorer. Example: Scanning all children of a bus system, jumping to the parent element.
Display I/O warnings as errors	Warnings on the I/O configuration are displayed as errors.

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter "Tab '<Device name> I/O mapping' on page 504](#)
- [chapter "'Parameters' tab" on page 499](#)
- [chapter "'Build' command" on page 611](#)
- PDF document "CAA Device Diagnosis", which is a component of the library.

"Parameters" tab

It is preset, that the floating-point exceptions are blocked in the PLC program. This ensures compatibility for existing user projects.

For the IndraMotion MTX and IndraMotion MLD, handling of floating-point exceptions in the PLC cannot be configured. The dialog in the device editor is omitted

In the PLC program, floating-point exceptions can occur during floating-point calculations.

Example: Division by zero

For this purpose, the PLC provides a default handling. That means that the PLC goes into STOP state and the floating-point exception is displayed.

The user can select

- whether the PLC goes into "Stop" state (default reaction) or
- whether the PLC remains in the "Run" state and continues the calculation for the PLC project when a floating-point exception occurs.

Background

If the handling of floating-point exceptions is blocked and if an exception occurs during a floating-point calculation, the floating-point processor (FPU) automatically uses a value allowing the calculation to continue in most of the cases.

Example:

For floating point calculations, a division by 0 results in the value "Infinity".

This does not always work out and depends on the calculation method in the user program. Continuing the calculation can thus also cause an invalid reaction of the user program.



For new PLC projects, the floating-point exception handling should be at least enabled in the engineering phase.

Types of floating-point exceptions

In the IEEE standard for floating-point calculations, different types of floating-point exceptions are defined. These are supported by floating-point processors according to IEEE standard.

- Overflow
- Underflow
- Divide by zero
- Invalid
- Denormalized
- Inexact (precision)

It is not reasonable to execute an exception handling via software for all floating-point exceptions. If "Inexact" exists permanently and the FPU hardware rounds according to a preset algorithm, no software handling can be selected for the "Inexact" floating-point exceptions.

Dialog description, Floating-point exceptions

To open the "Devices" dialog in the right window, double-click on the control in the PLC project. There is a new "Configuration" tab.

Parameter	Type	Value	Default Value	Unit	Description
Floating-Point Settings					
Behaviour on Exception	Enumeration of BYTE	no reaction	no reaction		
Floating-Point Exceptions					
Overflow	Enumeration of BYTE	enabled	enabled		
Underflow	Enumeration of BYTE	enabled	enabled		
Division by Zero	Enumeration of BYTE	enabled	enabled		
Invalid Operation	Enumeration of BYTE	enabled	enabled		
Denormalized	Enumeration of BYTE	enabled	enabled		

Fig. 18-4: Device dialog, here IndraMotion MLC L65, "Parameters" tab

The floating-point settings are in the "Parameters" tab.

The floating-point settings are divided into two components:

- Behavior on exception
- Floating-point exceptions

In the component "Behavior on exception", there are the following selection options:

- **No reaction:** All exception conditions occurring at floating-point calculations are ignored.

The PLC program is still executed (PLC remains in Run).

The floating-point exceptions set in the dialog are not relevant in this case.

- **PLC in Stop:** If enabled exception conditions occur, the PLC is switched to Stop.

Floating point exceptions set to the "enabled" state in the dialog are considered.

The floating-point exceptions displayed in the dialog are only relevant if "PLC in Stop" is set as response.

In the presetting, all floating-point exceptions are enabled. However, the user can disable any floating-point exceptions:



The floating-point exception "Inexact" is not displayed. It is always disabled.

Default behavior

The reaction of the PLC when a floating-point exception occurs is the same as for the previous exceptions.

- The PLC goes into the "STOP" state and "EXCEPTION".
- The cause of the exception is entered in the logger.
- The program location causing the exception is displayed in the monitor.
- "STOP" and "EXCEPTION" are displayed in the status line of the programming system.

Example:

Floating-point exception "Division by 0", exception **enabled**

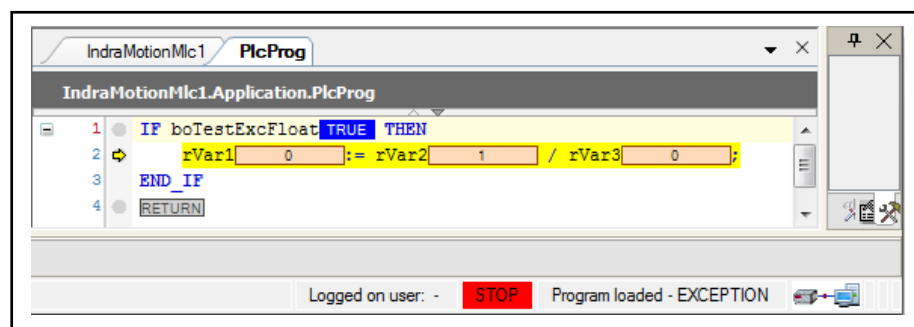


Fig. 18-5: PLC in STOP, EXCEPTION: Display of the cause in the program code

☐ Offline-Logging ☐ UTC Time			
0 Warnung(en) 0 Fehler 6 Ausnahme(n) 199 Information(en) 295 Debug-Meldung(en) <All > Logger: <Defa >			
Severity	Time Stamp	Description	Component
!	17.02.2016 15:52:07	*SOURCEPOSITION* App=[Application] area=0, offset=396750	CmpIecTask
!	17.02.2016 15:52:07	*EXCEPTION* [UNKNOWN] occurred: App=[Application], Task=[PlcTask]	CmpIecTask
!	17.02.2016 15:51:30	Update I/O Mapping after download succeeded.	CmpOpenCtrl
!	17.02.2016 15:51:30	Parse of I/O configuration after download succeeded.	CmpOpenCtrl

Fig. 18-6: Logger message

Example:

Floating-point exception "Division by 0", exception **disabled**

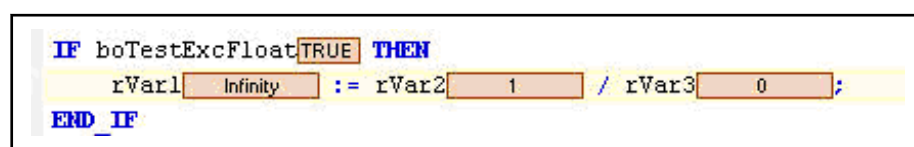


Fig. 18-7: Program code, status display at running program

If no exceptions are enabled (as now), the PLC program continues running. The result of the calculation is "Infinity" in this case.

“PLC shell” tab

This tab of the generic device editor contains a text-based control monitor to query certain information from the control.

Device-dependent commands can be entered and the response of the control is output in a result window.

Prerequisites:

- The option 'PLC shell' is enabled
- The application node is logged in (connection to control/PLC is required)

Activation: IndraWorks menu **Tools ▶ Options ▶ IndraLogic 2G ▶ General settings**.

Command with possible parameters	Description
?	List of available PLC shell commands with possible parameters and brief description.
getcmdlist	List of the names of the available PLC shell commands.
mem <MemoryAddress> [<Size>]	Provides the hexdump of a defined storage area. The size specification is optional and describes the number of bytes to be output. Default value: 16. Example: mem 16x0422139C 8
reflect	Repeats the entered command (to test the connection)
applist	Provides a list of all loaded applications. The sequence in the list defines the application index starting with 0.
pid [<ApplicationName> <ApplicationIndex>] *	Provides the GUID (application index) of one or all loaded applications.
pinf [<ApplicationName> <ApplicationIndex>] *	Provides the contents from the following fields from the library info of the project: Title, version, author, description. Requirement: The option Automatically creating POU's for the properties access was enabled in the "Library info".
startprg [<ApplicationName> <ApplicationIndex>] *	Starts the given application or - if no application is specified - all loaded applications.
stopprg [<ApplicationName> <ApplicationIndex>] *	Stops the given application or - if no application is specified - all loaded applications.
resetprg [<ApplicationName> <ApplicationIndex>] *	Resets the given application or - if no application is specified - all loaded applications.
resetprgcold [<ApplicationName> <ApplicationIndex>] *	Executes a cold start of the given application or - if no application is specified - all loaded applications.
reload [<ApplicationName> <ApplicationIndex>] *	Loads the boot application of the given application or - if no application is specified - the boot projects of all loaded applications.
getprgstat [<ApplicationName> <ApplicationIndex>] *	Provides the program status of the given application or - if no application is specified - all loaded applications.
plcload	Displays the processor load of the control in percent.
rtsinfo	Provides information on the runtime system, such as processor and runtime system version.
channelinfo	Provides information about the communication channel

Command with possible parameters	Description
<code>rtc-get</code>	Provides the universal time (UTC) via the DataTime string
<code>listpcicards [<VendorID>]</code>	Provides a list of all PCI adapters or adapters filtered by <VendorID>
<code>gettaskgroups</code>	Provides a list of all task groups, their tasks, and the CPU core binding
<code>cert-getapplist</code>	Shows all registered and used certificates (ID of the component and use).
<code>cert-genselfsigned [<number for search result by "cert-getapplist"> <expdays=>]</code>	Generates self-signed certificates. The validity period of the certificate can be specified by means of <code>expdays=</code> . The default is 365 days.
<code>cert-gendhparams [length in bits]</code>	Generates the parameters for the Diffie-Hellman key exchange. Caution: This process can take several minutes!
<code>cert-getcertlist [<trust level>]</code>	Lists all certificates of the specified trust level. If a trust level is not given, then all certificates are listed. Possible trust levels: • <code>untrusted</code>: Untrustworthy certificates • <code>trusted</code>: Trustworthy certificates • <code>own</code>: Certificates of the controller • <code>quarantine</code>: Certificates whose trust level (trusted, untrusted) cannot be determined by validation. Therefore, incoming connections were denied.
<code>cert-createcsr [<number for search result by "cert-getapplist">]</code>	Generates CSR files for all applications
<code>cert-import <trust level> <filename.cer></code>	Imports the specified certificate
<code>cert-export <trust level> [<number for search result by "cert-getcertlist">]</code>	Exports the specified certificate.
<code>cert-remove <trust level> <number for search result by "cert-getcertlist" or "all"></code>	Removes the specified certificate
<code>cpuload</code>	Displays the processor load of the CPU (in case of multicore, per processor kernel)
<code>gettaskgroups</code>	Returns a list of the defined task groups. The assigned tasks are displayed for each task group.
<code>getmulticoreinfo</code>	Indicates if multicore is supported and how many processor kernels are available.

Tab. 18-3: PLC shell standard commands

*:

Application name: Name of the application in the Project Explorer;

Application index: Results from the list of all applications on the control to be queried with the `applist` command. Index 0 for the application in the list, and 1 for the second, and so on.

Also refer to

- [chapter 13.7 "PLC shell to query information" on page 210](#)

- [chapter "Generic device editor" on page 490](#)
- [chapter ""Library info" command" on page 676](#)

Tab '<Device name> I/O mapping'

Devices with I/O channels

This tab is displayed in device editors for devices with I/O channels. It shows the available channels and allows for the mapping of input, output, and memory addresses of the controller to variables or entire function blocks of the application.

In this way you create the 'I/O Mapping'.

The application responsible for the I/O handling is defined in the tab **PLC settings**.



The I/O mapping can also be created in the dialog **Edit I/O mapping**. A mapping list with search and filter functions for a complete Project Explorer is obtained there.



Mapping data types that are "too large"

If a variable of a data type, which is larger than one byte, is mapped to a byte address, the value of the variable is cut to the byte size!

For the monitoring of the variable value in the dialog **I/O mapping**, this means:

The value that the variable presently has in the project is displayed in the root element of the address. The individual present bit values of the byte are displayed in succession in the bit elements below that, but this may not be sufficient for the entire variable value.

The tab contains a table to edit the I/O mapping. The information displayed for the inputs and outputs originates from the device description.

Search	Input field for a character string to search for the mapping table. The search results are marked in yellow.
Filter	Selection list with filters for the I/O assignments shown in the mapping table: • "Show all" • "Show only outputs" • "Show only inputs" • "Show only unmapped variables" • "Show only mapped variables" • "Show only mappings to existing variables" • "Show only mappings to new variables"
 Add function block for I/O channel	Depending on the device, available if the channel entry is selected in the mapping table. Opens the "Select Function Block" dialog for selecting the function block that should be linked directly to the channel.
 Go to instance	Available if the entry is selected in the mapping table. Jumps to the corresponding entry on the <Device name> IEC Objects tab.

Variable	Depending on the device, the inputs and outputs of the device appear as nodes and below them, indented, the associated channels or, depending on the device, only the implicitly created device instance. The symbol indicates the channel type:  : Input  : Output A double-click on the cell opens an input field. - Option 1: The variable already exists. Enter complete path: <Application-Name>.<FunctionBlockName>.<VariableName>, example: <code>appl.PlcPrg.ivar</code>; Input assistant via . - Option 2: The variable does not exist yet; enter a simple name; automatically created internally as a global variable. Depending on the device, inputs or outputs can be linked directly to a function block. In this case, the  "Add FB for I/O channel" button is available.
Mapping	Mapping type:  : Already existing variable  : New variable
Channel	Symbolic channel name.
Address	Address of the channel, e.g. %IW0. Struck-through address: Indicates not to assign any further variables to this address. Reason: Although the variable specified here - as an already existing variable - is managed at a different memory location, ambiguities could result during the writing of the values, particularly with outputs.  : Indicates that this address was edited and specified. If the arrangement of the device objects in the Project Explorer changes, IndraLogic does not automatically adapt this address.
Type	Data type of the channel, e.g. BOOL. Structures or bit arrays defined in the device description are only displayed if they are part of the IEC standard and identified as IEC data types in the device description. Otherwise, the table cell remains empty. When mapping structured variables, the editor does not allow to enter the structural variable (for example to %QB0) and individual structural elements (for example to %QB0.1 and %QB0.2). Therefore, if there is a main entry with a sub-tree of bit channel entries in the mapping table, the following applies: Input a variable either in the line of the main entry or in the lines of the sub-elements (bit channels), but not in both.
Default value	Default value of the parameter applied to the channel: Appears only if the option Set all outputs to default value is activated in the PLC settings for the behavior of the outputs at stop. Note: From IndraWorks 14V22, an initialization value of the variable is used as default value for a mapping on an existing value. In the "Default value", editing is only possible a newly generated variable is mapped or if no mapping is entered. In previous versions, the user had to ensure that default value and initialization value are identical.
Unit	Unit for the parameter value, for example ms for milliseconds.

Description	Brief description of the parameter.
Current value	Actual value of the parameter applied to the channel; appears only in online mode.
Reset mapping	IndraLogic resets the mapping settings to the default values defined by the device description file.
Refresh variables	Definition for the device object about updating I/O variables. The default value is defined in the device description: • Use parent device setting: Update according to the settings of the superordinate device. • Activates 1 (use bus cycle task if not used in another task): IndraLogic updates the I/O variables in the bus cycle task if it is not used in another task. • Activated 2 (always in bus cycle task): IndraLogic updates all variables in each cycle of the bus cycle task, regardless of whether they are used or mapped to an input channel or output channel.



If a UNION is represented by I/O channels in the mapping dialog, it depends on the device whether mapping to the root element is also possible.

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter ""Edit I/O mapping" command" on page 609](#)
- [chapter ""PLC settings' tab" on page 497](#)

Tab '<Device name> IEC objects'



Please note that manually creating another instance of the device object can lead to malfunctions.

On this tab of the generic device editor, "objects" are listed that allow for access to the device from the IEC application. In online mode, this is used as the monitoring view.

For devices for which a function block instance is created implicitly, at least this instance is listed as an object in the table. This instance can be used, for example, in order to restart a bus or to query information from the application.

The device type determines whether this kind of device instance is available and which access options it has. Refer to the help for the special device configuration.

Instances of function blocks that are linked to inputs or outputs of the device are also displayed. Mapping of a function block to a channel is defined on the

"<Device name> I/O mapping" tab. The  "Go to Instance" command takes you directly to the affected object from there.

In addition, you can create more objects in the table that are not yet linked with a device channel.

In online mode, you can use the table of IEC objects as a monitoring view. This view also shows the current value, the address, and the comment for the function block variable at the channel. Finally, it provides the option of writing and forcing values.

 Add  Edit	Opens the "Select Function Block" dialog for creating a new instance or for editing the instance selected in the table.
 Delete	Deletes the selected entry
 Go to variable	Jumps from the selected entry directly to the corresponding mapping on the "<Device name> I/O mapping" tab.
"Variable"	The object name comprises the device name and the function block name. Example: EL2004_Relay. Changing the device name has an immediate effect. The part of the name after the device name can be edited.
"Mapping"	Mapping type, as in the "<Device name> I/O mapping" tab
"Type"	Data type, e.g. the name of the function block
"Value" "Prepared Value" "Address" "Comment"	Only available in online mode: Display of the current value, the address, and the comment for the variable at the channel. Option to enter a value for writing or forcing the variable.

Also refer to

- [chapter "Tab '<Device name> I/O mapping'" on page 504](#)

"Users and groups" tab



Recommendations on data security

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running:

If possible, avoid exposing the control and control networks to open networks and the Internet.

Use additional security layers such as a VPN for remote access and install a firewall. Restrict access to authorized personnel, and change any existing default passwords when commissioning and then in regular intervals.

On this tab of the generic device editor you edit the "online user management" of the control device.

Depending on how this is supported by the device, you can define user accounts and user groups. In combination with the configuration on the **Access rights** tab, you thus control access to control objects and files at runtime.

Prerequisite: the controller has a user management and you have login data in order to be able to log in to the controller.

In principle the handling of the dialog **Users and Groups** is very similar to that of the project user management. There is even a possibility to "import" user account definitions from the project user management.

Users

All currently defined users, and below them their memberships of user groups, are listed in a tree structure.	
 Add	Opens the Add user dialog. Define a new user name and a password. Confirm the password and create the new user account by clicking on OK .
 Import	Opens the Import user dialog. It displays all the user accounts defined in the project user management. Select the desired entries and click on OK in order to import them into the device-specific user management. NOTE: The passwords are NOT applied!
 Edit	Opens the Edit user <UserName> dialog. It corresponds to the dialog Add User and you can change the user account.
 Delete	Deletes the account of the currently selected user.

Groups

All currently defined groups, and below them the users assigned to them, are listed in a tree structure.	
 Add	Opens the Add group dialog. Define a new group name. From the list of defined users, select those that are to belong to the group. Confirm the selection with OK . The group is displayed in the tree.
 Import	Opens the Import user dialog. It displays all the user groups defined in the project user management. Select the desired entries and click on OK in order to import them into the device-specific user management.
 Edit	Opens the Edit group <GroupName> dialog. It corresponds to the dialog Add group and you can change the group definition.
 Delete	Deletes the currently selected group.

Also refer to

- [chapter 18.1.9 ""Device"" object and generic device editor" on page 490](#)
- [chapter "Generic device editor" on page 490](#)
- [chapter ""Access rights"" tab" on page 508](#)

"Access rights" tab



Recommendations on data security

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running:

If possible, avoid exposing the control and control networks to open networks and the Internet.

Use additional security layers such as a VPN for remote access and install a firewall. Limit access to authorized persons, change any existing standard passwords during the initial commissioning and continue to change them regularly.

In this device editor tab, define the device users for access rights to objects on the control.

Prerequisites:

- A user management has to be set up on the control unit.
- The option "tab show "Access rights"" has to be enabled in the IndraLogic options.

Activation: IndraWorks menu **Tools ▶ Options ▶ IndraLogic 2G ▶ Device editor**



This IndraLogic option is not supported by each control type and if required, can be overwritten by the device description!

Toolbar of tab

 Synchronization	Switches the synchronization between editor and user management on the device on and off. If the interface is not "pressed", the editor is empty or contains a configuration downloaded from the hard drive. If the interface is "pressed", IndraLogic continuously synchronizes the display in the editor with the current user management on the connected device. When enabling the synchronization while the editor still contains a user configuration not yet synchronized with the device, a query is displayed, asking to user to select what to do with the editor content. Options: • "Upload from device and overwrite editor content": The configuration is loaded to the editor and overwrites the current content. • " Load editor content to the device and overwrite the user management": The configuration contained in the editor is transmitted to the device and becomes active on the device.
 Import from hard disk	Open the standard dialog to select and import a user management configuration from the hard drive. On the "User and groups" tab, the file type is "Device user management files (*.dum)".
 Export to hard disk	Opens the standard dialog to save a file on the hard disk. The user management configuration is thus saved in an XML file. The data type is "device computer management files (*.drm)".
Device user:	User name of the user currently logged in.

Objects

The objects executed during runtime are listed in the tree structure. The objects are assigned to their object source and partially sorted in object groups. In the Rights view, configure the access options to a selected object for a user group.

Object source (top node in the object tree)

- File system objects
- Runtime system objects

Object groups and objects (indented).

Example: Device with child objects Logger, PlcLogic, Settings, UserManagement.

Rights

The table applies to the object selected in the tree. Displays the currently configured rights for possible actions to this object for each user group.

- Add/remove
- Modify
- View
- Execute

The symbols represent the access rights:

-  : Access (action) is permitted explicitly.
-  : Access (action) is explicitly prohibited.
-  : Grayed out plus or minus symbol: The right can be applied from the parent object.
-  : The right was not explicitly set or refused, not even in the parent object. Access is not possible.
- No symbol: Several objects are selected that do not have uniform access rights.

Change the rights by clicking on the symbol.

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter ""Users and groups" tab" on page 507](#)

"Task list" tab

This sub-dialog of the device editor displays a table of inputs and outputs as well as their assignment to the defined tasks.

The information only becomes visible after code has been generated for the application. It is used for troubleshooting, because it shows where inputs or outputs are used in several tasks with different priorities. Multiple use can lead to undefined values through overwriting.

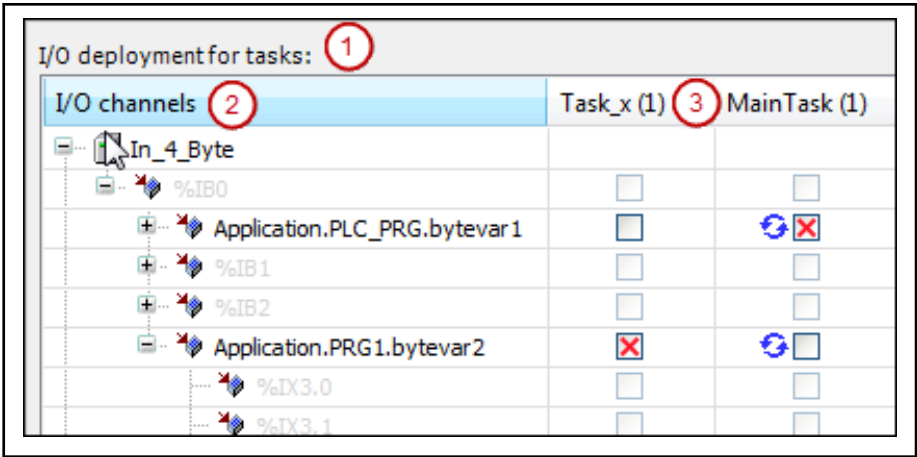


Fig. 18-8:

I/O channels (2)	All inputs and outputs of the connected devices. The display corresponds to that in the dialog I/O mapping of the device editor. By double-clicking on an input or output you can open the associated I/O mapping editor.
<TaskName> (3)	A column appears for every task defined in the task configuration. The title contains the task name and the priority. The priority of the tasks decreases from first to the last column. A red cross is displayed in the box for inputs and outputs written or read by a task: . In addition, the task defined as a bus cycle task in the PLC settings of the device editor is marked at these points with a blue double arrow symbol . Following a mouse-click on the title cell, only the I/Os assigned to this task are displayed. Following a mouse-click on the I/O channels cell, all channels are shown again.

Tab. 18-4: I/O list for tasks (1)

Also refer to

- [chapter "Generic device editor" on page 490](#)
- [chapter "Tab '<Device name> I/O mapping'" on page 504](#)
- [chapter "'PLC settings' tab" on page 497](#)
- [chapter 9.15.1 "Creating task configurations" on page 137](#)

"Status" tab

This tab of the generic device editor displays status information, for example "Running" or "Stopped", and specific diagnostic messages from the respective device, also information about the card used and the internal bus system.

Also refer to

- [chapter "Generic device editor" on page 490](#)

"Information" tab

This tab of the generic device editor displays general information that originates from the device description file: name, vendor, categories, version, order number, description, if necessary an illustration.

Also refer to

- [chapter "Generic device editor" on page 490](#)

18.1.10 “GlobalTextList” object

Symbol: 

This object is used to manage and translate texts written as static text in the project in the visualizations.

The object contains a table with these texts. If a text is written in a visualization in an element below the property **Texts**, IndraLogic automatically adds a line to the table. No new text can be written here. Only existing text can be edited.

IndraLogic additionally provides the following commands to consolidate the **GlobalTextList**:

- **Check visualization text IDs**
- **Update visualization text IDs**
- **Remove unused text list entries**

The object does not exist more than once in the "General module folder".

ID	Unique text ID
Default	Source text as character string with up to one formatting specification, e.g. <code>Information A: %i options</code> . If there is no translation under one language column, IndraLogic uses this text. Double-click into the field to edit the text.
The table includes any number of added language columns. A language column is named with a country code specified when creating the column with the Add language command.	
<LanguageCode>	Name of the language as country code (e.g., <code>en-US</code>). This column contains the translation of the text written under Standard . If the country code is selected as language in the visualization manager, the visualization outputs the translation at runtime. A visualization at runtime can select another language upon request. Double-click into the field to edit the text.

Also refer to

- [chapter ""Add language" command" on page 684](#)
- [chapter ""Remove language" command" on page 687](#)
- [chapter ""Import/export text lists" command" on page 686](#)
- [chapter ""Create global text list" command" on page 684](#)
- [chapter ""Remove unused text list entries" command" on page 687](#)
- [chapter ""Check visualization text IDs" command" on page 687](#)
- [chapter ""Update visualization text IDs" command" on page 688](#)
- [chapter 18.1.23 ""Text list" object" on page 553](#)
- [chapter 9.8 "Managing text in a text list" on page 113](#)

18.1.11 “GVL” object - Global Variable List

Symbol: 

A global variable list is used for the declaration, editing and display of global variables.

Add a GVL using the command **Add ► Global variable list** of the application or the "General module folder".

- If you insert a GVL under an application in the Project Explorer, the variables are valid within this application.
- If you add a GVL in the "General module folder", the variables are valid for the entire project.

You can make settings for the editor of the object in the dialog **Tools ► Options** in the categories **Declaration editor** and **Text editor**.

You can convert the variables of a GVL into network variables and thus use them for data exchange with other devices in the network. To do this, define corresponding properties for the GVL in the dialog **Properties**, tab **Network variables**.

GVL for configurable constants ("parameter list") in Libraries

It can be intended that the value of a global constant provided via a library can be replaced by a value defined by the application.

For this purpose, the constant has to be declared in a "parameter list" in the library. If the library is integrated in the application, the value can be edited in the "Parameter list" tab of the library manager of the application.

Handling parameter lists

`lib1.library` provides an array variable `g_Array`. Its size is defined by the global constant `g_c_Arraysize`.

The library is integrated in different applications using different array sizes each. Thus, the global constant from the library should be overwritten with an application-specific value.

How to proceed:

When creating `lib1.library`, the global constant `g_c_Arraysize` is defined in a special type of Global Variable list (GVL), the so-called "Parameter list".

Therefore, use the command **Add ► Parameter list** to add a "Parameter list" object, in this example called "Param".

The declaration of the variable `g_c_Arraysize` is entered into the editor of this object. This editor is identical to the standard GVL editor.

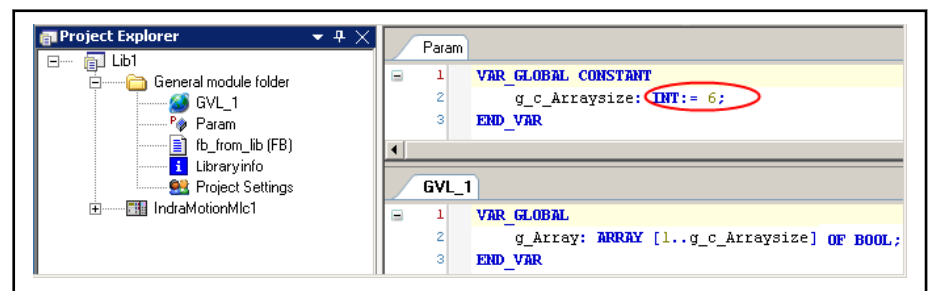


Fig. 18-9: Parameter list "Param" in the "Lib1.library"

The "Lib1.library" is created. The array `GVL_1.g_Array` is preset to six elements.

The "lib1.library" is then inserted into the library manager of an application project:

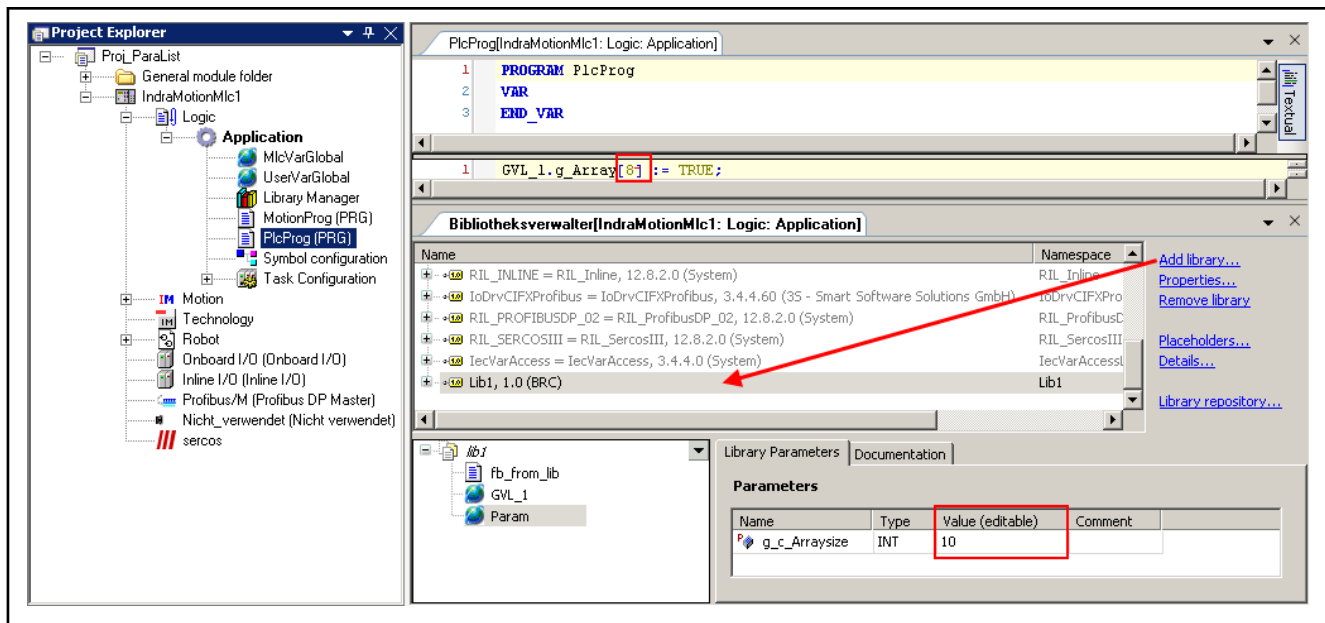


Fig. 18-10: Editing the parameter "g_c_Arraysize" in the library manager of a project

Select the library in the upper window of the library manager to display the function block tree.

Select Param to obtain the "Parameter list" tab showing declarations.

Select the cell in the column "Value (editable)" and press the spacebar to open the input field. Enter the new value for g_c_Arraysize. It is used in the local validity range of the library when the input field is closed.

In the example above, the value is now set to 10 and the array element 8 is assigned in the PlcProg program.



For nested libraries, the parameterizable library has to be on top.

Also refer to

- chapter 9 "Programming applications" on page 83
- chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288
- chapter "The "Properties"- "Network variables" dialog" on page 708
- chapter "'Options" – "Declaration editor" dialog" on page 722
- chapter "'Options" dialog - Text editor" on page 731

18.1.12 'Persistent variable list' object

Symbol:

The object contains the declaration of global persistent variables in the declaration section `VAR_GLOBAL PERSISTENT RETAIN .. END_VAR`.

The variables are stored in special non-volatile memory.

The persistence editor shows the variables as a list. The displayed list does not influence the persistence behavior of the variables, but only the list stored internally in the process image. The list contains all variables ever declared in chronological order. Variables that were removed are replaced by a placeholder and continue to exist as a gap.

The declaration section can also contain instance paths, which refer to locally declared persistent variables and were created with the **Declarations ► Add All Instance Paths** command.



Before you decide how to set up persistence for an application, it would be helpful for you to be familiar with the use cases described in the Data Persistence chapter. Moreover, it is helpful if you can differentiate between the mechanisms of persistent variables, retain variables, variables of the Persistence Manager, and recipe variables.

Commands The following commands are provided in the persistence editor:

- Command **Declarations ► Add All Instance Paths**
- Command **Declarations ► Reorder list and clear gaps**

Also refer to

- [chapter ""Rearrange list and clean gapson page 678](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)
- [chapter 9.16.2 "Preserving data with retain variables" on page 142](#)
- [chapter "Command 'Add all instance paths'" on page 679](#)

18.1.13 "Image pool" object

The **Image Pool** object contains a table with image ID assignments.

ID	ID of the image. Use this ID to reference the image in the visualization for example.
File name	File path of the image file. Click on  to open the Select image dialog.
Image	Show a thumbnail of the image.
Link type	Opens the Select image dialog. The type of linking is defined there.

"Select image" dialog

Image file	Directory and name of the image file, e.g. "C:\Program Files\images\logo.bmp". IndraLogic supports the following image formats: BMP, EMF, GIF, ICO, JPG, PNG, SVG and TIFF. Please note that a control does not support all formats. The usefulness of screens in the scalable vector format *.svg depends on the operating system. Any necessary information is located in the device description of the hardware vendor.

What should happen to the image file?

Saving the link	IndraLogic only saves the link. Thus, IndraLogic updates a change of the image file automatically in the image pool. Ensure that the path of the image file does not change. Saving the project as project archive, IndraLogic embeds the image file into the project archive.
Saving the link and embedding into the project	IndraLogic copies the image to the image pool. However, the linking information remains. IndraLogic can thus detect when an image file changes and update the image in the image pool if required. The response is controlled by the options of the following table. Embedded image files increase the memory requirement of the project.
Embedding into the project	IndraLogic copies the image to the image pool. If the image file is changed again afterwards, then it is not updated in the project. For, the images have to be embedded into the project. Embedded image files increase the memory requirement of the project.

When the image file changes

These options are only available if the Remember the link and embed into project checkbox is selected as described above.	
Reload the file automatically	IndraLogic updates the image file automatically in the project without prompt.
Prompt whether to reload the file	After an image file has changed, send a query whether the image file should be updated in the project.
Do nothing	IndraLogic does not update the image file in the image pool.

Also refer to

- [chapter 9.9 "Using image pools" on page 121](#)

18.1.14 Object 'Library manager'

Symbol: 

The Library Manager lists all libraries that were linked in the project for creating the application, and it notifies about its type, properties, and contents.

You can add or remove other linked libraries in the list, as well as modify their properties.

The library manager is subdivided into three views.

- Upper window section: List of the integrated libraries
- Lower left window section: Tree structure with all function blocks of a library selected in the upper window section
- Lower right window section: Documentation of a function block selected in the tree

Also refer to

- [chapter 16 "Creating and managing libraries" on page 219](#)
- [chapter 16.2 "Adding a library to the application" on page 220](#)

List of the integrated libraries

List of all libraries currently implemented into the project. If a library depends on other libraries (referenced libraries), these referenced libraries are automatically integrated.

Displayed in gray fonts	The library was added to the project automatically by means of a plug-in.
Displayed in black fonts	The library was added to the project automatically by means of the "Add library" command.
"Name"	Display of the integrated library in the following syntax: "<PlaceholderName> = <LibraryName>, <Version> (<Company>)": "<PlaceholderName>": If it is a placeholder library for a library, then the placeholder name is before a = "<LibraryName>": Name of the library that is used for management in the library repository. "<Version>": Version referenced upon its first integration. "(<Company>)": Manufacturer (optional)
Namespace	Namespace for unique access to the contents of the library. The namespace is used as prefix before a function block identifier: <Namespace>.<LibraryFunctionBlockID> The namespace usually coincides with the library name. Note: If the library has the property <code>LanguageModelAttribute</code> "qualified-access-only", you have to access the library function block in the application code using the namespace. A qualified (unique) access is enforced. Modify the standard namespace for local use (within the project) in the Properties dialog.
Effective version	Version of the library after the resolution. This version is currently used in the project. Prerequisite: If the library manager is below the device node "Application" and if a placeholder library is selected, the effective version is displayed. Example: 15.xx.0.0 A placeholder library implemented below an application is resolved as follows: If you assigned a special resolution to the placeholder library in the "Placeholders" dialog, the selected library is loaded. Other resolutions are ignored. If no special resolution is given, then a check is performed as to whether or not a resolution is specified in the device description and library profile of the application. The first search hit is applied.
	Symbol with tooltip to inform on the current device-dependent resolution. Prerequisite: If the library manager is in the "General module folder" and a placeholder library is selected, the symbol is displayed. A placeholder library implemented in the "General module folder" is resolved as follows: An application-dependent check is performed as to whether a resolution is specified in the device description. Then the library profile is checked. The first detected resolution is used. If you have assigned a special resolution to the placeholder library in the "Placeholders" dialog, then this will always be ignored. The result is shown in the tooltip of the symbol
	Library that is signed with a trusted certificate
	Library that is signed with a trusted certificate, but references at least one unsigned library
	Library that is signed with a private key and token
	Library that is not signed, or signed with an untrusted or expired certificate. In the case of an untrusted certificate, the "Trust Certificate" command is provided in the context menu.

	Library that is defined as optional and not currently available
	Library whose status is being determined
	Licensed library for which no valid license is currently available
	Library symbol for a library that cannot be loaded because its signature (encryption) could not be verified

Commands in the library manager

 Add library	Opens a dialog to select a library. All libraries installed in the library repository are provided
 Delete library	Removes the presently selected library from the project
 Properties	Opens a dialog to display and edit the properties of the presently selected library.
 Details	Opens a dialog with details for the presently selected library (general information, contents, properties, license information)
 Try to reload library	If you select a library marked as not found, you can attempt to load it into the project again using this command.
 Download missing libraries	IndraLogic browses for the missing libraries in the download servers specified in the project options. After that you can download and install the library.
 Placeholders	The Placeholders dialog opens. The current resolution is displayed and can be edited
 Library repository	Opens the Library repository dialog for installing and uninstalling libraries and for defining library locations.
 Icon legend	Opens the Information dialog with a legend of the icons that display the current status of a library in the list of integrated libraries (see above)
 Trust certificate	Only in the context menu of a library selected in the Library Manager, in which the library has been signed with an untrusted certificate. The command turns the untrusted certificate into a trusted certificate and the prepended icon changes from  to 
Export library	Available in the context menu of a library selected in the library manager. Opens the default dialog for saving the library file in the file system.

Also refer to

- [chapter ""Add library" command" on page 672](#)
- [chapter ""Properties" command" on page 673](#)
- [chapter ""Library repository" command" on page 636](#)
- [chapter ""Placeholder" command" on page 675](#)

Tree structure of all function blocks of a selected library

All library function blocks that were integrated with the library are listed in the tree structure.

Prerequisite: A library is selected in the upper view.



The usual sorting and search functions are available in the menu bar.

Documentation for the library function block selected in the lower left view

"Inputs/outputs" tab	Interface (inputs/outputs) of the library function block
Tab "Graphical"	Graphical display of the interface

Tab "Documentation"	Documentation for the library function block Note: As a library developer, comply with the rules for documentation inclusion in "Guidelines for library development".
Tab "Parameter list"	Prerequisite: The library project contains a parameter list. Change the values of these parameters in the column "Value (editable)".



Using parameter lists is only permitted if the library is in the top position of the hierarchy.

Also refer to

- [chapter 16.1 "Information for library developers" on page 219](#)

18.1.15 "Global Network Variable List (sender)" object

Symbol: 

A network variable list (sender) is used for declaring and listing global variables that should be sent to network variable lists (receiver) of other devices or network projects.

Click on **Add ► Global Network Variable List (sender)** of an application to add the object to the Project Explorer.

You can configure the protocol and transfer parameters in the **Add Global Network Variable List (Sender)** dialog or **Properties** dialog of the object in the **Network variables** tab.

"Add Global Network Variable List (sender)" dialog

Function: This dialog defines the network properties for the sender NVL. When you close the dialog, IndraLogic adds the sender NVL of the application to the Project Explorer.

Call: Command **Add ► Global Network Variable List (sender)** when the application is selected in the Project Explorer.

This dialog corresponds to the **Network Variables** tab in the **Properties** of the network variable list object.

Also refer to

- [chapter 10.3 "Network variables" on page 158](#)
- [chapter "The "Properties"- "Network variables" dialog" on page 708](#)
- [chapter 18.1.16 ""Global Network Variable List \(receiver\)" object" on page 519](#)
- [chapter 10.3.1 "Configuring a network variable exchange" on page 160](#)

18.1.16 "Global Network Variable List (receiver)" object

Symbol: 

The object lists the received network variables and displays the information: Network information and transmission information and sender.

You add the object to an application by clicking **Add ► Global Network Variable List (receiver)**.

The Network Variable List (receiver) shows the received network variables.

The received network variables were declared in the Network Variable List (sender) of another device or project. You cannot change the network variables in the object editor.

The object editor consists of two parts:

- Information about the sender and transfer log of the list
- List of declarations of the network variables

"Add Global Network Variable List (receiver)" dialog

Function: This dialog defines the receiver NVL to a sender NVL and adds the receiver NVL to the application object in the Project Explorer.

Call: Menu command **Add ► Global Network Variable List (receiver)** after selecting the application object.

Task	Task of the current application that controls the variables to be received.
Sender	<i>Drop-down list</i> • Available sender NVLs of another device in the project • Import from file: Required if the necessary sender NVL is defined in another project. For this, the necessary sender NVL must have been generated in another project as GVL export file *.gvl in the properties dialog of the NVL in the Link To File tab.
Import from file	File name in GVL export file *.gvl format if you have selected Import from file for Sender .

Also refer to

- [chapter 10.3 "Network variables" on page 158](#)
- [chapter 10.3.1 "Configuring a network variable exchange" on page 160](#)
- [chapter 18.1.15 "'Global Network Variable List \(sender\)' object" on page 519](#)

18.1.17 Object "POU"

Icon: 

An object of type **POU** is a programming organization unit in an IndraLogic project. Write the source code for the control program to the POU.

There are the following POU types:

- Program
- Function
- Function block

Add a **POU** object to the application or to the "General module folder" via **Add** from the context menu. When adding a POU, specify the POU type and the implementation language.

Other programming objects (method, action, etc.) can be added to these objects as objects.

Call POU

Certain POU can call other POU. Recursion is not permitted.

When calling POU via the namespace, IndraLogic scans the project for the POU to be called as per the following sequence:

1. Current application
2. **Library manager** of the current application
3. "General module" folder

4. Library manager in the "General module" folder



When calling a POU, which exist with the same name in a library used in the application and as object in the "General module folder", the following applies:

There is no syntax that can be used to call the POU in the "General module folder" via its name.

If the names of the POUs are identical in the "General module folder" and in a library, add the namespace of the library to the name of the POU.

Also refer to

- [chapter 18.1.17 "Object "POU"" on page 520](#)
- [chapter 18.3.6 ""Properties" dialog" on page 704](#)

"Add POU" dialog

Function: This dialog is used to configure a new POU acc. to the IEC 61131-3 standard. Thus, a POU can either be a program, a function or a function block.

Call: Context menu "Application", context menu "General module" folder.

Name	POU name
Program	
Function block	• Advanced: Specifying or selecting a basic function block in terms of the object-oriented programming. Is specified with the keyword EXTENDS in the function block declaration. • Implemented: Specifying or selecting an interface in terms of the object-oriented programming. Is specified with the keyword IMPLEMENTS in the function block declaration. • Final: Derived access is not permitted. This means that a function block cannot be extended by another function block. This allows an optimized code generation. • Abstract: Identifies that the function block has a missing or incomplete implementation and cannot be instantiated. Abstract FBs are used exclusively as base function blocks and the implementation typically occurs in a derived FB. • Access specifier – PUBLIC: Does not corresponds any access modifier specification – INTERNAL: Access to the function block is limited to the namespace (library). • Method implementation language: Select the option Implemented and an implementation language for all method objects, which IndraLogic generates via the implementation of the interface, can be selected. The Method implementation language does not depend on the implementation language of the function block.
Function	Note: Not available if Sequential Function Chart (SFC) is selected in the drop-down list Implementation language. Return type: Selection list for the data type of the return value
Implementation language	Selection list for the implementation language of the POU

Also refer to

- [chapter ""Function block" object \(FUNCTION_BLOCK\)" on page 523](#)
- [chapter ""Program" object \(PROGRAM\)" on page 522](#)
- [chapter ""Function" object \(FUNCTION\)" on page 525](#)
- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)

"Program" object (PROGRAM)

A program is a POU that provides one or more values upon execution. All values remain after one program execution until the following one. The call sequence of the programs in an application is defined in the task objects.

Add a program to the application or the "General module folder" via **Add ► POU**.

For the suffix (**PRG**) of the program POUs, go to the Project Explorer below the application or to the "General module folder".

The editor of a program consists of the declaration section and the implementation section.

The upper line of the declaration section includes the following declaration:

Program:

```
PROGRAM <program>
```

Calling a program

Programs and function blocks can call a program. A program call is not permitted in a function. There are no program instances.

When a POU calls a program and values of the program change, the changes remain up to the next program call. The values of the program also remains if a different POU calls next.

This is different from calling a function block.

When calling a function block, only the values of the respective instance of the function block change. The changes have to be considered only when a POU calls the same instance.

The input parameters and/or output parameters for a program can also directly be set upon the call.

Syntax:

```
<program>(<input variable> := <value>, <output variable>
=> <value>):
```

When inserting a program call via the input assistant and the **Insert with arguments** option is enabled in the input assistant, IndraLogic adds input parameters and/or output parameters to the program call as per the syntax.

Examples

Calls:

IL:

1	CAL	PlcProg(
		in1:= 2)
	LD	PlcProg.out2
	ST	erg

Fig. 18-11:

With parameter assignment:

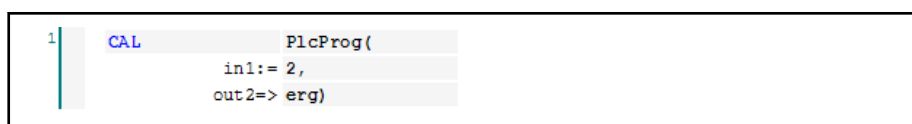


Fig. 18-12:

ST:

```
PlcProg()
erg := PlcProg.out2;
```

With parameter assignment:

```
PlcProg(in1:=2, out1=>erg);
```



Fig. 18-13:

Also refer to

- [chapter 18.1.17 "Object "POU" on page 520](#)
- [chapter 9.15 "Task configuration" on page 136](#)

"Function block" object (FUNCTION_BLOCK)

A function block is a POU providing one or multiple values at execution.

Add the object to the application or to the "General module folder" via **Add ► POU**.

The function block POUs are provided with the suffix **(FB)** in the Project Explorer or in the "General module folder".

They always call a function block via the instance that is a function block copy.

The editor of a function block consists of the declaration section and the implementation section.

The values of the output variables and the internal variables remain up to the next execution. This means that if the function block is called multiple times with the same input variables, the same output values do not necessarily result.

In addition to the functionality described in the IEC 61131-3, function blocks in IndraLogic can also be used for the following functionalities of the object-oriented programming:

- Extension of a function block
- Implementing interfaces
- Methods
- Properties

The upper line of the declaration section includes the following declaration:

```
FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> |
IMPLEMENTS <comma-separated list of interfaces>
```

Calling a function block

It is always called via an instance of the function block. When calling a function block, only the values of the respective instance change

Instance declaration:

<instance>: <function_block>;

Access a function block variable in the implementation section as follows:

<instance>.<variable>



Note the following:

- From outside the function block instance, only the function block input variables and output variables can be accessed (not the internal variables).
- Access to a function block instance is limited to the POU, in which the instance is declared, unless it is declared globally.
- When calling the instance, values can be assigned to the function block variables.

Example

Access to function block variables:

The function block **FB1** is provided with the input variable `iVar1` of type `INT` and the output variable `out1`.

The variable `iVar1` is called from the program **Prog** in the following.

```
PROGRAM Prog
VAR
inst1:FB1;
END_VAR
```

```
inst1.iVar1 := 33; (* FB1 is called and the value 33 is assigned to the variable iVar1 *)
inst1();          (* FB1 is called, that's necessary for the following access to the
ires := inst1.out1 (* the output variable out1 of the FB1 is read *)
```

In FBD:

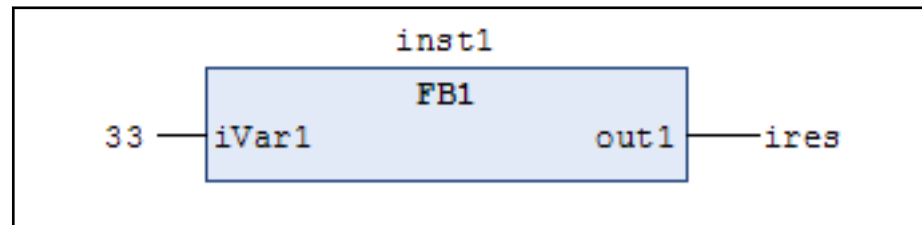


Fig. 18-14:

Assigning variable values upon call:

In the textual languages IL and ST, values can be directly assigned to input variables and/or output variable when calling the function block.

A value is assigned to an input variable with `:=`

A value is assigned to an output variable with `=>`

Example

The instance `CMD_TMR` of the timer function block is called with assignments for the input variable `IN` and for `PT`. Afterwards, the output variable `Q` of the timer is assigned to variable `A`

```
CMD_TMR (IN := %IX5, PT := 100=);
A:= CMD_TMR.Q
```



When inserting a function block instance via the **Input assistant** and the option **Insert with arguments** is selected in the **Input assistant** dialog, IndraLogic inserts the call with all input variables and output variables.

Then, insert the value assignment.

In the example, above, IndraLogic inserts the call as follows:

```
CMD_TMR (IN:= , PT:= , Q=> ).
```

Also refer to

- [chapter 18.1.17 "Object "POU"" on page 520](#)
- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)
- [chapter ""Method" object " on page 532](#)
- [chapter "Object 'Property'" on page 537](#)

"Function" object (FUNCTION)

A function is a POU that provides exactly one data element upon execution and its call can occur in textual languages as operator in expressions. The data element can also be an array or a structure.

Add the object to the application or to the "General module folder" via **Add ► POU**.

In Project Explorer below the "Application" or in the "General module folder", the function POUs have the suffix **(FUN)**.



Functions are not provided with internal status information, That means that the functions do not save the values of their variables up to the next call.

Calling a function with the same input variable values always provides the same output value. For this reason, functions may not use global variables and addresses!

The editor of a function consists of the declaration section and the implementation section.

The upper line of the declaration section includes the following declaration:

Syntax

```
FUNCTION <function> : <data type>
```

Declare the input variables and function variables below.

The output variable of a function is the function name.



If you declare a local variable in a function as **RETAIN**, this has no effect!

In this case, IndraLogic outputs a compiler error.



In IndraLogic 2G, explicit and implicit parameter assignments cannot be mixed in function calls.

Thus, either explicit or implicit parameter assignments can be used in function calls.

The sequence of parameter assignments upon the function call is not specified.

Calling a function

In ST, a function call can be used as an operand in expressions.

In SFC, a function call can only occur in step actions or transitions.

Examples

Function with declaration section and a line of implementation code

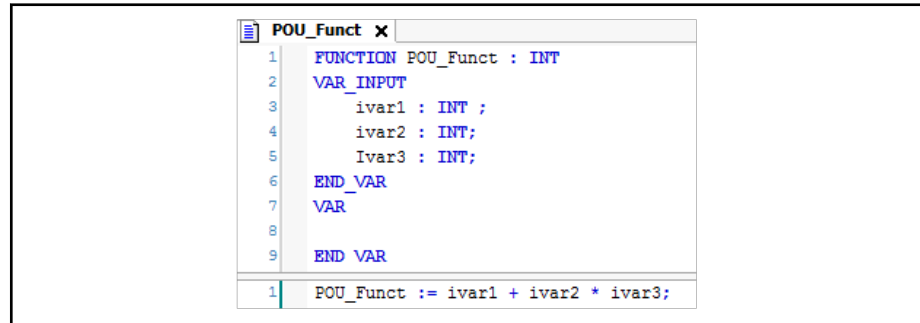


Fig. 18-15:

Function calls:

ST:

```
result := POU_Funct(5,3,22)
```

IL:

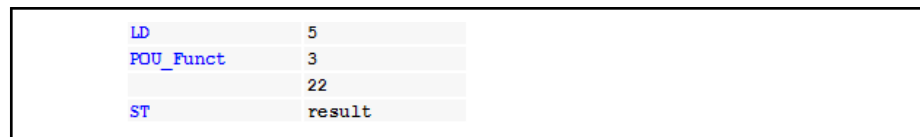


Fig. 18-16:

FBD:

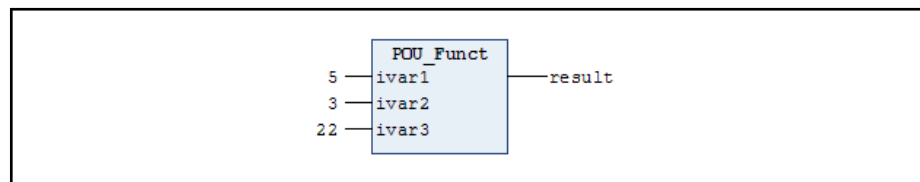


Fig. 18-17:

Functions with additional outputs

According to the IEC 61131-3 standard, functions can have additional outputs. Declare the additional outputs in the function between the keywords VAR_OUTPUT and END_VAR. Call the function acc. to the following syntax:

Syntax

```
<function>(<function output variable> => <output variable 1>, <function output variable n> => <output variable n>)
```

Example

The function fun is defined with the two input variables in1 and in2. The output variables of the function fun is written to the locally declared output variables loc1 and loc2.

```
fun(in1 := 1, in2 := 2, out1 => loc1, out2 => loc2);
```

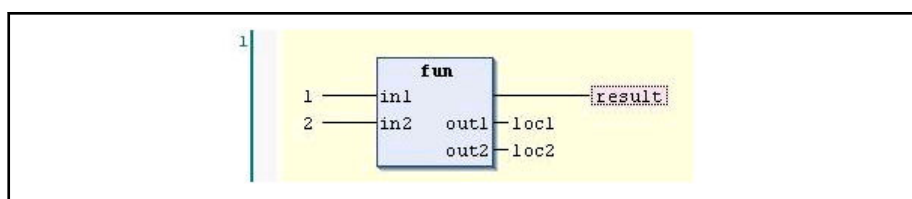


Fig. 18-18: Function with return value and two output variables

Transferring instance data via VAR_INPUT

The declaration of a function block instance within a function is not permitted. However, data of an FB_Instance can be transferred to a function for further processing via VAR_INPUT:

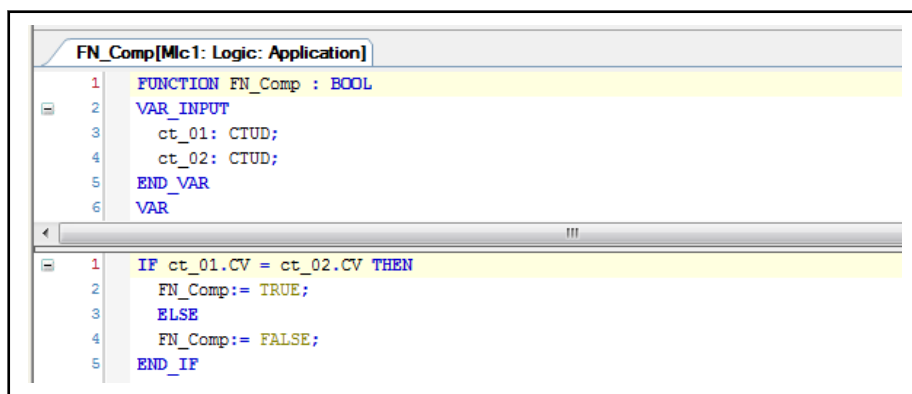


Fig. 18-19: Transferring two instances to one function via VAR_INPUT

The preceding example shows the transfer of data sets of two (counter) instances to the FN_Comp function to compare the CV outputs. The instances are processed in the superordinate POU.

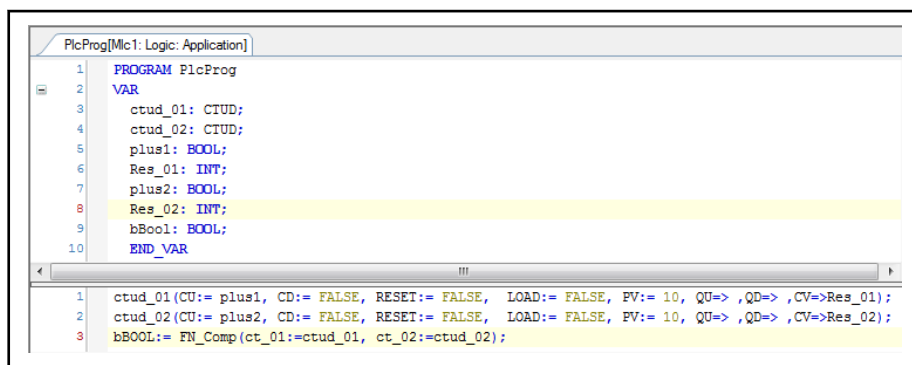


Fig. 18-20: Processing counter instances and transferring their data sets to the function

Transferring via POINTER in VAR_INPUT

Information computed in a POU (function, function block) can be externally output via a pointer declared under VAR_INPUT.

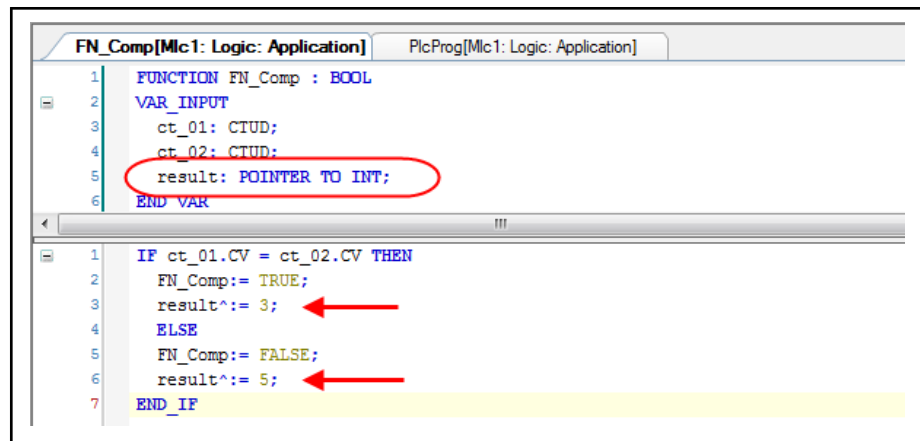


Fig. 18-21: Pointer at the interface of a function

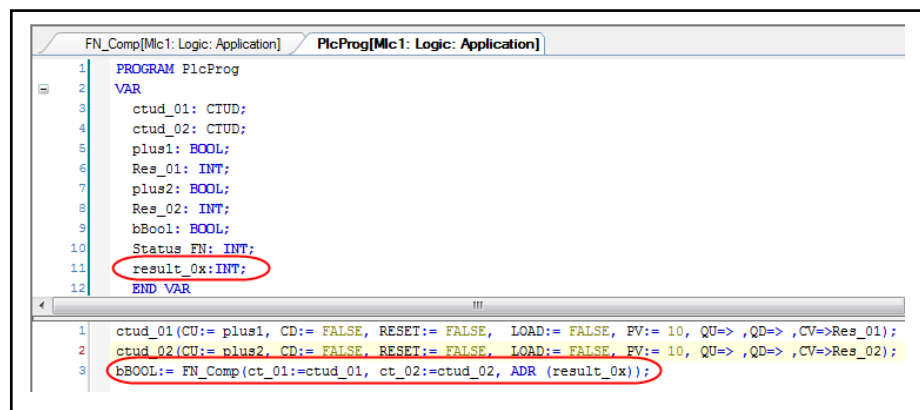


Fig. 18-22: Applying information via ADR operator

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'"](#) on page 619.

Also refer to:

- [chapter 17.5.12 "Data type 'POINTER'"](#) on page 359

Transferring via REFERENCE TO to VAR_INPUT

Information computed in a POU (function, function block) can be externally output via a pointer declared under REFERENCE TO.

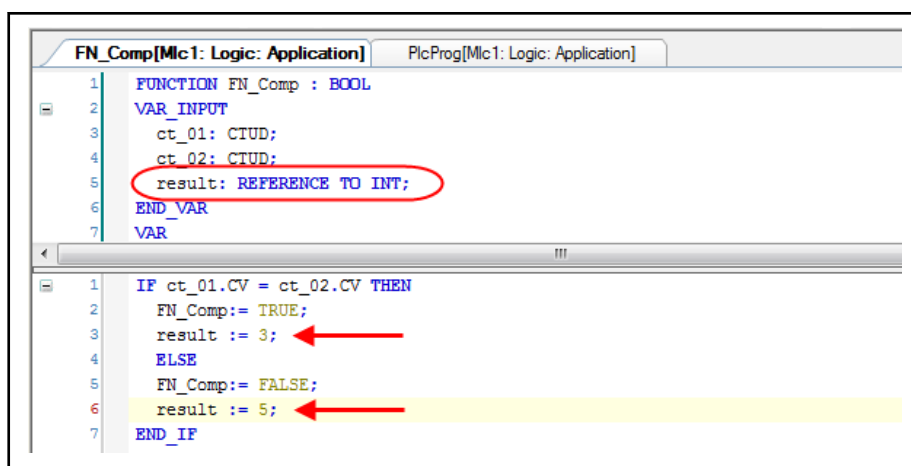


Fig. 18-23: REFERENCE TO at the interface of a function

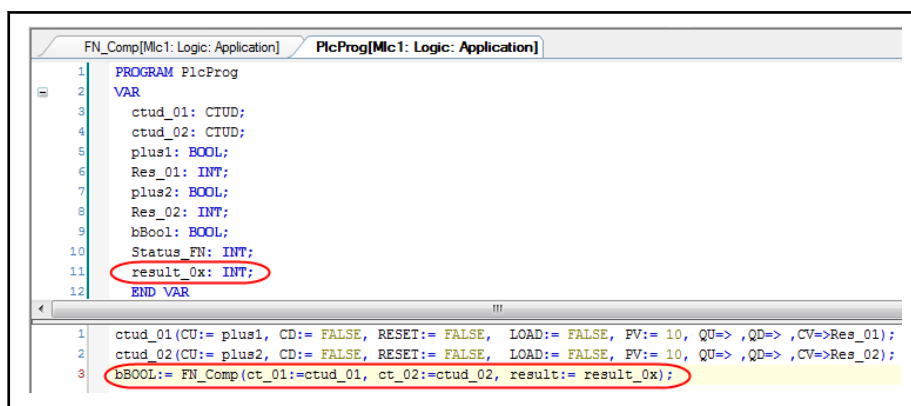


Fig. 18-24: Applying information via assignment

NOTICE

If you use pointers to addresses, then the contents of addresses can be moved during an online change.

If you use absolute addresses, then the contents of addresses does not change during an online change.

Also refer to [chapter "Command 'Online change'" on page 619](#).

Also refer to

- [chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)

Also refer to

- [chapter 18.1.17 "Object "POU"" on page 520](#)

"Interface" object (INTERFACE)

Symbol:

Keyword: INTERFACE

An interface is a means of object-oriented programming. The object `ITF` describes a set of method and property prototypes.

In this context, prototype means that the methods and properties contain only declarations and no implementation.

This allows different function blocks having common properties to be used in the same way.

An object **ITF** is added to the application or the "General module folder" via **Add ► Interface**.

Inheritance	
Name	Interface name
Advanced	 Extends the interface that you enter in the input field or via the input assistant
	 This means that all methods of the interface that extend the new interface are also available in the new interface.

Tab. 18-5: Adding an interface

You can add the objects **Interface property** and **Interface method** to the object **ITF**.

Interface methods may contain only the declarations of input, output and VAR_IN_OUT variables, but no implementation.

So that you can also use an interface in the program, there must be a function block that implements this interface.

That means:

- the function block contains the interface in its IMPLEMENTS list in its declaration part
- the function block contains an implementation for all methods and property prototypes of the interface

A function block can implement one or more interfaces. You can use the same method with identical parameters, but different implementation code in different function blocks.

Note the following

- You may not define variables within an interface. An interface has no implementation part and no actions. Only a collection of interface methods is defined, in which you may define only VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT variables.
- IndraLogic always treats variables declared with the type of an interface as references.
- A function block that implements an interface must contain implementation code for the methods of the interface. You have named the methods exactly as in the interface and the methods contain the same VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT variables as in the interface.



Interface references and online change

The following can result when using a compiler version < 3.4.1.0: If a function block changes its data, as variables are added or deleted or as the type of variables changes, IndraLogic copies all instances of the function block to a new storage location. In this case, however, an interface reference refers not to the new memory location, but still to the old one.

In case of compiler versions $\geq 3.4.1.0$, IndraLogic automatically re-addresses the interface references so that IndraLogic also references the correct interface in case of an online change. IndraLogic requires additional code and more time for this, so that jitter problems can occur depending on the number of objects concerned.

Therefore, IndraLogic displays the number of variables and interface references before executing the online change. Then, decide whether the online change should be executed or aborted.

Example:

Definition of an interface and its use in a function block

You have inserted the interface **ITF** below the application.

The interface contains the interface methods **METHod1** and **METHod2**.

ITF, **METHod1** and **METHod2** contain no implementation code. You insert the required variable declarations only in the declaration part of the methods.

If you subsequently insert a function block in the Project Explorer that implements the interface **ITF**, IndraLogic automatically also inserts the methods **METHod1** and **METHod2** below the function block (**FUNCTION_BLOCK POU IMPLEMENTS ITF**).

Here you can implement function-block-specific code in the methods. The **interface methods** become **POU methods**.

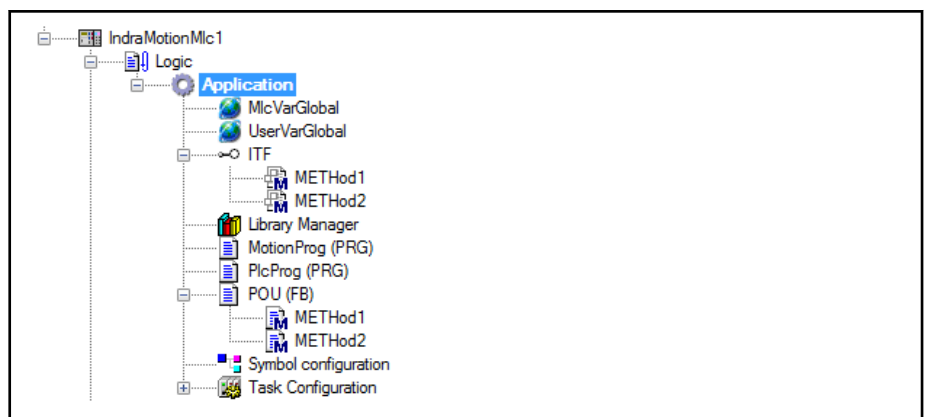


Fig. 18-25:

- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)
- [chapter 9.18.3 "Extending an interface" on page 149](#)
- [chapter ""Interface method" object" on page 534](#)
- [chapter ""Interface property" object" on page 534](#)
- [chapter ""Method" object " on page 532](#)
- [chapter "Object 'Property'" on page 537](#)

“Method” object

Symbol: 

Keyword: METHOD

Methods are an extension of the IEC 61131-3 standard and a tool for object-oriented programming which is used for data encapsulation. A method contains a declaration and an implementation. However, unlike a function, a method is not an independent POU, and it is subordinated to a function block or program. A method can access all valid variables of the superordinate POU.

Use interfaces for the organization of methods.

Add method

Methods can be added to programs or functions.

Select **Add ► POU method** in the respective context menu.

The "Add method" dialog opens.

Declaration

- The variables of a method contain temporary data that are valid only during the execution of the method (stack variables). All variables that are declared and implemented in a method are reinitialized each time the method is called.
- Like functions, methods can have additional outputs. You must assign these additional outputs in the method call.
- Depending on the declared access specifier, a method can be called only within its own namespace (*INTERNAL*), only within its own POU and its derivatives (*PROTECTED*), or only within its own POU (*PRIVATE*). The method can be called from anywhere for *PUBLIC*.

Interface methods can have declared input, output, and *VAR_IN_OUT* variables, but do not contain an implementation.

Also refer to

- [chapter ""Interface method" object" on page 534](#)

Implementation

- Access to function block instances or program variables is permitted in the implementation of the method.
- The *THIS* pointer allows for access to its own function block instance. Therefore, the pointer is permitted only in methods that are below a function block.
- A method cannot access *VAR_TEMP* variables of the function block.
- A method can call itself recursively.

Dialog “Add method”

Function: Defines a method below the selected POU when the dialog is closed.

Call: Via the context menu of a program (PRG) or a function (FUN)
Add ► POU method

Name	Name of the method, for example <i>meth_DoIt</i>
Return type	Standard type or structured type of return value. Example: <i>BOOL</i>

Implementation language	Example: Structured text (ST)
Access specifier	Controls access to data. • "PUBLIC" or not specified: Access is not restricted. • "PRIVATE": The access is limited to the program, the function block or the GVL. The object is marked as (<code>private</code>) in the POU or device tree. The declaration contains the keyword <code>PRIVATE</code>. • "PROTECTED": Access is restricted to the program, function block, or GVL with its derivations. The declaration contains the keyword <code>PROTECTED</code>. The object is marked as (<code>PROTECTED</code>) in the POU or device tree. • "INTERNAL": The access to the method is limited to the namespace (library). The object is marked as (<code>internal</code>) in the POU or device tree. The declaration contains the keyword <code>INTERNAL</code>.

Special methods for a function block

FB_Init	Declarations are automatically implicit. Explicit declaration is also possible. Contains initialization code for the function block, as is defined in the declaration part of the function block.
FB_Reinit	Explicit declaration required. After the instance of the function block is copied (as during an online change): calls and reinitializes the new instance module.
FB_exit	Explicit declaration required. Call for each instance of the function block before a new download or a reset or during an online change for all shifted or deleted instances.
POU properties and interface properties (PROPERTY)	Each consists of a <code>Set</code> and/or a <code>Get</code> accessor method.

Input assistance when generating function blocks that inherit properties



When you do object-oriented programming and want to use inheritance for blocks, you have the following support:

When you insert a method, action, etc. below an inherited block, the "Add Object" dialog box includes a combo box with a list of methods, actions, etc. used in the base block.

In this way, you can easily accept a method definition of the base and adapt it accordingly for the inherited method of the block.

Methods and attributes with the `PRIVATE` access modifier are not available in this selection because they should not be inherited.

When accepted into the inherited block, methods and attributes with the `PUBLIC` access modifier automatically have a blank access modifier field. (Functionally seen, it is the same.)

Also refer to

- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18 "Object-oriented programming" on page 145](#)

Also refer to

- [chapter ""Interface" object \(INTERFACE\)" on page 529](#)

- [chapter 9.18.4 "Method call" on page 150](#)
- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)
- [chapter ""Interface property" object" on page 534](#)
- [chapter "Attribute 'monitoring'" on page 394](#)
- [chapter "ST editor in online mode" on page 226](#)
- [chapter 13.1.2 "Using watch lists" on page 196](#)
- [chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427](#)
- [chapter 17.2.9 "Instance variables - VAR_INST" on page 287](#)

"Interface method" object

Symbol: 

This object is used for object-oriented programming.

The object **Interface method** is added to an interface via the command **Add**.

If a method is inserted underneath an interface, you can add and instance only variable declarations (input, output and input/output variables) in this method.

You can only add program code to the method if a function block "implements" the interface to which the method belongs. IndraLogic inserts the method below the function block.

Also refer to

- [chapter ""Interface" object \(INTERFACE\)" on page 529](#)
- [chapter ""Method" object " on page 532](#)
- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)

"Interface property" object

Symbol: 

Interface properties are an extension of the IEC 61131-3 standard and a tool for object-oriented programming. An interface property declares the accessor methods `Get` and `Set` (no implementation code). Therefore, a function block that implements an interface also inherits their interface properties.

An interface property can be added to an interface in the Project Explorer. Consequently, the interface is extended by the accessor methods `Get` and `Set`. The `Get` accessor is used for read access. The `Set` accessor is used for write access. An accessor that is not required can be deleted. To add an accessor, use the command **Project → Ad object → Interface property**. The **Add interface property** dialog opens.

Also refer to:

- [chapter ""Interface" object \(INTERFACE\)" on page 529](#)
- [chapter "Object 'Property'" on page 537](#)

Creating an Interface property

To create the object, open the context menu of the interface object, an interface property is to be added to in the Project Explorer. Use the **Add ► Interface property...** command to extend the interface by the accessor methods `Get` and `Set`.

The `Get` accessor is used for read access.

The `Set` accessor is used for write access.

Accessors that are not required can be deleted.

To add a deleted accessor, select the interface property and use the command

Add ► Interface Set access definition... for a Set accessor or

Add ► Interface Get access definition... for a Get accessor.

Use case Declaration and implementation of the interface property `Literal_A`:

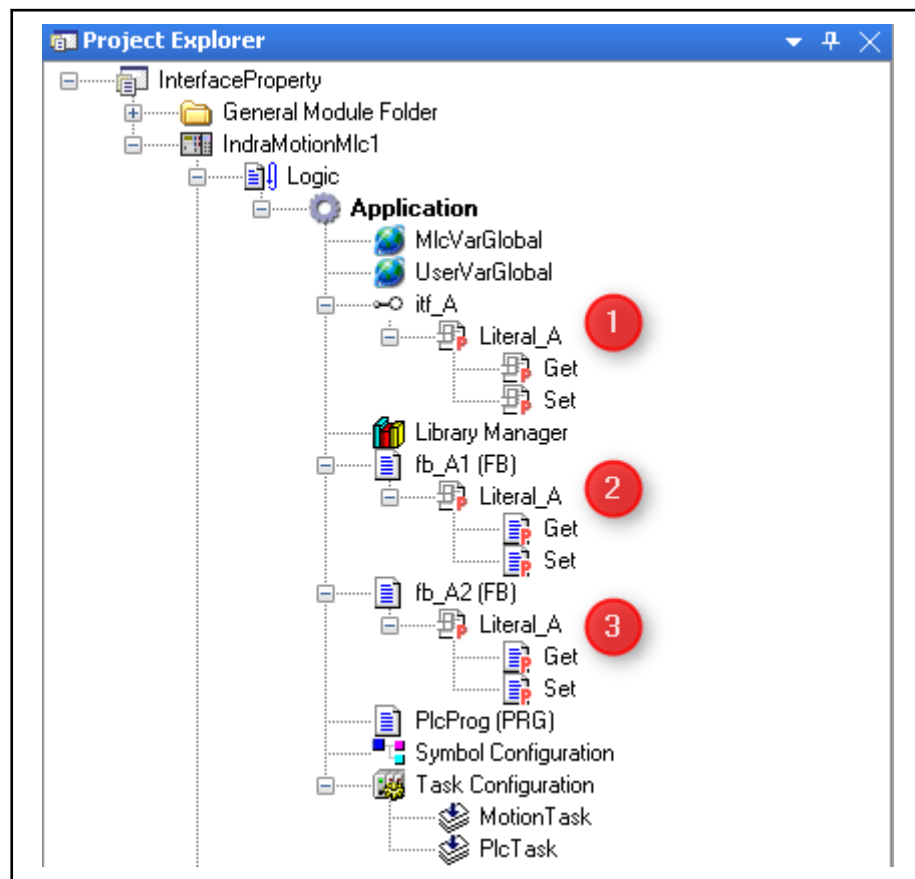


Fig. 18-26:

(1) The interface property "Literal_A" was added to the "itf_A" interface in the Project Explorer (in this example with return type `STRING`).

(2) & (3) The POU property "Literal_A" was added to the "fb_A1" and "fb_A2" function blocks (in this example with return type `STRING` and the implementation language "Structured text").

This interface "itf_A" has the property "Literal_A" with the accessor methods "Get" and "Set".

The function blocks "fb_A1" and "fb_A2" implement the interface "itf_A" and therefore inherit its interface property.

In the following the implementation for interface, interface properties, function blocks and program is listed:

Interface itf_A

```
INTERFACE itf_A
VAR
END_VAR
```

Interface property Literal_A

```
PROPERTY Literal_A : STRING
```

FB fb_A1

```

FUNCTION_BLOCK fb_A1 IMPLEMENTS itf_A
VAR
  str_1 : STRING;
  str_2 : STRING;
  iCnt : INT;
END_VAR

iCnt := iCnt + 1;
str_1 := 'Function block A1';

```

Accessor fb_A1.Literal_A.Get

```

VAR
END_VAR

Literal_A := CONCAT (str_1, ' and property.');
```

Accessor fb_A1.Literal_A.Set

```

VAR
END_VAR

str_2 := Literal_A;
```

FB fb_A2

```

FUNCTION_BLOCK fb_A2 IMPLEMENTS itf_A
VAR
  str_1 : STRING;
  str_2 : STRING;
  iCnt : INT;
END_VAR

iCnt := iCnt + 1;
str_1 := 'Function block A2';

```

Accessor fb_A2.Literal_A.Get

```

VAR
END_VAR

Literal_A := str_1;
```

Accessor fb_A2.Literal_A.Set

```

VAR
END_VAR

str_2 := Literal_A;
```

Program PLC_PRG

```

PROGRAM PLC_PRG
VAR
  iCnt : INT;
  my_1 : fb_A1;
  my_2 : fb_A2;
  strName_1 : STRING;
  strName_2 : STRING;
END_VAR

iCnt := iCnt + 1;
my_1();
my_2();
strName_1 := my_1.Literal_A;
strName_2 := my_2.Literal_A;
my_1.Literal_A := 'Hello 1';
my_2.Literal_A := 'World 2';

```

After the sample project has been compiled and transferred to the control, the following values are monitored at application runtime in PLC_PRG:

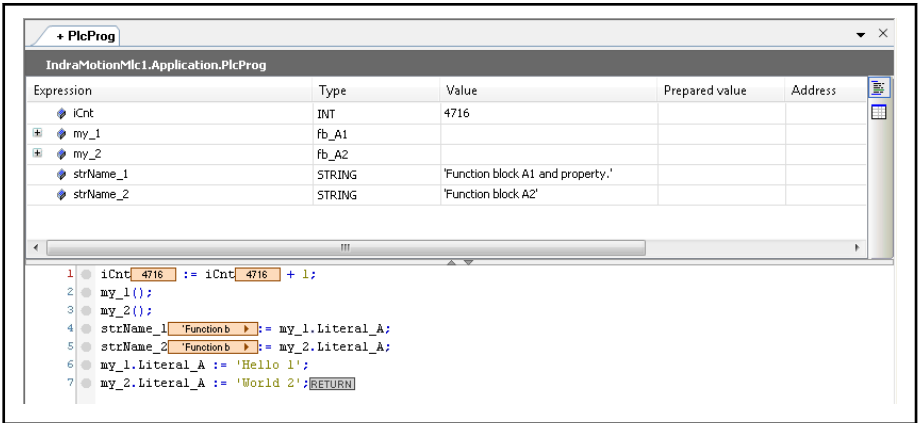


Fig. 18-27:

Object 'Property'

Symbol: 

Keyword: PROPERTY

Properties are an extension of the IEC 61131-3 standard and a tool for object-oriented programming. Properties are used for data encapsulation because they allow for external access to data and act as filters at the same time. For this purpose, a property provides the accessor methods “Get” and “Set” which allows for read and write access to the data of the instance below the property.

Add a property with accessor methods below a program or a function block in the Project Explorer. Open the context menu of the respective POU and select **Add ► POU property...**

The "Add property" dialog opens.

Also refer to

- chapter 17.10 "'FB_Init', 'FB_Reinitand 'FB_Exit' methods" on page 427
- chapter "'Interface property" object" on page 534

Dialog Add property

Function: Configures the property that is added by closing the dialog.

Call: POU context menu, command**Add ► POU property...**

Prerequisite: A program (PRG) or a function block (FUNCTION_BLOCK) is selected in the Project Explorer.

Name	Name (identifier) of the property
Return type	Standard type or structured type of return value. Example: BOOL

Implementation language	Example: Structured text (ST)
Access specifier	<i>Controls access to data.</i> • PUBLIC or not specified: Access is not restricted. • PRIVATE: Access is restricted to the program or the function block. The object is identified by (private) in the Project Explorer. The declaration contains the keyword PRIVATE. • PROTECTED: Access is restricted to the program or the function block with its derivations. The object is identified by (protected) in the Project Explorer. The declaration contains the keyword PROTECTED. • INTERNAL: Access is restricted to the namespace (library). The object is identified by (internal) in the Project Explorer. The declaration contains the keyword INTERNAL.

Editor 'Property'

You can program the data access in the editor. The code can contain additional local variables. However, it cannot contain any additional input variables or (as opposed to a function or method) output variables.

Example:

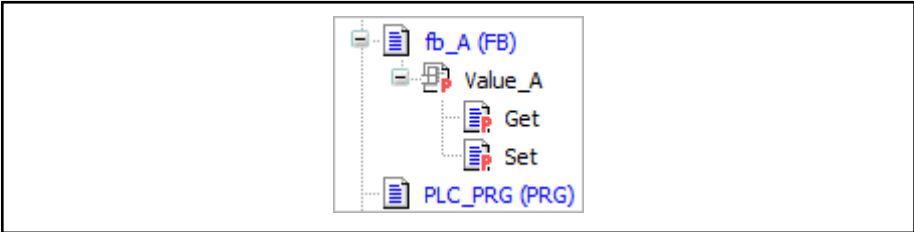


Fig. 18-28:
Function block fb_A

```
FUNCTION_BLOCK fb_A
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    iA : INT;
    iB : INT;
END_VAR

iA := iA + 1;
```

Property Value_A

```
PROPERTY PUBLIC Value_A : INT
```

Accessor method fb_A.Value_A.Get

```
VAR
END_VAR

Value_A := iA;
```

Accessor method fb_A.Value_A.Set

```
VAR
END_VAR

iB := Value_A;
```

Get and Set accessors

Upon calling the Set accessor, data is written to the property. It is used like an input parameter. When the Get accessor is called, the property is read. It is used like an output parameter. Access is restricted via access modifiers, and the objects are highlighted accordingly.

When a property is accessed as read only or write only, delete the accessor that is not required.

Explicitly add accessors by selecting a property and clicking on **Add ► POU Set access definition...** or **POU Get access definition....** A dialog opens, either Add Get accessor or Add Set accessor. There you can set the implementation language and the access.

Implementation language	Example: Structured text (ST)
Access specifier	<i>Controls access to data.</i> • PUBLIC or not specified: Access is not restricted. • PRIVATE: Access is restricted to the program or the function block. The object is identified by (<code>private</code>) in the Project Explorer. The declaration contains the keyword <code>PRIVATE</code>. • PROTECTED: Access is restricted to the program or the function block with its derivations. The object is identified by (<code>protected</code>) in the Project Explorer. The declaration contains the keyword <code>PROTECTED</code>. • INTERNAL: Access is restricted to the namespace (library). The object is identified by (<code>internal</code>) in the Project Explorer. The declaration contains the keyword <code>INTERNAL</code>.

Monitoring properties in online mode

The following pragmas are provided for the monitoring of properties in online mode. You insert them at the top position of the property definition:

- `{attribute "monitoring:= "variable"}:` Each time the property is accessed, IndraLogic saves the momentary value in a variable and displays the value of this variable. This value can become outdated if no further access to the property takes place in the code.
- `{attribute "monitoring" := "call"}:` Each time the value is displayed, IndraLogic calls the code of the Get accessor. If this code contains a side effect, then the monitoring executes the side effect!

Monitor a property with the help of the following functions.

- Inline monitoring

Requirement: The option "Enable inline monitoring" is selected in the option settings of the text editor.

Call: IndraWorks menu **Tools ► Options ► IndraLogic 2G ► Text editor ► Monitoring**

- Monitoring list

Also refer to:

- [chapter 9.18.4 "Method call" on page 150](#)
- [chapter "Attribute 'monitoring'" on page 394](#)

Input assistance when generating function blocks that inherit properties



When you do object-oriented programming and want to use inheritance for blocks, you have the following support:

When you insert a method, action, etc. below an inherited block, the "Add Object" dialog box includes a combo box with a list of methods, actions, etc. used in the base block.

In this way, you can easily accept a method definition of the base and adapt it accordingly for the inherited method of the block.

Methods and attributes with the PRIVATE access modifier are not available in this selection because they should not be inherited.

When accepted into the inherited block, methods and attributes with the PUBLIC access modifier automatically have a blank access modifier field. (Functionally seen, it is the same.)

Also refer to

- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18 "Object-oriented programming" on page 145](#)

Object "Action"

Symbol:

Keyword: ACTION

Implement more program code in an action. This program code can be implemented in a language different from the basic implementation. This basic implementation is a function block or a program where you inserted the action.

An action does not have its own declaration and it works with the data from the basic implementation. This means that the action uses the input and output variables and the local variables from its basic implementation.

Add an **Action** to a function block or program by clicking **Add ► Action**.

Adding action

Name	Action name
Implementation language	Drop-down list of implementation language

Input assistance when generating function blocks that inherit properties



When you do object-oriented programming and want to use inheritance for blocks, you have the following support:

When you insert a method, action, etc. below an inherited block, the "Add Object" dialog box includes a combo box with a list of methods, actions, etc. used in the base block.

In this way, you can easily accept a method definition of the base and adapt it accordingly for the inherited method of the block.

Methods and attributes with the PRIVATE access modifier are not available in this selection because they should not be inherited.

When accepted into the inherited block, methods and attributes with the PUBLIC access modifier automatically have a blank access modifier field. (Functionally seen, it is the same.)

Also refer to

- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18 "Object-oriented programming" on page 145](#)

Calling an action

Syntax:

<program>.<action> or <FB-instance>.<action>

To call an action from only within the basic implementation, you only have to provide the action name.

Examples

Calling a **Reset** action from another POU. The call is not executed from the basic implementation.

Declaration:

```
PROGRAM PlcProg
VAR
    Inst : Counter;
END_VAR
```

Calling a **Reset** action from an IL-POU

```
CAL Inst.Reset(In := FALSE)
LD Inst.Out
ST ERG
```

Calling a **Reset** action from an ST-POU:

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

Calling a **Reset** action from an FBD-POU:

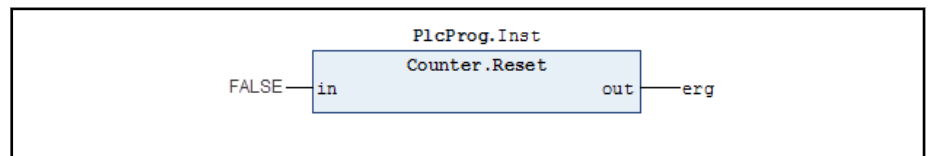


Fig. 18-29:



Actions are used frequently in the SFC implementation language.

Also refer to

- [chapter "SFC element "Action"" on page 247](#)

'Transition' object

Symbol:

Keyword: TRANSITION

This object can be used as a transition element in a function block implemented in SFC.

Also refer to

- [chapter "SFC editor in online mode" on page 236](#)

Input assistance when generating function blocks that inherit properties



When you do object-oriented programming and want to use inheritance for blocks, you have the following support:

When you insert a method, action, etc. below an inherited block, the "Add Object" dialog box includes a combo box with a list of methods, actions, etc. used in the base block.

In this way, you can easily accept a method definition of the base and adapt it accordingly for the inherited method of the block.

Methods and attributes with the PRIVATE access modifier are not available in this selection because they should not be inherited.

When accepted into the inherited block, methods and attributes with the PUBLIC access modifier automatically have a blank access modifier field. (Functionally seen, it is the same.)

Also refer to

- [chapter 9.18.1 "Extension of a function block" on page 145](#)
- [chapter 9.18 "Object-oriented programming" on page 145](#)

18.1.18 Monitoring the PLC program of the application at runtime

To monitor PLC programs at runtime for exceedances of arrays and sub-range types, the validity or pointer accesses and division by zero, IndraLogic 2G provides the following options:

- [chapter "POU templates" on page 543](#)

To select specific check functions, go to the context menu of the application node and select **Add ► POU templates ► IndraMotion Check-Functions**. The individual functions are implicitly integrated before a possible incorrect access to the program code. If there is an incorrect access, an entry is generated in the control logbook and processing the program code is stopped with exception by default. In this case, check the call list (**Debug ► Call list**) and change the incorrect code.

When integrating or deleting a check function, login is only possible with download. Furthermore, additional compilation outputs "Checkxxxx: Please implement your own error reaction below" are generated to inform the user that the implemented function code has to be adjusted to the machine program if required. Depending on the application, it might not be permitted to switch the machine to stop at runtime.

An alternative is exemplarily specified in each check function, adjusting the incorrect operation to ensure that this operation does not cause a machine standstill and that an entry is made in the error and diagnostics memory of the control. A division by zero is thus substituted by a division by one for example. If required, implement more error reactions, as the application would keep running with an incorrectly calculated value.



1. When implementing the error reaction in the check function, adjust to the requested machine behavior!
If an error occurs, the application is stopped by default.
2. When integrating or deleting the function, a subsequent login is only possible as download. An online change is not possible anymore.
3. By integrating a check function, the PLC program runtime can be significantly increased.
The functions should be removed after recovering all errors to reduce the runtime again.

- The **RIL_CheckRTV.library**

If this library is added to the application by the library manager, the compiler automatically implements its functions at the problematic positions of the program code.

As soon as the check function detects an incorrect access or an incorrect operation, an exception is specifically triggered to switch the application to the safe state. It is not possible to change this behavior.

In this case, check the call list (**Debug ▶ Call List**) and change the incorrect code.

If the application may not be stopped while the machine is running, this library may not be used!



1. The application is always stopped when an error occurs.
2. When integrating or deleting the function, a subsequent login is only possible as download. An online change is not possible anymore.
3. By integrating this library, the PLC program runtime can be **significantly** prolonged.
The functions should be removed after recovering all errors to reduce the runtime again.

POU templates

PLC monitoring functions

Under the menu item **Tools ▶ Template database** or in the context menu of ... ▶ **Logic ▶ Application ▶ Add ▶ POU templates ▶ IndraMotion ... Check-Functions**, several functions of the RIL_CheckRTV.library were extracted to monitor the PLC program code. The functions listed can be specifically implemented to monitor the PLC program code.

The individual functions are implicitly integrated before a possible incorrect access to the program code. If there is an incorrect access, an entry is generated into the control logbook and processing the program code is stopped with exception. In this case, check the call list (**Debug ▶ Call list**) and change the incorrect code.



1. By integrating the function(s), the PLC program runtime can be significantly prolonged.
2. After troubleshooting in the commissioning, remove the function(s).



To maintain the functionality of monitoring functions, the declaration part may not be modified.



If one of the objects was added to the application, an RIL_CheckRTV.library added by the library manager has no effect. Only the individually selected function blocks are applied during compilation.

Brief description of the functions:

Checkbounds	The "CheckBounds" function checks the correct access to array elements in the IEC61131.3 program. The task of the monitoring function is an adequate handling of array bound violations. The function is called implicitly as soon as values are assigned to a variable of the type "array".
CheckDivInt	The "CheckDivDInt" function is used to monitor the value of a divisor (type 32 bits), i.e., to avoid a division by 0. After implementing the "CheckDivDInt" function into the IndraLogic project, the "CheckDivDInt" function is automatically called before each division with 32-bit data types in the respective code.
CheckDivLint	The "CheckDivLInt" function is used to monitor the value of a divisor (type 64 bits), i.e., to avoid a division by 0. After implementing the "CheckDivLInt" function into the IndraLogic project, the "CheckDivLInt" function is automatically called before each division with 64-bit data types in the respective code.
CheckDivLReal	The "CheckDivLReal" function is used to monitor the value of a divisor of type LREAL, i.e., to avoid a division by 0.0. After integrating the "CheckDivLReal" function into the IndraLogic project, the "CheckDivLReal" function is automatically called before each division with LREAL variables in the respective code.
CheckDivReal	The "CheckDivReal" function is used to monitor the value of a divisor of type LREAL, i.e., to avoid a division by 0.0. After integrating the "CheckDivReal" function into the IndraLogic project, the "CheckDivReal" function is automatically called before each division with REAL variables in the respective code.
CheckPointer	After integrating the "CheckPointer" function, the function is used to check read or write access using a pointer.
CheckRangeSigned	The "CheckRangeSigned " function is used to monitor the range limits for signed subrange types at runtime. After implementing the "CheckRangeSigned" function into the IndraLogic project, the "CheckRangeSigned" function is automatically called before each access to signed, value-restricted variables in the respective code.
CheckRangeUnsigned	The "CheckRangeUnsigned " function is used to monitor the range limits for unsigned subrange types at runtime. After implementing the "CheckRangeUnsigned" function into the IndraLogic project, the "CheckRangeUnsigned" function is automatically called before each access to unsigned, value-restricted variables in the respective code.

18.1.19 "Project settings" object

Symbol 

Function: This object contains the project configuration.

Call: Double-click on the object in the Project Explorer, "General module folder"

IndraLogic saves the project settings directly in the project. If, for example, you transfer a project to another system, the **Project settings** object is also transferred with it without a project archive being required.

The project settings are applied project-wide and provide different setting options for various categories such as **SFC**, **Compiler options / compiler warnings**, **StaticAnalysisLight** and **Visualization**.

Also refer to

- [chapter ""Project settings" - "SFC" dialog" on page 714](#)
- [chapter ""Project settings" - "Compiler options" dialog" on page 715](#)
- [chapter ""Project settings" - "Compiler warnings" dialog" on page 716](#)
- [chapter ""Project settings" dialog - "Static analysis light"" on page 716](#)
- [chapter ""Project settings" dialog - "Visualization"" on page 719](#)

18.1.20 "Library info" object

Symbol: 

Function: This object contains the properties, meta-information, and project information. With this, you can check the authorship and integrity of the project.

Call: Double-click on the object in the Project Explorer, "General module folder"

Prerequisite: IndraLogic automatically creates the object in the library projects or via the context menu item **Add ► Library info....**

IndraLogic directly saves the library info to the project.

For example, if you transfer a project to another system, the **Library info** object is also transferred. No project archive is required.

"File" tab The tab displays the properties of the project file and their attributes. These attributes cannot be edited. They correspond to the file properties of the Windows Explorer. The IndraLogic version used to save the **library info** is specified.

"Summary" tab This tab contains general information and meta-information on the project file. IndraLogic uses this information to create keys in the **Properties** tab. For example, if the name `Company_A` is specified in **Company**, the `Company` key with the value `Company_A` is provided in the **Properties** tab.



If you save your project as a library project, then you should pay attention to the guidelines for library developers "Library Development Summary".

For a library project, a Company , a Title , and a Version must be specified to install the library.	
Company	Name of the company (example: <code>Company_A</code>).
Title	Title of the project, e.g. <code>Automation_A</code>

Version	Project version, e.g. 0.0.0.1
Released	 : Activates protection against modification. Consequence: If you edit the project now, then a dialog prompt opens to confirm whether you really want to change the project. If you reply to this prompt one time by clicking Yes , then no additional prompts appear for more editing actions.
Categories	Categories of the library project, according to which you can sort in the Library Repository dialog. If no category is specified, then the category Other is assigned to the library. The categories originate from one or more external description files in XML format. However, they can also originate from a library project that has already been created. Prerequisite: The project is a library project.
	The Library categories dialog opens where you can add library categories.
Standard namespace	If you do not define a standard namespace here, then the name of the library file is applied automatically as the namespace.
Author	Author of the project, e.g. Arnold Best
Description	Example: For internal use only



The meta data **Released** is applied to library projects as well as standard projects!

If a standard project is released for example, it is not permitted to store a new list with **persistent variables**. This is only possible after deleting the metadata.

Also refer to: [chapter 9.16.5 "Saving the values of a persistent variable list in a recipe" on page 144.](#)

List of categories	List of the categories that are assigned to the library project. They can originate from several sources. After you specify all desired categories, click OK to confirm.
Button Add	The From description file and From other library commands appear.
Button Remove	IndraLogic removes the selected category.
From description file command	The Select description file dialog opens where you select a description file (*.lib-cat.xml). The file contains command categories. When you click Open , IndraLogic applies the categories.
From another library command	The Select library dialog opens, where you select a library with command categories to be accepted. When you click Open , IndraLogic applies the categories.
Button OK	IndraLogic provides the categories as library information and displays it in the Library categories field.

Tab. 18-6: **Library categories dialog**

Also refer to

- [chapter 19.1 "Guidelines to create libraries" on page 743](#)
- [chapter ""Library repository" command" on page 636](#)

"Properties" tab

In this tab, define keys that can be controlled externally from user-specific programs.



If you have opened a library project, please observe the description of the relevant keys in the guidelines for library developers: "Library development summary".

If you have opened a symbol library as a project, then the key `VisuSymbolLibrary = TRUE` must be defined. It identifies the library as symbol library.

Key	Name of the key. Specify any string of text for the new key, or select an existing key from the Properties table.
Type	Data type of the key. Possible types: (Text , Date , Number , Boolean , Version)
Value	Value of the key in permitted format: • "Text": Any character string • "Date": Example: Friday, January 1, 2016 00:00:00. Minimum entry for the date: 1.1 • "Number": Integer in Integer32 format with or without a sign, example: -32500. • "Bool": True or False, capitalization irrelevant • "Version": Examples: 1.1, 1.0.1.0, maximum four figures.
Add	Adds the new defined key to the Properties table.
Modify	Saves the change made for the key selected in the "Properties" table.
Remove	Removes the key selected in the "Properties" table.
Properties	List of the properties that are defined as keys. IndraLogic creates keys automatically for the information in the Summary tab. Click a key to edit it in the input fields above the list.

Automatically generate library info POU's	 : IndraLogic creates programming objects of the <code>FUNCTION</code> type in the "General module folder", subfolder "Library info" and thus allows to access the library properties in the application. • <code>GetLibVersion</code> (version number as character string) • <code>GetLibVersionNumber</code> (version number as numerical value) • <code>IsLibReleased</code> (TRUE/FALSE)
Automatically generate project information POU's	 : IndraLogic creates POU objects of the <code>FUNCTION</code> type in the "General module folder", subfolder "Project information" and thus allows to access the project properties in the application. The following function blocks are created for the properties Company , Title and Version . • <code>GetCompany</code> • <code>GetTitle</code> • <code>GetVersion</code> The following function blocks are available for user-defined properties: • <code>GetBooleanProperty</code>: <code>BOOL</code> (TRUE/FALSE) • <code>GetNumberProperty</code>: <code>DINT</code> (Numerical value) • <code>GetTextProperty</code>: <code>WSTRING</code> (character string) • <code>GetTextProperty2</code>: <code>POINTER TO WSTRING</code> (without length limitation) • <code>GetVersionProperty</code>: <code>VERSION</code> (version number as character string)

Also refer to

- [chapter 19.1 "Guidelines to create libraries" on page 743](#)
- [chapter 20.1.3 "Using the symbol library in the visualization" on page 763](#)
- [chapter ""Properties" - "Image Pool" dialog" on page 712](#)

"Statistics" tab	This dialog provides statistical information on the number of objects of the individual type or use in the project.
"Signing" tab	In this dialog, sign a library project with a vendor-specific, unique key. Prerequisite: This key is available as a "private key file" *.libpk with an associated token.

Enable signing	 : IndraLogic signs the library project with a single-use, manufacturer-specific key.
Private key file	Location of the private key file *.libpk (example: D:\for lib developers only\mycomp_libkey.libpk).
Public key token	Example: 427A5701DA3CF3CF Prerequisite: A private key file is specified, and IndraLogic has read and entered the token
Generate private key file...	IndraLogic creates a new private key file.

Also refer to

- ["Signing library project" on page 12](#)

Also refer to

- [chapter 2.5.1 "Retrieving and editing library information" on page 11](#)
- [chapter 2.5.2 "Applying project settings" on page 13](#)

18.1.21 "Recipe manager" object

"Storage" tab The recipe manager provides functions for maintaining user-defined variable lists, known as recipe definitions.

Storage type	Textual: IndraLogic saves the recipe in a readable Format with the configured columns and delimiters. Binary: IndraLogic saves the recipe in a non-readable binary format. This format requires less storage space. Note: You can read binary recipes again only if you have not changed the variable lists.
File path	Relative path on the runtime system. The path has to end with a "\". Example <code>MyRecipes\</code>. This path is created on the target system in the directory for the runtime files. IndraLogic saves a file in this directory for each recipe when downloading to the control. The prerequisite is that you select the option Recipe Management in the plc. The files are loaded to the recipe manager each time the application is restarted.
File extension	File extension for the recipe file The resulting default name for recipe files is in the form <code><Recipe>.<RecipeDefinition>.<FileExtension></code>
Separator	Delimiters between the individual values in the saved file: Tab, ";", ",", space, ":", " "
Available columns Selected columns	Defines which information is saved in which order in the recipe file
Save as default	IndraLogic uses the default setting across the project for all other recipe managers.

"General" tab

Recipe management in the PLC	 Must be selected for the user program or visualization elements to load recipes in runtime mode. If you transfer recipes to the control exclusively via the IndraLogic program interface, then you can clear this option.
Save recipe changes to recipe files automatically	 We recommend this option because it helps the recipe manager operate "normally". The recipe files are updated automatically at runtime whenever a recipe is changed. This option is active only if you have selected the option Recipe Management in the plc .
Download only for exact match of the variable list	Recipe files are downloaded only if the file contains all variables from the variable list of the recipe definition of the application. The variable list in the file must be in the same order. Additional entries at the end are ignored. Otherwise, the error status <code>ERR_RECIPE_MISMATCH</code> is set (<code>RecipeManCommands.GetLastError</code>).
Download variables with matching names	Only variables with matching variables names are loaded from the recipe file. No error status is set when the variable list in the file is different from the list in the recipe definition of the application. In this way, recipe files can also be downloaded if variables in the file or in the recipe definition have been deleted.

Actions	Recipes defined in the project	Recipes defined at runtime
Online reset warm Online reset cold Download	The recipes of all recipe definitions are loaded with the values from the current projects.	Dynamically generated recipes remain unchanged.
Online reset origin	The application is removed from the PLC. If a new download is done afterwards, then the recipes are restored as for an online reset warm	
Shutdown and restart of the PLC	After restarting, the recipes are reloaded from the automatically generated files. Thus, the same state as before shutting down, is restored.	
Online change	The recipe values remain unchanged. In runtime mode, a recipe can be changed only via the function block command <code>RecipeManCommands</code> .	
Stop	The recipes remain unchanged when the PLC is stopped or started.	

Tab. 18-7: Recipes in online mode if you have selected the option "Save recipe changes to recipe files automatically"

Actions	Recipes defined in the project	Recipes defined at runtime
Online reset warm Online reset cold Download	The recipes of all recipe definitions are loaded with the values from the current projects. However, these are only set in the memory. To save recipes to a file, you must execute the command Save Recipe explicitly.	Dynamically generated recipes are lost.
Online reset origin	The application is removed from the PLC. When a download is performed afterwards, the recipes are restored.	Dynamically generated recipes are lost.
Shutdown and restart of the PLC	After restarting, the recipes are reloaded from the automatically generated files. Thus, the same state as before shutting down, is restored.	
Online change	The recipe values remain unchanged. In runtime mode, a recipe can be changed only via the function block command <code>RecipeManCommands</code> .	
Stop	The recipes remain unchanged when the PLC is stopped or started.	

Tab. 18-8: Recipes in online mode if you have NOT selected the option "Save recipe changes to recipe files automatically"

One of the specified "memory types" (textual or binary) as well as the storage location ("file path") and the "file extension" of the recipe to be saved, have to be specified.

"Binary" storage type



For the memory type "binary", data is saved small in the recipe to be saved without name.

The length in byte corresponding to the data type is reserved for each data.

As a consequence, the recipe definition has to be the same it was while saving when loading the recipe from a binary file.

The following errors can result when "loading the recipe":

- Interchanging two variables of the same type in the recipe definition is not detected and results in incorrect variable values.
- A change in variable type (also if the variable length is the same) is detected. "The file format is invalid" is output as error.

- Additional or less variables are detected in the recipe definition. Restoration is not possible anymore.

GVL PlcProg Recipe Manager Recipes_01							
Variable	Type	Name	Comme...	Minimal Value	Maximal Value	Current Value	dd
GVL.dword1	DWORD					16#87654321	2271560481
GVL.bool1	BOOL					TRUE	TRUE
GVL.dint1	DINT					16#00000021	33
GVL.bool2	BOOL					FALSE	FALSE
GVL.word1	WORD					16#4321	17185
GVL.bool3	BOOL					TRUE	TRUE
GVL.string1	STRING					'Ein Text'	'Ein Text'
GVL.bool4	BOOL					TRUE	TRUE
GVL.real1	REAL					3.14	3.14
GVL.bool5	BOOL					TRUE	TRUE

Fig. 18-30: Recipe, online

00000000:	04 00 05 00	21 43 65 87	01 00 00 00	01 04 00 04		?	Cell.	?	?
00000010:	00 21 00 00	00 01 00 00	00 00 02 00	03 00 21 43		?	?	?	?
00000020:	01 00 00 00	01 09 00 00	00 45 69 6E	20 54 65 78		.	..	Ein Tex	
00000030:	74 00 01 00	00 00 01 04	00 06 00 C3	F5 48 40 01		t	.	-	ÄÖHö.
00000040:	00 00 00 01								

Fig. 18-31: Binarily saved recipe

Framed: Length in bytes of the subsequent date followed by the type ID;

Without frame: Date (current value)

"Textual" memory type

If saved as text, the columns to be saved are separated by a **separator**. One of the six selected options is selected, see above.

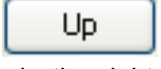
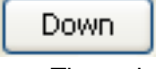
The textual variant can be edited externally and also transferred to other applications. However, it requires more storage location than the binary ones.

All selected columns of the recipe definitions, i.e. those to be saved, are listed with their respective column header in the lower section of the dialog ("Available columns").

The selected columns of the recipe definition, i.e. those to be saved, are located in the right section ("Selected columns"). There is at least the "Current value" column. It cannot be selected.

The other columns can be moved to the right  or to the left  by selecting the corresponding entry.

Press  or  to move all entries simultaneously.

Use the buttons  and  to change the order of the selected columns in the right window. The columns are then arranged accordingly in the memory file.

GVL PlcProg Recipe Manager Recipes_01							
Variable	Type	Name	Comme...	Minimal Value	Maximal Value	Current Value	dd
GVL.dword1	DWORD					16#87654321	2271560481
GVL.bool1	BOOL					TRUE	TRUE
GVL.dint1	DINT					16#00000021	33
GVL.bool2	BOOL					FALSE	FALSE
GVL.word1	WORD					16#4321	17185
GVL.bool3	BOOL					TRUE	TRUE
GVL.string1	STRING					'Ein Text'	'Ein Text'
GVL.bool4	BOOL					TRUE	TRUE
GVL.real1	REAL					3.14	3.14
GVL.bool5	BOOL					TRUE	TRUE

Fig. 18-32: Recipe, online

```
GVL.dword1:=2271560481
GVL.bool1:=TRUE
GVL.dint1:=33
GVL.bool2:=FALSE
GVL.word1:=17185
GVL.bool3:=TRUE
GVL.string1:='Ein Text'
GVL.bool4:=TRUE
GVL.real1:=3.14
GVL.bool5:=TRUE
```

Fig. 18-33: Textually saved recipe

Also refer to

- chapter 13.2 "Changing values with recipes" on page 198

18.1.22 "Recipe definition" object

In the recipe definition (1), you define different value sets for the variables, which are termed recipes (2).

You can toggle the display of the recipe definition between the flat list view (3) and the structured view (4). In the structured view, IndraLogic groups variables according to structure.

RecipeDef1 x RecipeDef2									
Variable	Type	Name	Comment	Min...	Max...	Current...	R1	R2	
PLC_PRG.iColorG	INT	Green501	green Color	0	100		10	15	3
PLC_PRG.iColorR	INT	Red001	red Color	0	100		18	30	4
PLC_PRG.iColorY	INT	Yellow001	yellow color	0	100		85	55	
PLC_PRG.sVisutext	STRING						'Product A'	'Product B'	
PLC_PRG.bSpecialColor	BOOL						TRUE	FALSE	

Fig. 18-34:

Data type	Entered automatically
Name	Optional
Minimum value Maximum value	If the variable value is less than the Minimal Value or greater than the Maximal Value , then IndraLogic sets the value to the Minimal Value or Maximal Value .
Comment	Additional information, for example the value of the unit.
Current value	Current variable value; not shown in online mode.

Also refer to

- chapter 13.2 "Changing values with recipes" on page 198

18.1.23 “Text list” object

Symbol: 

This object is used to create, manage and translate texts. It contains a table of texts into which new text can be inserted.

A text written here can be selected in a visualization in an element in the **Dynamic texts** properties. The visualization at runtime dynamically outputs this text in the language selected.

ID	Unique text ID
Default	Source text as character string, e.g. Information A: %i options. Use the shortcut [Ctrl] + [Enter] to insert a line break. Double-click into the field to edit the text.
The table includes any number of added language columns. A language column is named with a country code specified when creating the column with the Add language command.	
<LanguageCode>	Name of the language as country code (e.g., en-US). This column contains the translation of the text written under Standard . If the country code is selected as language in the visualization manager, the visualization outputs the translated text at runtime. If there is no translation, IndraLogic uses the text under Standard . A visualization at runtime can also execute a language switching requested by a user.
Empty line	Edit the line to insert own text.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

18.1.24 "Symbol configuration" object

Function: You can use the symbol configuration for creating symbol descriptions for project variables.

The "Symbol configuration" object is normally created together with the application object.

If the "Symbol configuration" is missing, insert it below the application via the context menu item **Add**.

Double-click on the **Symbol configuration** object to open the symbol configuration editor.

"Add symbol configuration" dialog

Function: This dialog box is used for defining presets for a "Symbol Configuration" object.

Call: Via the context menu of the "Application" object.

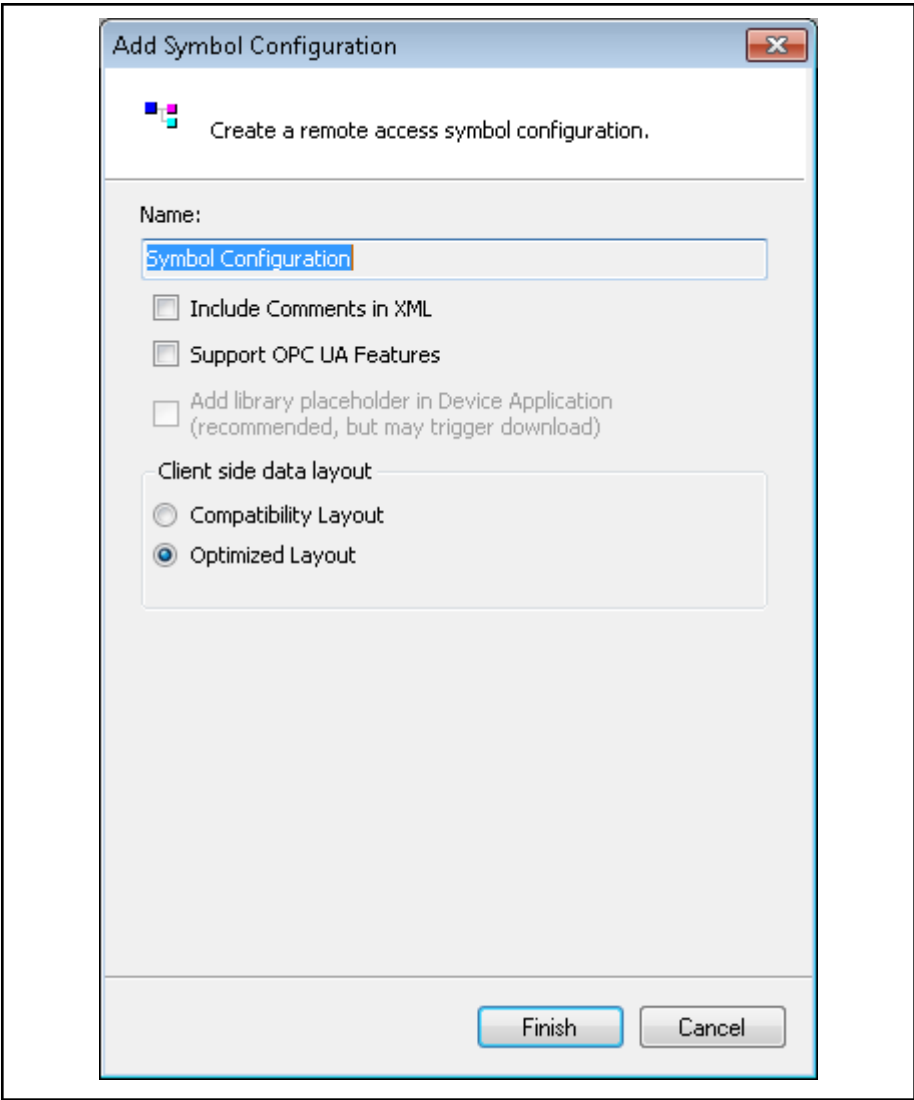


Fig. 18-35: Dialog: Add symbol configuration

Dialog elements

Name	Symbol configuration name
Include comments in XML	IndraLogic exports the symbol file with the comments assigned to the variables.

Support OPC UA functionalities	Note: Availability and editability of this option depend on the device. : When downloading the symbol configuration, additional information is also downloaded to the control required to operate the OPC UA server: • Basic types of derived function blocks • Contents of attributes assigned via compiler pragmas • Variable type (INPUT / OUTPUT / IN_OUT)
Client side data layout For detailed information and examples of layout options, refer to the next section "Symbol configuration editor".	
Compatibility layout	This setting is used for the compatibility of old projects. The data layout created for the client is adjusted as much as possible to the layout created internally by the compiler.
Optimized layout	From IndraWorks 14V12 (compiler version 3.5.7.0) Recommended for new projects. Calculates the output layout in optimized form, detached from the internal compiler layout. Does not generate any gaps for unpublished elements and strictly fulfills the requirements for data type alignment.

See:

- [chapter 10.2 "Symbol configuration" on page 156](#)

Symbol configuration editor

The editor includes a table with the selected variables, a menu bar for editing, a field with supporting information, and buttons for required actions.

Menu bar

View	<i>Use this button to enable and disabled the following variables for use in the configuration editor:</i> • Unconfigured from project: Variables that have not been added to the symbol configuration, but are provided in the project. • Unconfigured from Libraries: Variables that have not been added to the symbol configuration, but are provided in the included libraries. • Symbols exported via attribute: This filter causes IndraLogic to also list the variables that have already been marked for export in the symbol file by means of the <code>{attribute "symbol" := "read"}</code> pragma. The symbols are displayed in gray. The Attribute column shows which access right is set by the pragma.
Create	Building the project. Requirement for current preparation of variables in the configuration editor.

Settings

- **Support OPC-UA functionalities** (device-dependent): : When downloading the symbol configuration, additional data is loaded to the control required to operate the OPC UA server. Currently, it includes the following information:
 - Basic types of derived function blocks
 - Contents of attributes assigned via compiler pragmas
 - Variable type (VAR_INPUT / VAR_OUTPUT / VAR_IN_OUT)
- **Include comments in XML:** IndraLogic exports the symbol file with the comments assigned to the variables.
- **Include node flags in XML:** : The namespace node flags provide additional information about the origin of a node in the namespace. The node flags always in the symbol table when OPC UA is activated. However, its inclusion in the XML file can be deactivated as some defective parsers have problems with it.
- **Configure comments and attributes:** In the "Comments and attributes" dialog, you configure the contents in the symbol configuration and the XML file.
- **Configuring synchronization with IEC tasks:** Opens the dialog "Properties - <PLC name>" , "Options" tab. This setting allows for symbolic clients (e.g. visualizations or database links based on the PLC handler) to have consistent read/write access synchronized with the IEC tasks. For a detailed description of this setting, refer to section "Setting: Configuring synchronization with IEC tasks below. Setting: Configure synchronization with IEC tasks"

Note:

Variable access which is synchronous with the IEC tasks can increase the jitter for all IEC applications on this device.

Synchronized consistent access can interrupt the real-time capability.

- Selection for the type of data layout that should be created for the client of the symbol configuration:

Note: Refer to the section "Example of data layout types" at the end of this help page. [chapter 18.1.24 "Symbol configuration" object](#) on page 553

- **Compatibility layout:** This setting is used for the compatibility of old projects. The data layout created for the client is adjusted as much as possible to the layout created internally by the compiler.

Due to the configuration possibilities of the symbol configuration which have grown over time, problematic offsets can still occur.

Possible causes:

- Memory gaps by internal pointers or references in blocks and by structure members that are not released in the symbol configuration
- Depending on the data type (`__XINT/ __XWORD` etc.) also different memory leaks in case of 32 and 64 bit systems
- Accidentally incorrect alignment by using the `pack_mode` or `relative_offset` attributes.
- **Optimized layout** (recommended for new projects): Calculates the output layout in optimized form, detached from the internal compiler layout. Does not generate any gaps for unpublished elements and strictly fulfills the requirements for data type alignment. Minimum requirement: Compiler version 3.5.7.0. from IndraWorks 14V12

- **Activate direct I/O access:** Warning: This feature is potentially dangerous and **not intended for operation in production**.

Activate only for error checking and tests, or when commissioning the machinery (for example, for checking cables connections).

: In the symbol configuration, you can also use access to direct I/O addresses that correspond to IEC syntax (for example, `"%IX0.0"`). Access to input addresses (I) is read-only*. Access to output addresses (Q) and memory addresses (M) can be read-write.

*Info: In simulation mode, write access to symbols is also possible for input addresses.

Download	If you use a device that supports its own application file for the symbol configuration, then this button is also available in the toolbar. If the symbol configuration is changed in online mode, the new <ApplicationName>._Symbols file can be loaded immediately to the control. Also refer to: chapter 10.2 "Symbol configuration" on page 156
Tools	"Save xml schema file": The command opens the standard dialog for saving a file in the file system. Thus, the XSDSchema of the symbol file can for example be provided for the use in external programs

Symbol table

Access rights	You can change the access rights for a symbol by clicking the symbol in the Access Rights column. <i>Symbols for access rights</i> : Read-only : write only : Read and write : execute These rights allow for executing access to functions, function blocks, methods, and programs. Requirements for assignment: The device provides the options "Support calls of functions, FBs, methods and programs" and "Support OPC UA features". Both options are activated in the "Settings".
Maximum	Maximum access rights to be specified.
Type	Alias data types are also displayed in the PLC programming system V3.5 SP6 and later. Example: MY_INT : INT for a variable declared with the data type MY_INT (type INT).
Member variables	Variables of a structured data type can also be added by selecting a checkbox for symbol configuration in the Symbols column. This causes IndraLogic to export all member variable symbols to the symbol file. However, in the Members column, click on the button  to select specific structural components. Note: This selection applies to all instances of this data type for which symbols are exported. If no member of a structured type is selected, an asterisk is displayed in the checkboxes of the members to show that all exportable members of the type are exported.

Setting: Configuring synchronization with IEC tasks

For the synchronous consistent access, it is first waited in the runtime system when processing a read or write request from the symbolic client until a point in time is found at which no IEC tasks is executed. When this gap is detected, restarting the IEC tasks is prevented until all values of the variable list have been copied. Subsequently, IEC tasks are scheduled again. The synchronized consistent access can cause a delayed start of IEC tasks. This is shown by the increased jitter. As all applications in the runtime system are managed by a common scheduler, this potential impairment of the real-time behavior affects all applications on the device. Thus, all application of the device are affected, irrespective of whether they contain a symbol configuration or whether they were loaded from one or multiple IndraLogic projects to the control. Thus, the runtime system only allows the synchronized consistent access if all applications loaded to the control when accessed, allow this access.



For the setting, go to the editor of the symbol configuration "Settings".



After changing the setting, all applications on the device have to be loaded again via download/online change and all boot applications have to be refreshed.

When is a synchronized consistent access required?

Generally, consistent values are not required for visualized values, as it is mostly not relevant for changing values from which IEC task cycle they originate. Not in case of rarely changing values. There are mostly also no strict consistency requirements while writing, as the machine has to be normally in a kind of stand-by mode when writing the recipes for example. In this mode, the values written as recipes cannot be accessed directly.

However, consistent values are especially required for database connections to save production data. For clocked machines, these values have to be synchronous to the production cycle (per produced product one set of values) and not consistent with regard to one or multiple IEC tasks. With reference to the machine clocking, the consistency must be already ensured by the IEC application. For this purpose, the values that arise during a production cycle are typically collected in a global variable list. At the end of the cycle, the symbolic client is notified by means of an additional variable (BOOL or counter) that the machine cycle has ended and the values are valid. Now the client has the chance to archive the values from the production cycle. If required, successful reading can also be displayed in opposite direction using a released variable. Thus, the production can be stopped, even if production data could not be archived. For this use case, the synchronized consistent access is neither required nor useful, as the synchronization has to take place on application level.

In contrast, synchronized consistent access by symbolic clients is usually used in the process industry with continuously running systems without production cycle if cyclic process values are to be written consistently within a specified time frame.

This can either be performed by synchronization on application level as for the clocked machines (see above) or via the synchronized consistent symbolic access. The advantage of the latter is that no logic has to be implemented in the IEC program and that the access is only controlled by the client.

CAUTION

Due to the increased jitter, the synchronized consistent monitoring is not suitable for Motion or real-time critical applications. Due to these reasons, the synchronized consistent access should only be released and used if absolutely required.

If a client uses the synchronous consistent access enabled via this setting, it also affects the client. Depending on the scheduler of the runtime system, the response time on read/write access can jitter more, as it has to be waited for the execution gap of the IEC tasks if required. Furthermore, read and/or write access can fail if IEC tasks run for a very long time (multiple 100 ms) or if the CPU load by one or multiple IEC tasks is close to 100% for a longer period.

Support for the current configuration and possible corrective actions

The availability of the values thus also depends on the control load by the IEC application.

Additionally, the client can minimize effects on itself and on the runtime system if considering the following when defining the variable lists to be read/written:

- Synchronously consistently access only those variables that are required as consistent
- Separated variable lists for variables that have to be consistent and for variables that may be inconsistent
- Split variable lists with many consistent variables into multiple smaller lists
- Select reading intervals for cyclic value reading as high as possible.

Entries marked in red in the symbol table show variables that they are configured for export to the symbol file but are currently invalid in the application. The cause for this can be that the declaration has been removed from the block.

In version 14VRS and later, a warning appears in the editor if variables that have configured symbols are not used in the IEC code or are not mapped in the case of I/O variables. In addition, the compiler indicates variables that are referenced from outdated library versions in the symbol configuration.



Object variables that are not used in the program code, are not compiled by default and are therefore not available in the symbol configuration.

But: However, IndraLogic provides variables from uncompiled objects in the symbol configuration when one of the following conditions is met:

- The "Link always" function block property is enabled.
- The `{attribute "linkalways"}` pragma attribute is used.

Also refer to

- [chapter 10.2 "Symbol configuration" on page 156](#)
- [chapter 10 "Working with control networks" on page 153](#)
- [chapter "Properties" - "Build" dialog" on page 705](#)
- [chapter "Attribute linkalways" on page 394](#)

Examples of data layout types

The following examples of an IEC application will show how gaps can result in the client-side memory layout caused by unpublished symbols, internal "invisible" pointers, or a "pack mode" definition in the device description. With the setting "Optimized layout", the gaps are avoided. The symbol file contains different information about the size and offset of memory locations, depending on the selected layout setting.

Large structure

```
// Beispiel einer großen Struktur, von der nicht alle Members veröffentlicht v
// Example of a big structure, where not all members get published :
STRUCT
{attribute 'symbol':='readwrite'}
PublicNumber : INT;

{attribute 'symbol':='none'}
InternalData : ARRAY[0..100] OF BYTE;

{attribute 'symbol':='readwrite'}
```

```

SecondNumber : INT;

{attribute 'symbol':='none'}
MoreData : ARRAY[0..100] OF BYTE;
END_STRUCT
END_TYPE

```

Resulting entries in the symbol file; check "size" and "byteoffset":

Symbol file, large structure, compatibility layout option

```

<TypeUserDef name="T_GrosseStruktur" size="208" nativesize="208"
typeclass="Userdef" pouclass="STRUCTURE" iecname="GrosseStruktur">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR"></UserDefElement>

<UserDefElement iecname="SecondNumber" type="T_INT" byteoffset="104"
vartype="VAR"></UserDefElement>

</TypeUserDef>

```

Symbol file, large structure, optimized layout option

```

<TypeUserDef name="T_GrosseStruktur" size="4" nativesize="208"
typeclass="Userdef" pouclass="STRUCTURE" iecname="GrosseStruktur">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR"></UserDefElement>

<UserDefElement iecname="SecondNumber" type="T_INT" byteoffset="2"
vartype="VAR"></UserDefElement>

</TypeUserDef>

```

Structure with uneven addresses

```

// Die folgenden Mechanismen können zu Members mit nicht korrektem
// Alignment führen: / The following mechanisms can cause misalignment:
// - {attribute 'relative_offset':='...'} an einem Member / at a member
// - {attribute 'pack mode':='...'} in einer Strukturdeklaration / at a
//   structure declaration
// - target setting 'memory-layout\pack-mode' in der Gerätebeschreibung /
//   in the device description

{attribute 'pack_mode':='1'}
TYPE UngeradeAdressen :
STRUCT
{attribute 'relative_offset':='3'}
{attribute 'symbol':='readwrite'}
PublicNumber : INT;

{attribute 'symbol':='readwrite'}
PublicValue : LREAL;
END_STRUCT
END_TYPE

```

Resulting entries in the symbol file; check "size" and "byteoffset":

Symbol file, structure with uneven addresses, compatibility layout option

```

<TypeUserDef name="T_UngeradeAdressen" size="13" nativesize="13"
typeclass="Userdef" pouclass="STRUCTURE" iecname="UngeradeAdressen">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="3"
vartype="VAR"></UserDefElement>

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="5"
vartype="VAR"></UserDefElement>

</TypeUserDef>

```

Symbol file, structure with uneven addresses, optimized layout option

```

<TypeUserDef name="T_UngeradeAdressen" size="16" nativesize="13"
typeclass="Userdef" pouclass="STRUCTURE" iecname="UngeradeAdressen">

```



```
<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0"
vartype="VAR"></UserDefElement>

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="8"
vartype="VAR"></UserDefElement>

</TypeUserDef>
```

Function block

// Each POU contains some implicit variables, which do not get published.
 // Depending on the data type these might cause memory gaps of different sizes.
 // FUNCTION_BLOCK Baustein IMPLEMENTS SomeInterface.

```
VAR_INPUT
  in : INT;
END_VAR
VAR_OUTPUT
  out : INT;
END_VAR
VAR
END_VAR
```

Each block contains some implicit variables which are not published. If it is a data type such as `__XWORD`, different sizes of memory gaps result in the client-side data layout, depending on whether the system is 64-bit or 32-bit.

Resulting entries in the symbol file for 64-bit and 32-bit; take "size" and "byteoffset" into consideration:

Symbol file, function block, compatibility layout option, 64-bit

```
<TypeUserDef name="T_Baustein" size="24" nativesize="24"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="16"
vartype="VAR_INPUT"></UserDefElement>

<UserDefElement iecname="out" type="T_INT" byteoffset="18"
vartype="VAR_OUTPUT"></UserDefElement>

</TypeUserDef>
```

Symbol file, function block, optimized layout option, 64-bit

```
<TypeUserDef name="T_Baustein" size="4" nativesize="24"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="0"
vartype="VAR_INPUT"></UserDefElement>

<UserDefElement iecname="out" type="T_INT" byteoffset="2"
vartype="VAR_OUTPUT"></UserDefElement>

</TypeUserDef>
```

Symbol file, function block, compatibility layout option, 32-bit

```
<TypeUserDef name="T_Baustein" size="12" nativesize="12"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="8"
vartype="VAR_INPUT"></UserDefElement>

<UserDefElement iecname="out" type="T_INT" byteoffset="10"
vartype="VAR_OUTPUT"></UserDefElement>

</TypeUserDef>
```

Symbol file, function block, optimized layout option, 32-bit

```
<TypeUserDef name="T_Baustein" size="4" nativesize="12"
typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecname="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="0"
vartype="VAR_INPUT"></UserDefElement>
```

```
<UserDefElement iecname="out" type="T_INT" byteoffset="2"
vartype="VAR_OUTPUT"></UserDefElement>

</TypeUserDef>
```

18.1.25 “Task configuration” object

Symbol: 

You use this object to define and display the basic settings for the task configuration.

The **Task configuration** object must be included exactly one time in each application.

Task configuration tabs and their functions

- **Properties:** Displays the basic settings
- **System events:** Links function block calls with system events
- **Monitoring:** Displays the status and current statistics for the cycles times in online mode

Also refer to

- [chapter 9.15.1 "Creating task configurations" on page 137](#)
- [chapter 18.1.26 ""Task" object" on page 565](#)

“Properties” tab

Object: **Task configuration**

In this tab, you define the basic settings of the task configuration as predefined by the target system, such as the maximum values for tasks and watchdog parameters.

“System events” tab

Object: **Task configuration**

In the **System events** tab, define which event calls which functions and whether the configuration is currently activated. You use this tab when a system event (instead of a task) should call a project function.

Add event handler	Opens the Add event handler dialog.
Delete event handler	Deletes the selected list assignment.
Event info	Shows information from the respective event library.
Open event function	Opens the editor of the new function for the selected assignment. You have selected the implementation language of the new function in the Add Event Handler dialog.
List of assignment of functions to be called for events with: Name , Description , Function to be called and Active (enabling/disabling the configuration).	

Adds a new assignment to the list: "Event - Function to be called".	
Event	The options, which can be selected, depend on the target device. IndraLogic marks unavailable events with a red symbol in front of the name. For a list of the available system events, go to the end of this section.
Function to be called	Function name (POU of type FUNCTION) Enter the name of the new function. IndraLogic inserts the function to the Project Explorer after the dialog is confirmed.

Validity range	• Application: Function is available for the application • POU's: Function is available for the complete project
Implementation language	Implementation language of the new function
Description	Brief description of the selected event

Special features in Online mode

The list of assignments from functions to be called to events also includes the following information: **Event status** and **Call count** and the **Online reset** button.

Event status	0: No error occurred. Unequal to 0: Error. You must consult the respective runtime system documentation.
Call count	Displays how often the event has occurred or the associated function has been called.
Online reset	IndraLogic reinitializes the event lists and resets the counter for the events/function calls. Incorrectly initialized events are displayed with a red status cell.

Possible system events

Event	Description	Task	Debug
PrepareStart	Call before starting the application	Communication task	No
StartDone	Call after starting the application	Communication task	No
PrepareStop	Call before stopping the application	Communication task	No
StopDone	Call after stopping the application	Communication task	No
PrepareReset	Call before resetting the application	Communication task	No
ResetDone	Call after resetting the application	Communication task	No
PrepareOnlineChange	Call before online change of the application	Communication task	No
OnlineChangeDone	Call after online change of the application	Communication task	No
PrepareDownload	Call before downloading the application	Communication task	No
DownloadDone	Call after downloading the application	Communication task	No
PrepareDelete	Call before deleting the application	Communication task	No
DeleteDone	Call after deleting the application	Communication task	No
PrepareExit	Call before exiting the application	Communication task	No
ExitDone	Call after exiting the application	Communication task	No
CodeInitDone	Event is sent after code init. Called within the safe section task and only for online change (for example, the copy code is executed here for online change).	Communication task	No
Exception	The event is sent if an exception has occurred in the context of an application.	Exception handling task of the runtime system or the task itself if the runtime system does not have one	Depending on the task
Login	Login of a client to this application	Communication task	No
Logout	Logout of a client from this application	Communication task	No

Event	Description	Task	Debug
BeforeReadingInputs	Call before reading the inputs	IEC task	yes
AfterReadingInputs	Call after reading the inputs	IEC task	yes
BeforeWritingOutputs	Call before writing the inputs	IEC task	yes
AfterWritingOutputs	Call after writing the inputs	IEC task	yes
DebugLoop	Event is sent in cycles to the debug loop if the IEC task stops at a break-point.	Communication task	No
PrepareShutdown	Event is sent immediately before shutting down the runtime system.	Runtime system, main loop	No
PrepareExitComm	Event is sent before exiting the communication server during shutdown	Runtime system, main loop	No
PrepareExitTasks	Event is sent before exiting all tasks during shutdown.	Runtime system, main loop	No

“Monitoring” tab

Object: Task configuration

This tab shows the following information in online mode: status for tasks in the task configuration and some current measurements of the cycles and cycle times. IndraLogic updates the values in the same time interval as for the monitoring of values from the control.

Reset the displayed values 0 by right-clicking the task and then clicking the **Reset** command in the context menu.

Task	Task name as defined in the task configuration
Status	Not created: Task was not started since most recent update; especially for event tasks Created: The task is detected in the runtime system, but not yet in operation Valid: Task is normal in operation Exception: Task has exceptional state
Number of IEC cycles	Number of cycles executed since starting the application where the IEC code was executed (0 if the target system does not support the counter function).
Number of cycles	Number of executed cycles since logging into the control. It depends on the target system whether cycles are also counted where the application is not running. In this case, the Cycle count can be greater than the IEC cycle count .
Latest cycle time (µs)	Last measured cycle time in [µs]
Average cycle time (µs)	Average cycle time across all cycles in [µs]
Max. cycle time (µs)	Maximum cycle time measured across all cycles in [µs]
Min. cycle time (µs)	Minimum measured cycle time via all cycles in [µs]
Jitter (µs)	Latest measured jitter in [µs] Jitter is the difference between the time when the task should be started and the time when the task is actually started. Example: If the task should be started every 1000 µs and it actually starts after 1100 µs, then the jitter is 100 µs.
Min. jitter (µs)	Minimum measured jitter in [µs]

Max. jitter (µs)	Maximum measured jitter in [µs]
Core	Number of processor cores on which the task is currently running. Example: 2 Prerequisite: The control has a multi-core CPU.

'Variable use' tab

Object: **Task configuration**

In the "Variable use" tab, an overview of all variables and their use is available. The tasks in which the variables are to be accessed are displayed.

When using multicore, write access (w) should only be possible to a variable in a task, as otherwise inconsistencies can occur.

Individual tasks can be hidden via the context menu and the cross-reference list for variables can be displayed.

Variables	Variable name
Type	Data type
Number	Number of tasks accessing this variable
<Task name>	Access to variable (r: read, w: write, rw: read/write)

Also refer to [chapter 9.15 "Task configuration" on page 136](#)

'Task groups' tab

Object: **Task configuration**

Requirement: Multicore control

Define the task groups in the "Task group" tab. Task groups can be distributed to individual processor kernels in case of multicore systems. Tasks of a task group always run on the same processor kernel.

Add group	The interface adds a new task group with the name <code>NewGroup_<nr></code> .
Remove group	Deletes the selected task group
Group name	The name can be changed by double-clicking into the text field.
Core	Specifies in which process kernel the task of this group is processed. • "FreeFloating": All tasks are dynamically connected to different processor kernels - the operating system assigns the tasks, the user cannot influence the assignment. • SequentialPinned: All tasks are connected to different processor kernels - the operating system assigns the tasks, the user cannot influence the assignment. • "FixedPinned": All tasks are connected to a processor kernel. Which processor kernel the tasks are connected to is decided in the runtime system (standard setting). • "<Core number>": Defined processor kernel. If the process kernel is not available, an error message is output.

Also refer to [chapter 9.15 "Task configuration" on page 136](#)

18.1.26 "Task" object

Symbol: 

Define the conditions to start the task and to call the task in the object.

Insert the object below **Task configuration** in the Project Explorer.

Also refer to

- [chapter ""Configuration" tab" on page 566](#)
- [chapter 18.1.25 ""Task configuration" object" on page 562](#)

"Configuration" tab

Object: **Task**

Priority	Possible values: 1...20, 1 has the highest priority
Task group (Prerequisite: Multicore control)	Assigned task group. This assignment is displayed in the device tree in parentheses. Task groups can be assigned to certain processor kernels in multicore systems. This task group is displayed in the device tree in parentheses.

Type

Cyclic	IndraLogic processes the task in cycles. The cycle time of the task is defined in the Interval field.
Event	IndraLogic starts processing the task when the global variable defined in the Event field, receives a rising edge.
Free running	IndraLogic starts processing the task again automatically in a continuous loop at program start and at the end of a complete pass. No cycle time is defined.
external	IndraLogic starts processing the task as soon as the event defined in the Event input field occurs. The events supported and provided in the selection list depend on the target system. (Do not confuse with system events).
Interval	Task cycle time Required for the Cyclic type if the event requires a time specification. Period after which the task is to be restarted. If you enter a number here, then you can select the desired unit in the dropdown list after the input field. If you select ms , then an entry is automatically displayed in TIME format, for example <code>t#200ms</code> , as soon as the window is in focus again. The task cycle time can also be entered directly as TIME format. Entries in <code>[μs]</code> format is always displayed as an abstract number.



A defined cycle grid is required for field buses to ensure a determined response. Therefore, do not use the **type free-running** for a bus cycle task.



Please note the difference between the processing types **Status** and **Event**:

If the given event yields `TRUE`, then the start condition of a task of type **Status** is satisfied.

In contrast, the start for an **Event** type task requires a change in the event from `FALSE` to `TRUE`. If the scanning rate of the task scheduler is too low, then the rising edge of the event can remain unnoticed.

Watchdog

Defines the time monitoring for a task. If the target system supports an extended watchdog configuration, it is possible that the following settings are predefined in the device description:

- Upper and lower limit
- Standard watchdog time
- Time specified in percent

The default watchdog settings depend on the device.

Activate	Watchdog is active. If the task exceeds the currently set Time of the watchdog, then the task is halted with an error status (exception). The application in whose task the error occurred and its child applications are also halted. In this way, all tasks of the affected applications are also halted. The currently specified sensitivity is included in the calculation. If you activate the option Update I/Os in the control Settings of the control, then IndraLogic re-sets the outputs to the defined default values. <i>Possible cases:</i> • Multiple subsequent timeouts: Sensitivity: 0, 1 - exception in cycle 1 Sensitivity: 2 - exception in cycle 2 Sensitivity: n - exception in cycle n • One-time timeout: Exception if the cycle time of the current cycle is longer than (time * sensitivity). Example: Time=t#10ms, Sensitivity=5 (i.e., exception as soon as the one-time task runs longer than 50 ms)
Time (example: t#200ms)	Watchdog time Defined the time monitoring for a task together with the Sensitivity as described in Enabling. Depending on the target system, the monitoring interval might have to be entered in percent of the task interval. In this case, the drop-down list for the unit is disabled and displays %.
Sensitivity	Number Defined the time monitoring for a task together with the watchdog time as described in Enabling.



Using the functions from the library `CmpIecTask.library`, you can deactivate a watchdog for specific PLC cycles. This is useful for cycles that demand more time due to initialization.

Example

Disable/re-enable watchdog:

```
hIecTask := RTS_IEC_HANDLE //Declaration of the variable hIecTask
hIecTask := IecTaskGetCurrent(0);
IecTaskDisableWatchdog(hIecTask); // Watchdog disabled
...
IecTaskEnableWatchdog(hIecTask); Watchdog enabled
```

List of the POU's controlling the task.

The calling order corresponds to the POU order in the list (from top to bottom).

Add call	Defines a new program call

Open function block	Opens the selected POU
Move up Down	Change of calling sequence.

Also refer to

- [chapter 18.1.25 "Task configuration" object" on page 562](#)
- [chapter "Task groups' tab" on page 565](#)

18.1.27 "Trace" object

Symbol: 

The object displays trace data in one or more diagrams. During application runtime, value graphs of trace variables are recorded on the controller. The graphs are displayed in the trace editor so that you can monitor the value graphs. The requirement is setting of a trace configuration, the transfer of the trace configuration to the control and starting of the trace feature. The recorded data is transferred to the development system and displayed in diagrams as configured. During tracing, navigation in the data is possible.

Double clicking on the object opens the trace editor. The corresponding toolbar provides access to the most important trace commands. The trace variable list provides an overview of the trace configuration.

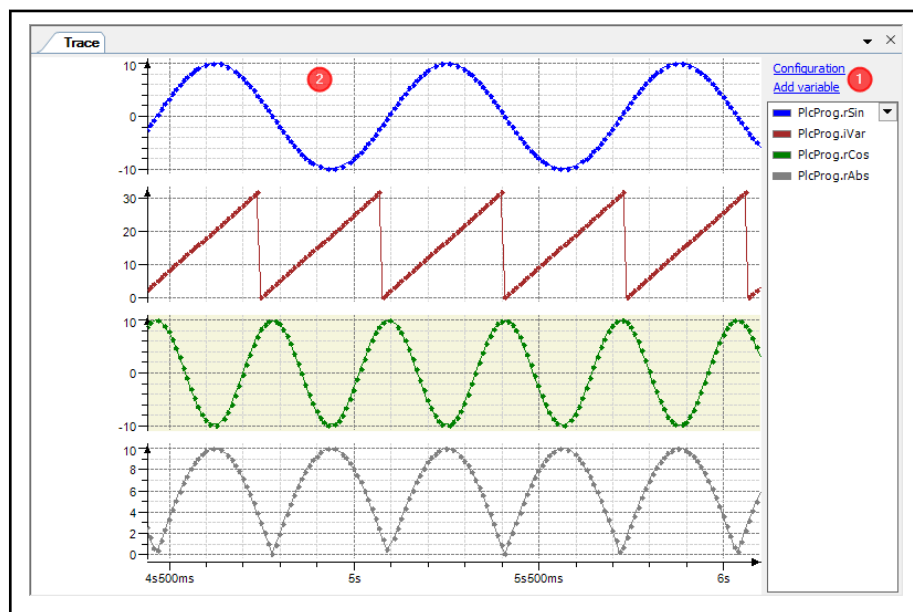


Fig. 18-36:

- (1): View of the trace configuration
 - Configuration: **Configuration**
 - Add variable: **Add variable**
- (2): Trace diagram in multi-channel display at runtime

"Trace" editor

View of the trace configuration

This view provides an overview of the current trace configuration where you can edit the configuration. Double-clicking a variable opens the **Trace configuration** dialog and shows the respective configuration in **Variable settings**.

The following commands and context menu commands are also provided.

Configuration	The Trace configuration dialog opens with the selected top node. The configuration for data recording is displayed on the right below Record Settings for quick configuration.
Add variable	The Trace configuration dialog opens with a new and blank entry in the trace configuration. The configuration of a new entry is displayed on the right below Variable settings to make immediate settings.
	Selection list opens with the Hide instance paths command
Hide instance paths	<i>Toggles the representation of the variable list</i> ☒ : Display with complete instance path (for example, <code>PlcProg.iCounter</code>)

Add variable	The Trace configuration dialog opens with a new and blank entry in the trace configuration. The configuration of a new entry is displayed on the right below Variable settings to make immediate settings.
Remove variable	The variable that is selected in the view is removed from the configuration.
Visible	<i>Toggles the visibility of the selected variable in the trace diagram:</i> ☒ : Graph of variables is visible in the trace diagram.
Display	The Y-axis display settings dialog opens.
Configuration	The Trace configuration dialog opens with the Record Settings .

Tab. 18-9: Context menu

Also refer to

- [chapter "'Trace configuration' dialog" on page 737](#)

Trace diagram at runtime

At application runtime, IndraLogic populates and updates the trace buffer with the recorded samples, and then it displays the recorded data in the trace diagram as a graph over time. If you close the trace editor, the buffer will be freed.

Use menu commands for controlling data recording. You can navigate the data by means of menu commands, keyboard shortcuts, and mouse input.

Also refer to

- [chapter 13.3.3 "Controlling data recording" on page 207](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

Navigating the trace diagram with the mouse

In the trace diagram, you can navigate data by using the mouse to scroll and zoom.

Prerequisite	User-input via mouse	Mouse cursor at user input	Effect
	Drag the graph along the x-axis		Horizontal scrolling of the trace graph along the time axis
	Hold down the <Ctrl> key and drag the graph along the y-axis		Vertical scrolling of the trace graph along the y-axis.
Multi-channel disabled Mouse zooming enabled:	Stretch a rectangle		IndraLogic zooms to the rectangle
	Move the mouse wheel back-wards		IndraLogic compresses the x-axis like the  Compress

Prerequisite	User-input via mouse	Mouse cursor at user input	Effect
	Move the mouse wheel forwards		IndraLogic stretches the x-axis like the  Stretch command
	Hold down the <Ctrl> key and move the mouse wheel backwards		IndraLogic compresses the y-axis
	Hold down the <Ctrl> key and move the mouse wheel forwards		IndraLogic stretches the y-axis
Multi Channel enabled	Click into a trace diagram		The trace diagram is selected
One or two trace cursors are provided	Drag the triangle of a trace cursor to another position		Horizontal moving of the trace graph along the time axis
Two trace cursors available	Click on the gray trace cursor		The gray trace cursor is black and therefore selected

Also refer to

- chapter "Zoom with mouse" command" on page 692

Navigating the trace diagram with the keyboard

Prerequisite	Shortcuts	Effect
	<←> <→>	IndraLogic scrolls the trace graph along the x-axis
	<Ctrl> + <←> <Ctrl> + <→>	IndraLogic scrolls the trace graph along the x-axis in large intervals
	<↑> <↓>	IndraLogic scrolls the trace graph along the y-axis
	<Ctrl> + <↑> <Ctrl> + <↓>	IndraLogic scrolls the trace graph along the y-axis in large intervals
Multi Channel enabled	<Up arrow> <↓>	IndraLogic scrolls the trace graph along the x-axis in the selected trace diagram
Multi Channel enabled	<Ctrl> + <↑> <Ctrl> + <↓>	IndraLogic scrolls the trace graph along the x-axis in the selected trace diagram in large intervals
One or two trace cursors are provided	<Alt> + <←> <Alt> + <→>	IndraLogic scrolls the trace graph along the x-axis in the selected trace diagram
One or two trace cursors are provided	<Ctrl> + <Alt> + <←> <Ctrl> + <Alt> + <→>	IndraLogic scrolls the trace graph along the x-axis in the selected trace diagram in large intervals
	<->	IndraLogic compresses the x-axis like the  Compress

Prerequisite	Shortcuts	Effect
	<+>	IndraLogic stretches the x-axis like the  Stretch command
Multi-channel disabled	<Ctrl>+<->	IndraLogic stretches the x-axis
Multi-channel disabled	<Ctrl>+<+>	IndraLogic stretches the y-axis
Multi Channel enabled	<Ctrl>+<->	IndraLogic compresses the y-axis of the selected trace diagram
Multi Channel enabled	<Ctrl>+<+>	IndraLogic stretches the y-axis of the selected trace diagram
Multi Channel enabled	<Tab>	Selection of the next lower trace diagram
One or two trace cursors are provided	<-> <->	IndraLogic moves the black trace cursor
One or two trace cursors are provided	<Ctrl> + <-> <Ctrl> + <->	IndraLogic moves the black trace cursor in large intervals
One or two trace cursors are provided	<Shift>+<-> <Shift>+<->	IndraLogic moves the gray trace cursor
One or two trace cursors are provided	<Ctrl>+<Shift>+<-> <Ctrl>+<Shift>+<->	IndraLogic moves the gray trace cursor in large intervals

Also refer to

- [chapter "'Trace configuration' dialog" on page 737](#)
- [chapter "'Cursor' command" on page 689](#)
- [chapter 13 "Application at runtime" on page 191](#)

18.1.28 "Access control" object

User management in a project only makes sense if it is connected with corresponding rights assignment for the access to project and objects. Observe the following:

- In a **new project** all rights are set with a **default value** (standard right). This default value is normally "permitted".
- **Each right can be explicitly assigned ("granted") or removed ("denied")** or reset to the default value. The access rights are managed in the "Rights" dialog or - for the objects access rights - in the "Access control" dialog. It is part of the "Properties" dialog.
- Access rights to objects can be **"inherited"**.

If an object has a "father" object, the rights of the father are automatically applied as default values for the child.

Father-child relations between objects with regard to the access rights normally correspond to relations shown in the Project Explorer are displayed in the "Rights" dialog as per the syntax "<FatherObject>.<Child-Object>".

Example: "ACT" action is assigned to the POU object "PlcProg". "ACT" is indented in the Project Explorer below "PlcProg".

In the "Rights" dialog, "ACT" is represented by "PlcProg.ACT" indicating that "PlcProg" is the "father" of "ACT".

If the "modify" right is explicitly "denied" for "PlcProg" and a specific user group, it is also "denied" for "ACT" for the default value of the "modify" right.

To apply the right for a PLC object, the object is selected via the context menu item **Properties** of the **Access control** tab. The corresponding field is selected in the table and the drop-down list is extended via the <space bar> or via a mouse click and the right is selected.

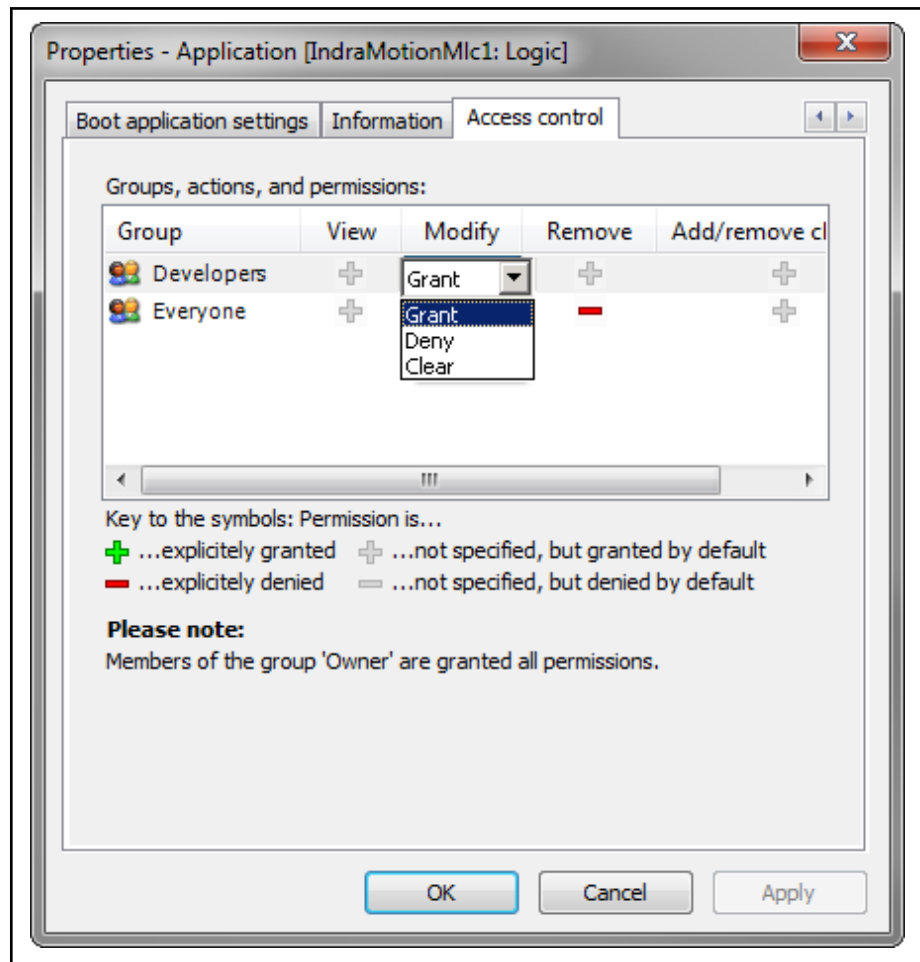


Fig. 18-37: "Access control" dialog in the properties of an application

First, assign the user rights for the respective access control:

- Grant
- Deny
- Clear

As each user can completely access an object via the "Everyone" group by default, the rights for the "Everyone" group has to be set to "Deny" for an effective object protection!

18.2 Menu commands

The most important commands are available in the IndraLogic user interface by default.

To customize the menu configuration, select **Tools ► Customize ► Commands**.

18.2.1 Menu 'Edit'

Standard commands

IndraLogic provides the following standard commands:

- Undo: , shortcut: <Ctrl> + <Z>
- Restore: , Shortcut: <Ctrl> + <Y>
- Cut: , shortcut: <Ctrl> + <X>
- Copy: , shortcut: <Ctrl> + <C>
- Insert: , shortcut: <Ctrl> + <V>
- Delete: , shortcut: <Del>
- Select all: Shortcut: <Ctrl> + <A>

The "Paste" command is not supported by all editors or the use is limited in some editors. In graphic editors, the command is only supported if a correct construct is generated by the paste operation.

In the Project Explorer, this command applies to the currently selected object. Several items can be selected at once.

Search and replace



The command group "Search and replace" is only available if the IndraLogic editor is open. For example for editing program organization units (POUs) or function blocks.

'Find' command

Symbol: 

Function: The commands scan the project or parts of it for a specified character string.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [Ctrl] + [F].

This command opens the "Find" dialog where the searched character string is specified and the search options are defined.

Look for	Character string to be searched.
Match case:	☒ : The search considers uppercase and lowercase.
Match whole word:	☒ : Only character strings with an exact match are found.
Search up:	☒ : The search area entered moves upwards. ☐ : The specified search range runs down.
Use regular expressions:	Use the  button to receive support when specifying regular expressions.

Search in:	 : Drop-down list with the areas of the project to be searched: • Active editor • All open editors • Selected objects & subobjects • Entire project • Entire project & uncompiled libraries • Selection only  : Opens a dialog where you set the areas of the project to be searched (see below)
Find next	Start the search
Find all	All search results are listed in the message view with their object path, project name, object name, and object position. Possible additional information for position: • (Decl): Declaration part of the object • (Impl): Declaration part of the object Double-clicking on the entry in the list opens the match position in the respective object editor.
Replace	Switches to the "Replace" dialog



The color used for highlighting search results can be adjusted in the options of the text editor. The color is adjusted via parameter "Selection color" - "Inactive" in the "Text area" tab.

Also refer to ["Tab "Text area"" on page 732](#)

Dialog for setting the objects to be searched

Entire project	All editable positions in all objects of the project are searched.
Within the following objects	Only the editable positions within the objects defined here are browsed: • Scheme: The Save command saves the current search configuration under the specified name. All saved schemes are available in the drop-down list (). • Object types: : The object is searched • Name filter: Name filter for the objects to be scanned. The placeholder "*" can be used. Example: Filters "*CAN*": All objects are searched that contain "CAN" in the name.
All open editors	All editors are searched that are currently open in a window.
Active editor	Only the editor is searched where the cursor is currently positioned in.
Only selection	Only the text is searched that is currently selected in an object.

Also refer to

- [chapter "'Replace' command" on page 574](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

'Replace' command

Symbol: 

Function: These commands scan the project or parts of it for a specified character string and replaces it.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [Ctrl] + [H].

Prerequisite: The application is in online mode.

This command opens the "Replace" dialog where the search and replace character strings are specified and the search options are defined.

Replace with	Input field for the new character string.
Replace	Each next found string is highlighted in the editor and replaced (step-by-step replace).
Replace all	All found strings are replaced at one time without them being displayed in the editors.
Leave changed objects open after "Replace all"	The editors of the found objects remain open.

Tab. 18-10: In addition to the options of the **Search** dialog, the following settings can be selected:



Replacement in referenced libraries is not possible.

Also refer to

- [chapter "'Find' command" on page 573](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

Command 'Search down'

Symbol:

Function: When searching a character string in the project, this command selects the next result at the position in the editor.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [F3].

Prerequisite: By using the commands **Find** or **Replace**, the search for a certain character string in the project was started.

Also refer to

- [chapter "'Find' command" on page 573](#)
- [chapter "'Replace' command" on page 574](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

Command 'Search up'

Symbol:

Function: When searching a character string in the project, this command selects the previous result at the position in the editor.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [Shift] + [F3].

Prerequisite: By using the commands "Find" or "Replace", the search for a certain character string in the project was started.

Also refer to

- [chapter "'Replace' command" on page 574](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

Command 'Search down' (selection)

Function: The command searches for the next character string in the project that matches the string currently selected or in which the cursor is currently positioned in.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [Ctrl] + [F3].

Prerequisite: You positioned the cursor in an editable character string the project or you selected an editable character string.

Also refer to

- [chapter "'Find' command" on page 573](#)
- [chapter "'Replace' command" on page 574](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

Command 'Search up' (selection)

Function: The command searches for the previous character string in the project that matches the string currently selected or in which the cursor is currently positioned in.

Call: IndraWorks menu **Edit ► Search and replace**, optionally via the shortcut [Ctrl] + [Shift] + [F3].

Prerequisite: You positioned the cursor in an editable character string the project or you selected an editable character string.

Also refer to

- [chapter "'Find' command" on page 573](#)
- [chapter "'Replace' command" on page 574](#)
- [chapter 9.13 "Project-comprehensive "Find and replace"" on page 132](#)

Bookmark



The command group "Bookmark" is only available if the IndraLogic editor is open. For example for editing program organization units (POUs) or function blocks.

'Toggle bookmark' command

Symbol:

Function: The command sets or deletes a bookmark at the current position.

Call: IndraWorks menu **Edit ► Bookmark**, optionally via the shortcut [Ctrl] + [F12].

Prerequisite: A POU is open in the editor and the cursor is on a program line.

Also refer to

- [chapter 9.12.3 "Setting and using bookmarks" on page 131](#)

'Next bookmark' command

Symbol:

Function: The command jumps to the next bookmark.

Call: IndraWorks menu **Edit ► Bookmark**, optionally via the shortcut [F12].

Prerequisite: A POU is open in the editor and the cursor is at a program line.

Also refer to

- [chapter 9.12.3 "Setting and using bookmarks" on page 131](#)

'Previous bookmark' command

Symbol: 

Function: The command jumps to the previous bookmark.

Call: IndraWorks menu **Edit ► Bookmark**, optionally via the shortcut [Shift] + [F12].

Prerequisite: A POU is open in the editor and the cursor is on a program line.

Also refer to

- [chapter 9.12.3 "Setting and using bookmarks" on page 131](#)

'Delete bookmark' command

Symbol: 

Function: The command deletes all set bookmarks.

Call: IndraWorks menu **Edit ► Bookmark**

Prerequisite: A POU is open in the editor and the cursor is on a program line.

Also refer to

- [chapter 9.12.3 "Setting and using bookmarks" on page 131](#)

Insert file as text



The command "Insert file as text" is only available if the IndraLogic editor is open. For example for editing program organization units (POUs) or function blocks.

Command 'Insert file as text'

Function: This command copies the contents of a text file to the active editor to the current cursor position.

Call: IndraWorks menu **Edit**

Prerequisite: The file must have the extension .txt. The command is available in a text editor only.

Many development environments and text processing applications provide the option of exporting code and text as a plain text file. This command can copy the contents of this file to the editor.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Browse

'Go to definition' command

Symbol: 

Function: The command shows the definition locations of a variable or function.

Call: IndraWorks menu **Edit ► Scan** in the implementation part of the editor: **Scan** context menu

Prerequisite: A POU is open in the editor and the cursor is at a variable or function.

Also refer to

- [chapter 9.12.2 "Finding declaration - "Go to definition"" on page 130](#)

Command 'Go to Reference'

Symbol: 

Function: The command opens the declaration location of the variable that is referenced by the pointer currently in focus in online mode.

Call:

- Via the context menu in the declaration part or implementation code
- **Edit ► Browse**

Prerequisite: Online mode A POU is open in the editor and the cursor is at a pointer. The referenced variable is stored in static memory.



If the pointer does not point exactly to the beginning of the variable, then a corresponding message is displayed when you switch to the variable declaration.

Also refer to

- [chapter 17.5.12 "Data type 'POINTER'" on page 359](#)

Command 'Go to instance'

Symbol: 

Function: The command opens the instance of a function block in a new window.

Call: **Edit ► Browse**

Prerequisite: The application is in online mode. A POU is open in the editor and the cursor is at an instance of a function block.



The command is not available for temporary instances or instances from compiled libraries.

Also refer to

- [chapter 9.12.2 "Finding declaration - "Go to definition"" on page 130](#)

'Browse cross references' command

Symbol: 

Function: The command opens the cross-reference list, showing all occurrences of a variable.

Call: IndraWorks menu **Edit ► Browse** Optionally, via the context menu in the implementation part of the editor **Scan**.

Prerequisite: A POU is open in the editor and the cursor is positioned in a variable. Or the "Cross reference list" view is open and a variable is specified in the "Name" field.

Also refer to [chapter 9.12.1 "Using the cross-reference list to find places of use" on page 129](#)

"Dump call tree" command

Symbol: 

Function: This command opens the "Call tree" view which displays the calls of the block as well as its callers.

Call: IndraWorks menu **Edit ► Browse** Optionally, via the context menu of the editor in the **Scan** implementation part.

Prerequisite: A function block is open in the editor and the cursor is located in a variable or a function block is selected in the "Application" device node or "General module folder".

Refer to [chapter "Command "Call tree"" on page 592](#)

Refactoring



The command group "Refactoring" is only available if a Program Organization Unit (POU) is selected in the Project Explorer. In this case, only the "Rename refactoring" command is available.

All other refactoring commands become available if the IndraLogic editor is opened. For example for editing program organization units or function blocks.

'Rename refactoring' command

Symbol: 

Function: The command opens a dialog for renaming an object or variable across the project.

Call: IndraWorks menu **Edit ► Refactoring**, optionally in the context menu of the editor in the implementation part **Refactoring**.

Prerequisite: An object is selected in the Project Explorer below the application or in the "General module folder" or the cursor is placed in front of or on a variable identifier in the declaration section of a programming object.

The following can be renamed:

- Variables
- Program organization units (POU)
- Global Variable Lists (GVL)
- Methods
- Properties
- Devices
- Variables and unit conversions in the unit conversion edit

"Rename" dialog

Current name	Name of the object or the variable
New name	Input field for a new name. If the name already exists, then IndraLogic reports this directly below this input field.
OK	Can be activated if you have typed a valid name in New name . Opens the Refactoring dialog The affected objects and occurrences are highlighted in both views. The handling for each place of use can be specified in both windows. The following commands are available in the context menu

"Refactoring" dialog

This dialog displays all occurrences in the project. The affected objects and occurrences are highlighted in both views.	
Right view	Displays the occurrence within an object where Current name occurs.
Left view	Device tree of the project with the object.
The handling for each place of use can be specified in both windows. The following commands are available in the context menu:	
Discard this change	Discard the single change in the right window
Accept this object	Accept all changes in the respective object
Discard this object	Discard all changes in the respective object
Apply complete project	Accept all changes in the project
Discard whole project	Discard all changes in the project
IndraLogic highlights the accepted changes in yellow and the rejected changes in gray.	

Also refer to

- [chapter 9.14 "Refactoring" on page 132](#)

Command 'Add variable'

Symbol: 

Function: The command enables the declaration of variables in a POU, as well as the automatic update to the occurrence of the POU.

Call: IndraWorks menu **Edit ► Refactoring**, optionally in the context menu of the editor in the declaration part.

Prerequisites: The declaration section of a POU is focused

The command opens the default dialog for declaring variables.

Also refer to

- [""Declare variable" dialog" on page 588](#)

"Refactoring" dialog

After clicking **OK** to close the declaration dialog, the **Refactoring** dialog opens with two frames.

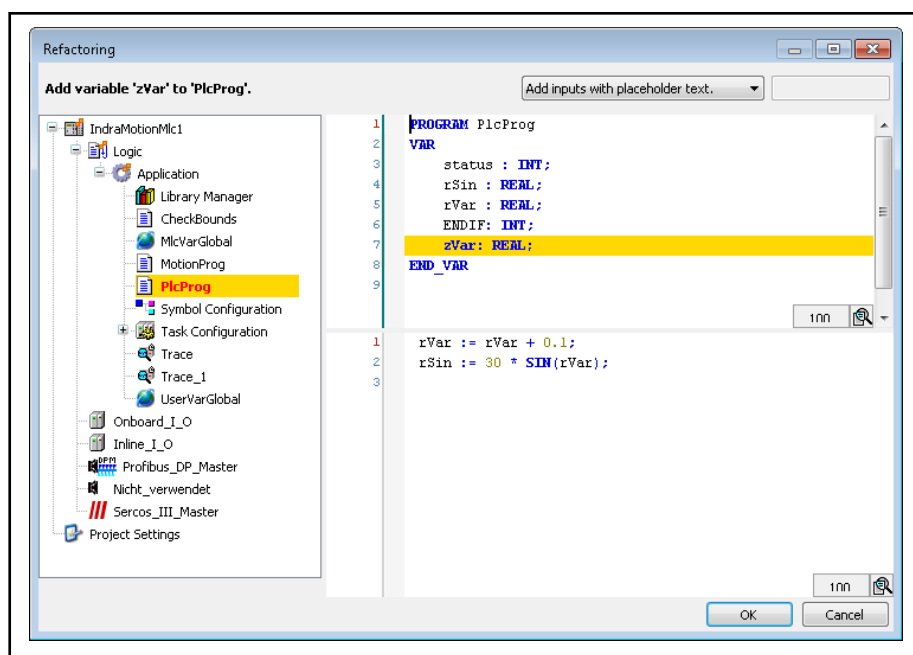


Fig. 18-38: Refactoring dialog, split

Right dialog frame	Declaration section and implementation of the POU where the variable is added Colored highlighting of changed locations: New added declarations have a blue font and are highlighted in yellow (1).
Left dialog section	Project Explorer or in the "General module folder" of the project. Colored highlighting of blocks where the POU is used: Red font and yellow highlight (2). After you double-click the POU object, the detail view opens.

Before you decide which changes to accept at which locations, select the required option from the drop-down list (3) at the upper right part of the window:

Add inputs with placeholder text	Default placeholder text : <code>_REFACTOR_</code> ; editable The placeholder text defined here is used at the occurrence locations of the new added variables in the implementation code. This is used to search for the respective locations.
Add inputs with the following value	Initialization value for the new variable

You can accept or reject changes by right-clicking the changed locations or by executing commands in the left or right area of the dialog.

Examples

- By refactoring, the `fun` block receives a new input variable `input3` with the initialization value 1. The change has the following effect:

Before:

```
fun(a + b, 3, TRUE);
fun(input1:= a + b , input2 :=3 , inputx := TRUE);
```

After:

```
fun(a + b, 3, 1, TRUE);
fun(input1:= a + b , input2 :=3 , _REFACTOR_ , inputx := TRUE);
```

- By refactoring, the "fun" block receives a new input variable `input3` with the placeholder text "`_REFACTOR_`":

Before:

```
inst(input1 := a + b, input2 := 3, inputx := TRUE);
fun(a + b, 3, TRUE);
```

After:

```
inst(input1 := a + b, input2 := 3, input3 := _REFACTOR_, inputx := TRUE);
fun(a + b, 3, _REFACTOR_, TRUE);
```

Also refer to

- [chapter 9.14 "Refactoring" on page 132](#)
- [chapter "'Rename refactoring' command" on page 579](#)

Command 'Remove variable'

Symbol: 

Function: The command removes an input variable or an output variable from the POU and all locations of use of the POU.

Call: IndraWorks menu **Edit ► Refactoring**, optionally in the context menu of the editor in the declaration part.

Prerequisites: In the declaration section of the POU, the cursor is located in the identifier of the variable to be removed.

Then, the command opens a dialog with information about the removal. After you confirm this, the **Refactoring** dialog opens.

For a description of the **Refactoring** dialog, refer to the **Edit ► Refactoring ► Rename** help page.

When you accept the changes in the **Refactoring** dialog, the respective input and output parameters are deleted at the occurrence locations of the affected POU.



In CFC, only the connection is removed between the removed input or output to the block. The input or output itself remains in the chart.

Example in ST

In a POU, "refactoring" removes the `input4` input variable. The occurrences are updated automatically:

Before removal:

```
inst(input1 := a + b, input2 := 3, input4 := 1, input5 := TRUE);
fun(a + b, 3, 1, TRUE);
```

After removal:

```
inst(input1 := a + b, input2 := 3, input5 := TRUE);
fun(a + b, 3, TRUE);
```

Also refer to

- [chapter 9.14 "Refactoring" on page 132](#)
- [chapter "'Rename refactoring' command" on page 579](#)

Command 'Rearrange variables...'

Symbol: 

Function: This command allows changing the order of variables in the declaration editor for the selected scope: VAR_INPUT, VAR_OUTPUT, or VAR_IN_OUT.

Call: IndraWorks menu **Edit ► Refactoring**; context menu of the focused scope in the declaration editor.

Prerequisite: One of the above scopes is selected in the declaration, and more than one variable is declared in it.

The command opens the "Reorder" dialog box with a list of all declarations of the selected scope. You can drag a selected declaration up or down to another position.

Also refer to

- ["Reordering variables in the declaration" on page 135](#)

Command 'Refresh affected connections'



Currently, this command applies only to the CFC, FBD, LD, and IL editors. It is a combination of the **Reset Pins** and **Update Parameters** commands.

Symbol:

Function: The command modifies the pins according to the latest block declaration in all affected occurrences of the block.

Call: Menu **Edit ► Refactoring**, optionally in the context menu of the editor.

Prerequisite: The cursor is placed in the name of the block in the first line of the block declaration or in the Project Explorer.

Also refer to

- [chapter 9.14 "Refactoring" on page 132](#)
- [chapter ""Reset connections" command" on page 656](#)
- [chapter ""Update parameters" command" on page 669](#)

Advanced



The command group "Advanced" is only available if the IndraLogic editor is open. For example for editing program organization units or function blocks.

'Show space' command

Symbol:

Function: With this command, control characters are displayed for spaces and tabs.

Call: IndraWorks menu **Edit ► Extended**

Prerequisite: A text editor is selected.

IndraLogic visualizes spaces by a period and tabs by an arrow.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

"Show indentation help lines" command

Function: The command enables the indentation help lines.

Call: Menu **Edit ► Advanced**

Prerequisite: A text editor is open.

If the indentation help lines are activated, a broken line is inserted for each manual indentation in the code. This facilitates the overview of the different levels in the code. Insert manual indentations with <Tab>.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Command 'Expand all'

Function: The command collapses all indented code sections (minus symbol at the root node) in the text editor or sources in the cross-reference list. As a result only the highest level of the code, or only the root nodes of the find locations are displayed.

Prerequisite: A text editor is active and indentation is activated in the "Options" dialog, "Text editor" category at the same time. Alternatively, the cross-reference list is active.

Call: Generally via the IndraWorks menu **Edit ► Advanced**, optionally via the context menu in the editor **Advanced**. In the cross-reference list via the context menu.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)
- [chapter "'Expand all' command" on page 584](#)
- ["Context menu commands in the cross-reference list" on page 592](#)

'Expand all' command

Function: The command expands all collapsed code segments (plus symbol) in the text editor or sources in the cross-reference list so that the code and all sources are displayed in full again.

Prerequisite: A text editor is active and indentation is activated in the "Options" dialog, "Text editor" category at the same time. Alternatively, the cross-reference list is active.

Call: Generally via the IndraWorks menu **Edit ► Advanced**, optionally via the context menu in the editor **Advanced**. In the cross-reference list via the context menu.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)
- [chapter "Command 'Expand all'" on page 584](#)
- ["Context menu commands in the cross-reference list" on page 592](#)

Command 'Uncomment selected lines'

Symbol: 

Shortcut: [Ctrl] + [O]



The command is not part of the standard setting of the IndraWorks interface.

If required, adjust the command in the following menu: **Edit ► Advanced...**

To adjust the command, open the menu **Tools ► Adjust...**

In the dialog, change to the "Commands" tab.

In the left column, select the category "Commands".

In the command list, scroll to the command and Drag&Drop it to a position in the IndraWorks menu.

Prerequisite: In the ST editor, either the cursor is located in a line of the implementation or multiple lines are selected.

Call: Menu **Edit ► Advanced...**

Function: The command inserts a comment signs ("//") at the beginning of the selected lines.

Also refer to

- [chapter "Command 'Remove uncomment selected lines'" on page 585](#)

Command 'Remove uncomment selected lines'

Symbol:

Shortcut: [Ctrl] + [I]



The command is not part of the standard setting of the IndraWorks interface.

If required, adjust the command in the following menu: **Edit ► Advanced...**

To adjust the command, open the menu **Tools ► Adjust...**

In the dialog, change to the "Commands" tab.

In the left column, select the category "Commands".

In the command list, scroll to the command and Drag&Drop it to a position in the IndraWorks menu.

Prerequisite: In the ST editor, either the cursor is located in a line of the implementation or multiple lines are selected.

Call: Menu **Edit ► Advanced...**

Function: The command removes any comment signs ("//") at the beginning of the selected lines.

Also refer to

- [chapter "Command 'Uncomment selected lines'" on page 584](#)

Command 'Go to'

Shortcuts: [Ctrl] + [G]

Function: With this command the cursor jumps to a defined line in the code.

Call: IndraWorks menu **Edit ► Extended**

Prerequisite: A text editor is selected.

The command opens a dialog with an input field for the line number.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Command 'Make upper case'

Function: The command converts all lower case letters in the selected code into upper case letters.

Call: Menu **Edit ► Advanced**, optionally via the shortcut [Ctrl] + [Shift] + [U].

Prerequisite: A text editor is open and code is selected or the declaration editor is open and variable declarations are selected.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Command 'Make lower case'

Function: The command converts all upper case letters in the selected code into lower case letters.

Call: Menu **Edit ► Advanced**, optionally via the shortcut [Ctrl] + [U].

Prerequisite: A text editor is open and code is selected or the declaration editor is open and variable declarations are selected.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Command 'Corresponding parenthesis'

Function: The command makes the cursor jump to the respective part of the selected code parenthesis.

Call: Menu **Edit ► Advanced**

Prerequisites: A text editor is open and an opening or closing code parenthesis is highlighted.

The option "Associated parentheses" in the "Text editor" category, "Text area" tab is selected in the "Options" dialog.

Also refer to

- ["Tab "Text area"" on page 732](#)

Command 'Highlight up to corresponding parenthesis'

Function: The commands highlights the complete code section between the two corresponding parentheses.

Call: Menu **Edit ► Advanced**

Prerequisites: A text editor is opened and the cursor is positioned at an opening or closing code parenthesis.

The option "Associated parentheses" in the "Text editor" category, "Text area" tab is selected in the "Options" dialog.

Also refer to

- [chapter ""Options" dialog - Text editor" on page 731](#)

Input assistance



The command "Input assistant" is only available if the IndraLogic editor is open. For example for editing program organization units (POUs) or function blocks.

'Input wizard' command**Symbol:** **Function:** The command opens the "Input assistant" dialog supporting the selection of a possible programming element at the current cursor position.**Call:** IndraWorks menu **Edit**, optionally via the shortcut [F2] or via the context menu of the editor **Input assistant**.**Prerequisite:** A POU is open in the editor and the cursor is positioned in a program line.*Also refer to*

- [chapter 9.5 "Using input assistants" on page 107](#)

"Input assistant" dialog, tab "Categories"

The dialog provides all program elements that you can insert at the current cursor position.

The elements are sorted by "Categories". In the category "Variables", you can also set a "Filter" for the variable type, such as "Local variables", "Global variables", "Constants", etc.

"Structured view"	 : The elements are displayed in a structure tree. You can show/hide the columns "Type", "Address", and "Origin" by right-clicking the column title and selecting/clearing the column name in the dropdown list.  : The elements are displayed in a flat structure.
"Show documentation"	 : IndraLogic extends the dialog by the field "Documentation".
"Insert with arguments"	 : If elements include arguments (functions, for example), then IndraLogic also inserts with these arguments at the cursor position. Example: When inserting the function block <code>fb1</code> containing an input variable <code>fb1_in</code> and an output variable <code>fb1_out</code> "using arguments", it looks as follows: <code>fb1(fb1_in:= , fb1_out=>)</code>
"Insert with namespace prefix"	 : IndraLogic inserts the selected element with the appended namespace. In the case of library function blocks, the checkbox remains disabled if the requirement for a namespace has been defined in the library properties.



If you create objects with the same name in the same category, whether globally (in the "General module folder") or assigned to an application, only one entry is displayed in the input assistant, as the use of the regular call priority applies (application assignment before global).

Also refer to

- [""Properties" dialog" on page 673](#)
- [chapter 9.5 "Using input assistants" on page 107](#)

"Input assistant" dialog, tab "Text search"

This tab allows to search for specific objects. When starting to type a search string into the search field, the names of all objects are listed whose names include the search string. Double-click on an object to insert it at the current cursor position in the editor.

Filter	Limits the search to a specific variable category.

Declare variable



The "Declare variable" command is only available if the IndraLogic editor is open. For example for editing program organization units (POUs) or function blocks.

Command 'Declare variable'

Function: The command opens the dialog "Declare variable", which supports the declaration of a variable.

Call: IndraWorks menu **Edit**, optionally via the shortcut [Ctrl] + [F2].

Prerequisite: An object or a device of the project is open in the editor.

Due to the auto-declaration function, the dialog "Declare variable" is also displayed when the cursor is located in the implementation part of a POU in a line containing the name of an undeclared variable. Requirement: The command **Tools ► Options ► IndraLogic 2G** has to be selected and in the category "Intelligent coding", the option "Auto-declare unknown variables" has to be enabled.

"Declare variable" dialog

Validity range	Validity range of the variable that has not been declared yet. Example: VAR (default setting for local variables)
Name	Variable name that has not been declared yet. Example: bIsValid
Data type	Example: BOOL  Lists the standard data types.  Input assistant: Opens the "Input assistant" dialog Array assistant: Opens the "Array" dialog
Object	Object in which the new variable is declared. By default the object that you are currently editing is displayed. Example: fbA  : Lists objects in which the variable can be declared. If no objects are available for the selected Scope, the entry <create object> appears. If you select the entry <create object>, the dialog Add object appears for the generation of a suitable object.
Initialization value	Example: FALSE If no initialization value is entered, the variable is initialized automatically.  : Opens the Initialization value dialog. This procedure is recommended for the initialization of structured variables.

Address	Memory address of the application for the variable that has not been declared yet. Example: %IX1.0 Note: Only possible in case of the following validity ranges: • Local variable (VAR) • Global variable (VAR_GLOBAL) • Persistent variable (PERSISTENT)
Flags	Attribute keywords • CONSTANT: Keyword for one constant • RETAIN: Keyword for a remanent variable • PERSISTENT: Keyword for persistent variable (stricter than RETAIN). The selected attribute keyword is added to the variable declaration.
Comment	Example: New input In1 In the tabular declaration editor the comment entered is displayed in the column Comment, while in the textual declaration editor it is displayed above the variable declaration.
Apply changes by means of refactoring	: When closing the dialog, the variable is not declared yet but the Refactoring dialog opens. Changes can be edited here. The options is available in the following validity ranges: • Input variables (VAR_INPUT) • Output variable (VAR_OUTPUT) • VAR_IN_OUT variables (input variable and output variable)
OK	Variable is declared and is displayed in the declaration. Example: <pre>VAR RETAIN // New input In1 xIn1 AT %IX1.0: BOOL := FALSE; END_VAR</pre>

"Array" dialog

Specification of dimension and basic type	Definition of the field sizes ("dimension") by entering the lower and upper limits and the "basic type" of the array. Enter the basic type directly or via the dialogs "Input assistant" or "Array" when clicking on the  button.
Result	Display of the defined array.



IndraLogic only re-initializes variables if you have modified the initialization values of the variables.

"Initialization value" dialog

List of the variables with name ("expression"), "initialization value" and "data type". Modified initialization values are displayed in bold lettering.	
Input field below the list	Input of an initialization value for the selected variable(s).

Apply value to the selected lines	Change of the initialization value of the selected line(s) according to the value of the input field.
Reset selected lines to default values	Generation of standard initialization values.
OK	IndraLogic applies the initialization values to the dialog "Declare variable".

Also refer to

- [chapter 9.2.2 "Using the dialog "Declare variable"" on page 86](#)
- [chapter 9.5 "Using input assistants" on page 107](#)
- [chapter 9.10.2 "AT declaration" on page 125](#)
- [chapter 9.16 "Data persistence" on page 137](#)

18.2.2 Menu 'View'

Project Explorer

Symbol: 

Function: If the  setting is enabled, the IndraWorks project explorer is displayed on the programming interface.

Call: IndraWorks menu **View**

Library

Symbol: 

Function: If the  setting is enabled, the IndraWorks library explorer is displayed on the programming interface.

Call: IndraWorks menu **View**

Other windows

"Tools" command

Symbol: 

Function: The command opens the **Tools** view.

Call: Menu **View ► Other windows**

This command opens the Tools view for the open object. By default, this view is available for graphical editors and visualizations. It includes the graphical programming elements that you can drag into the editor.

Also refer to

- [chapter 17.1.2 "Common functionalities in graphical editors" on page 224](#)

"Cross reference list" command

Symbol: 

Function: The command opens the "Cross-reference list" view.

Call: Menu **View ► Other windows**

This view shows a list of cross references for a symbol in the project. The symbol can be a variable, a POU (program, function block, function), or a

user-specific data type (DUT). The cross reference list offer two basic types of searches:

- **Text search:** By specifying a symbol name, the cross-references of all symbols in the project are displayed with this names. If multiple symbols with the same name are found, the display can be limited to individual declarations by means of the context menu.
- **Declaration search:** The symbol can be selected by means of the input assistant or by specifying a qualified path (for example, `__POOL.POU.a`). Then only the occurrence locations of this symbol are displayed, even if there exist other symbols with the same name.

Input field	Symbol name (variable name, function block name, DUT name). Input options: • Selection of a declared symbol via the input assistance, interface  • Manual input of the symbol name. Triggering of the search by pressing the  button or the [Enter key]. For text search, use the placeholders "*" (any number of characters) or "?" (exactly one character) in combination with a partial string of a variable identifier. Use "%" to search for IEC addresses. Examples: "%MW8", "%M*". • Automatic if the name of a declared symbol is selected in an editor, or if the cursor is in the name field. An automatic search is also possible if the object is selected in the device tree or POU pool. Requirement: The option Automatically list selection in cross reference view has to be selected. Call via the IndraWorks menu Tools ► Options ► IndraLogic 2G ► Intelligent coding. <i>The following input is valid:</i> • Variable name, simple or fully qualified. Examples: "iVar", "PlcProg.iVar". • Function block name: Examples: "PlcProg", "myFB". • DUT name: Example: "mySTRUCT". • Strings combined with placeholders: "*" for any character or "?" for exactly one character. Example: "iVar*" affects iVar1, iVar_glob2, iVar45 etc. "iVar?" affects iVar1, iVar2, iVarX etc. but not iVar_glob2, iVar45 etc. • "%<IEC address>": IndraLogic searches for variables that are assigned to this address and direct memory access. Example: "%QB0", "%Q0 := 2".
	Find text <F4>: The search is performed.
	Take the declaration from the input assistance: The "Input assistant" dialog opens. A variable, a data type or a function block can be selected.
	Show source position of previous cross reference<Shift>+<F4>
	Show source position of next cross-reference<F4>
	Show source position of selected cross reference: The focus jumps to the location where the variable is declared. The respective declaration editor opens and the variable is selected there.

	Limit results to current declaration Available if multiple declarations are found for a symbol. Limits the display to the declaration that you have selected in the list.
	Print cross reference list: The default dialog opens for setting up a print job.
The cross references are displayed with the following information:	
"Symbol"	The result locations for the symbols (variables, POU, or DUTs) are grouped by declaration. The declaration occurrence comprises the root node and the occurrence locations in the project are indented below. The precise expression is displayed that has the symbol at the occurrence location. Example: If there is a global variable <code>i</code> in the project and a local declared variable <code>i</code> in a POU, then two root node entries will be listed after a text search for cross-references with the occurrences of the variable <code>i</code> below each.
"Block"	Function block name, DUT name, for example also a task name if there is a function block call in the task configuration.
"Variable"	Only the variable name, e.g. <code>iVar</code> .
"Access"	Type of access to the variable at the place of use: "Declaration" / "Read" / "Write" / "Call". Special case for pointers: An assignment type <code>p := ADR(var1)</code> is displayed as <code>write address</code> when searching for <code>var1</code> . Reason: Any write access to <code>p</code> is not displayed when searching for <code>var1</code> . Write access is also possible by means of pointer variables.
"Type"	Variable data type.
"Address"	IEC address if the variables are assigned. Example: "AT %QB0".
"Position"	Location of the occurrence in the POU editor, for example line number, network number, declaration section, or implementation section. Example: "line 1, column 1 (Impl)".
"Object"	POU name plus complete path of the location of use in brackets, if they are not in the "General module folder". Example: "PlcProg [Device: Logic: Application]"
"Comment"	Comment if available in the declaration of the variable.

Context menu commands in the cross-reference list

The search yields all result locations in the project and in included, uncompiled libraries.

Show source position: Opens the respective function block and highlights the location of use: The declaration for root entries and the respective location of use for subordinate entries. As an alternative, you can double-click on a line.

Limit results to current declaration Limits the display of results to the selected symbol declaration if multiple declarations are found.

Expand all: In the list, every single result location is shown.

Collapse all: In the list, only the root nodes of the result locations are shown.

Also refer to

- [chapter 9.12.1 "Using the cross-reference list to find places of use" on page 129](#)
- [chapter "Command 'Limit results to current declaration'" on page 699](#)
- [chapter "Command 'Expand all'" on page 584](#)
- [chapter "'Expand all' command" on page 584](#)

Command "Call tree"

Symbol: 

Function: The command opens the "Call tree" view.

Call: Context menu of a function block that can be called in the device node "Application" or in the "General module folder"

View "Call tree"

The call tree is available at all times, even before compiling the application. It is a static representation of the caller and the calls of the block that you specify explicitly. Therefore, the tree always contains two root nodes above the respective call order is displayed as successive indented entries. Recursive calls are detected quickly in this tree representation.

Example of a call tree (1) for the (2) PlcProg block:

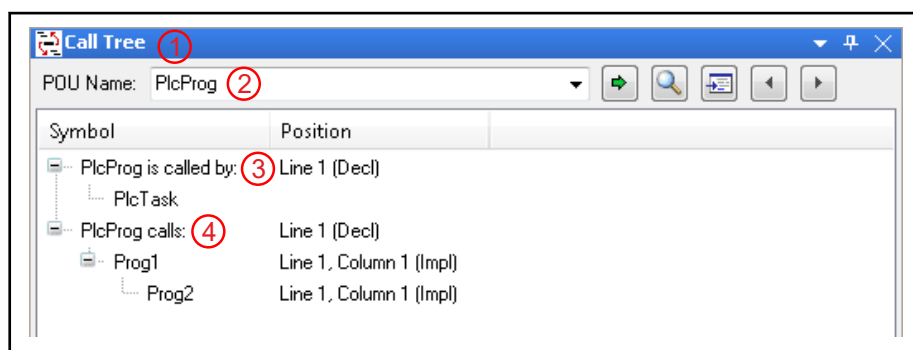


Fig. 18-39: "Call tree" dialog

- (3) Node "<BlockName> is called by:"
- (4) Node "<BlockName> calls"

Block name	Name of the program block; specified manually, by dragging from another view, or by means of the button . The list box includes the last specified block names.
Toolbar and keyboard usage	
Find block	IndraLogic searches for the block specified in "Block name" and displays its caller and calls.
Use function block from the input assistant	The "Input assistant" dialog opens for selecting a block call or instance call. The call tree is refreshed automatically after the selection.
Show source code position of the selected block	IndraLogic jumps to the occurrence location of the block in the source code of the program.
[F4] Show source code position of the next block [Shift] + [F4]: Show source code position of the previous block	The selection in the call tree jumps to the next or previous block in the call structure. At the same time, the associated source code position is open in the respective editor. Note: Double-clicking an entry in the call tree also opens the associated source code position.
Display of the call tree:	

Symbol	"<BlockName> is called by": The call order for the function block is displayed below this node. The bottom entry in this tree structure shows the start of the calls. "<BlockName> calls": The calls from this function block are displayed below this node. The bottom entry in this tree structure shows the end of the call chain.
Position	For the root node in the call tree: Line numbers of the declaration ("Decl") of the block. For the caller or calls below the root node: Line number, column number, and network number of the position, depending on the implementation language.
Context menu for the entry selected in the tree:	
Collapse all	The expanded entries in the call tree are collapsed, except for the two root nodes.
Show source code	IndraLogic jumps to the occurrence location of the block in the source code of the program.
Set as new root node	The entry selected in the call tree is displayed in "Block name". The tree is refreshed automatically for the new root nodes.



The "Call stack" view is provided for immediate information when stepping through a program, as opposed to the static call tree that provides call information about a block. The call stack always shows the full call path of the current position that is reached.

See:

- [chapter 12.3 "Stepping a program" on page 181](#)
- [chapter "Command 'Call list'" on page 627](#)

Command "Browse cross-references in classic view"

Symbol:

Function: This command opens the "Classic cross-reference list" view.

Call: Menu **View** ► **Other windows** after applying from **Tools** ► **Adjust**, command category "Source text navigation"

The view corresponds to the "Cross-reference list" view before IndraLogic 2G, 3.5 SP6.

The search yields all result locations in the project and in included, uncompiled libraries.

"Element properties" command

Symbol:

Function: The command opens the "Element properties" view.

Call: Menu **View** ► **Other windows**

This command opens the properties view for the open object. This view contains only contents for a few objects, for example visualization and POU (SFC).

The properties are displayed in a structured table. You change the property values by clicking into the value fields. Filter or sort the properties view.

'Memory' command

Symbol: 

Function: The command opens the "Memory" view.

Call: Menu **View** ► **Other windows** ► **Memory**

Prerequisite: One application is loaded and in online mode.

View "Memory view"

"Application"	Selection of the application for the displayed memory view. You must be logged in to this application on the control. It does not have to be the "active application".
"Range"	"Absolute": A memory address is assigned directly and completely. The address is displayed in the "Address" input field. "Area <i>": Memory areas of the control starting with Area 0. Memory areas that are reserved exclusively for code are not displayed.
"Address"	Absolute start address of the core dump Requirement: The value "Absolute" is selected in the input field "Range".
"Offset"	Address offset (in bytes) for the selected memory area, for example 0x0200, 16#0200, or 512 (decimal number) Requirement: A memory area is selected in "Area", for example "Area 0". IndraLogic lets you select all memory areas currently in use. Memory areas that are reserved exclusively for code are not displayed.
	Look up the address for a variable: An input support to select an IEC variable is displayed. When selecting more than one variable, IndraLogic reserves the starting address with the variable address.
	Load/refresh memory view
	Show previous memory segment: Navigate to the previous memory segment
	Show next memory segment: Navigate to the next memory segment
	Caution: IndraLogic does not check that the changes are permitted. Careless changes can crash the application! Upload changes to the PLC: IndraLogic transfers the new data to the control. Requirement: You have overwritten at least one byte in the memory view.
	Save core dump to file: The dialog "Save contents to memory file". Select a save location.
"Columns"	Number of columns for the hexadecimal display of the core dump. Two bytes are displayed per column. For "Auto", the number of columns is adjusted to fit the window size. The data is displayed as text on the right in the window.

"Full screen" command

Symbol: , shortcut: <Ctrl> + <Alt> + <F>

Function: The command switches the IndraWorks display to full screen mode.

Call: **View** menu

Choosing this command displays the main window of the IndraWorks user interface in full-screen mode. You can return to the previous setting by choosing the command again or with the keyboard shortcut <Ctrl>+<Shift>+<F>.

"Properties" command

Symbol: 

Function: The command opens the properties of the currently selected object.

Call: **View** menu

Also refer to

- [chapter 18.3.6 ""Properties" dialog" on page 704](#)

"Startup screen" command

Symbol: 

Function: The command opens the **Startup screen** view.

Call: **View** menu

The view includes some basic commands and a list of recently opened projects.

Select "Do not show at start" to disable the startup screen until canceled.

18.2.3 Menu 'Project'

"Project settings" command

Symbol: 

Function: The command opens the "Project settings" dialog.

Call: Double-click on the "Project settings" object in the "General module folder".

Prerequisite: A project is open.

Also refer to

- [chapter 2.5.2 "Applying project settings" on page 13](#)
- [chapter 18.3.7 ""Project settings" dialog" on page 714](#)
- [chapter 18.1.19 ""Project settings" object" on page 545](#)

"User management" command

In IndraWorks 13VRS, the IndraWorks user management is updated.

The new user management provides the following advantages:

- Protected PLC libraries can be created
- Debugging is supported in protected libraries
- it is an integral part of the IndraWorks project. Both are archived together
- Supports the Safety objects

When opening a IndraWorks project < 13VRS, these and an existing protection is detected and the following messages are output:

Message when opening an IndraWorks project < 13VRS

The project has to be converted into a new format, as it is currently in an old compatibility version.

The project can then not be opened anymore in the old compatibility version!

Do you want to convert the project now?

Message if this project was protected by the older user management

Object protected with the user management from IndraWorks 12VRS were found in the project.

The protection is removed when opening the project.

Do you want to continue?



All user management information is removed when opening the project.

Configuring the user management

Function: User management

Call: **Project ► User management**. The following commands are provided in this menu:



Fig. 18-40: "User management" menu

- [chapter 18.1.28 "Access control" object](#) on page 571

"Log in user" command

Symbol:

The **Log in user** command opens the **Login** dialog. Log into this dialog using a defined user account in a project or in a library.

Login via user account means that access rights to project objects are assigned that are assigned to a group the user is a member of.

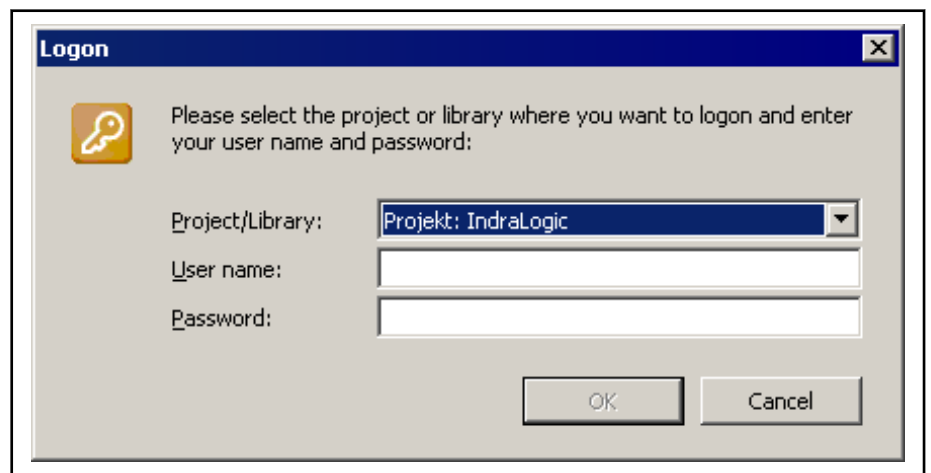


Fig. 18-41: "Log on" dialog

"Login" dialog if actions are invalid

If another user is already logged in, this user will automatically be logged out by the new login.

When you are logged into a project or a library and you try to execute an action without the respective right, the following login dialog opens automatically allowing to log in using a different user account with the respective rights:

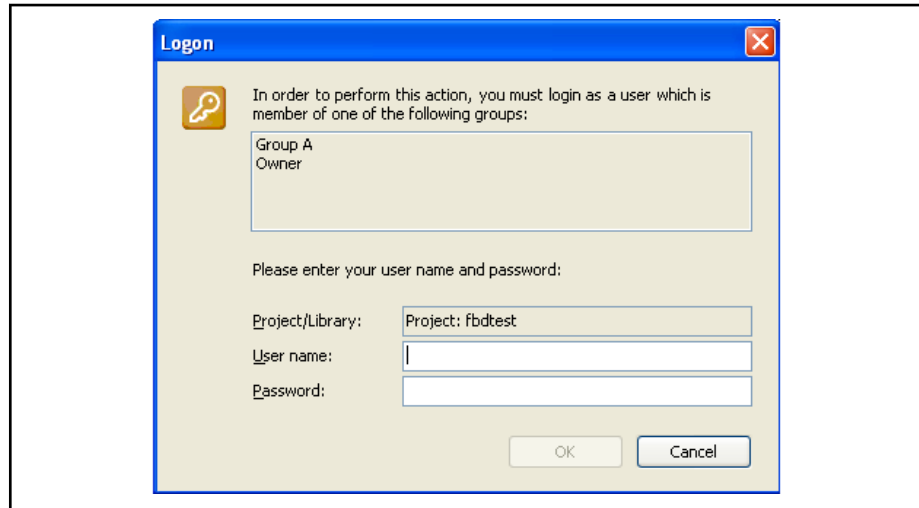


Fig. 18-42: "Login" dialog if an action is invalid

For information purposes, the upper part of the dialog display all groups with the rights required for the action.

If a user account can be used for one of these groups, enter the respective login details (user name and password) and execute the action. The toolbar always shows the user currently logged into the project.

The toolbar always shows the user currently logged into the project.



Fig. 18-43: Example: Status bar

"Log out user" command

Symbol:

The command **User management** logs off the user currently logged into the project or the library. A corresponding message is output only if currently no user is logged in. Otherwise, no dialog and no message appears.

If the user is currently logged in to several projects or integrated libraries (not necessarily using the same user account), the following **logout dialog** is opened.

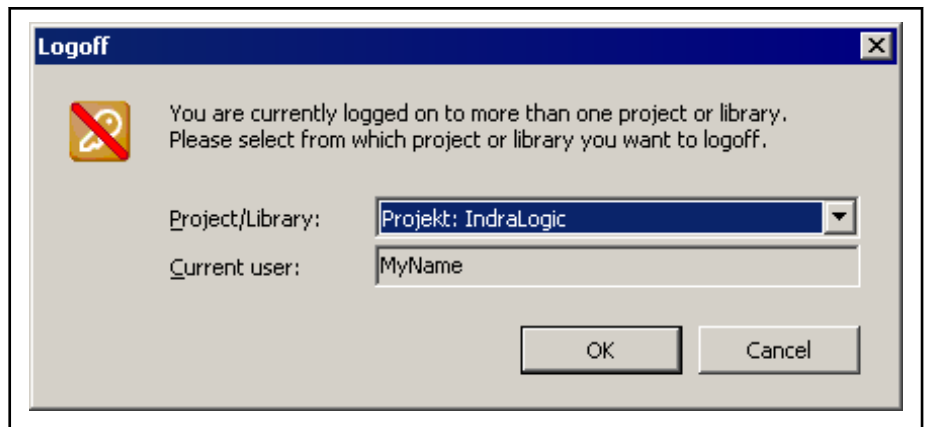


Fig. 18-44: "Logout" dialog

Via the selection list in the **Project/Library** field, you can now select the project or the library from which the **Current user** is to be logged off.

"Users und groups" command

Symbol:

The **Users and groups** command opens the **Users and groups** dialog. The dialog consists of three tabs: Users, Groups and Settings:

Users The created groups and their memberships to the groups are displayed.

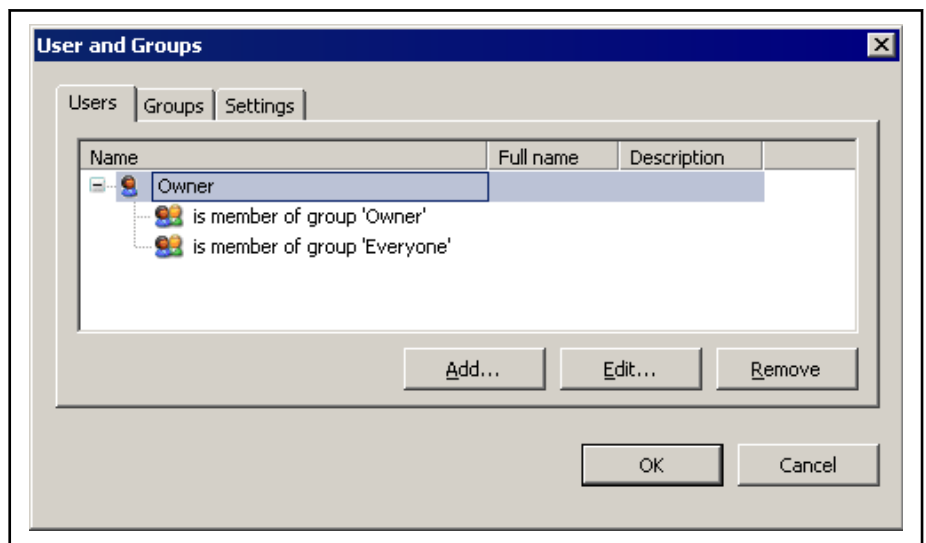


Fig. 18-45: "Users" dialog from "Users and groups"

Groups The created groups and their users are displayed.

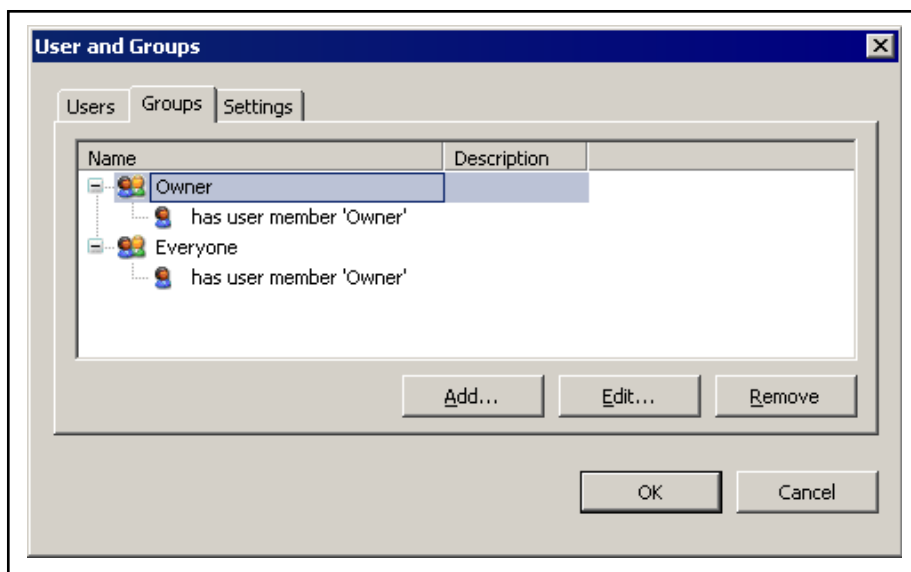


Fig. 18-46: "Groups" from "Users and groups" dialog

Settings

The number of unsuccessful logins and the automatic logout after inactivity can be configured.

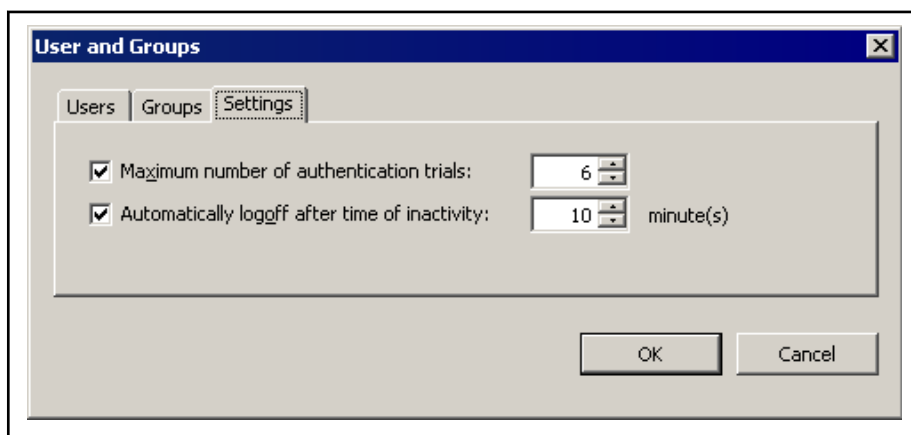


Fig. 18-47: "Setting" from "Users and groups" dialog

Permission to access project objects with a specific action is always given to groups only (not to individual user accounts). Thus, every has to be a member of a group to work at a project

Note the following when creating user accounts and user groups:

- There is automatically always a group **"Everyone"** and by default every user and every other group is initially a member of this group. Thus each user account is automatically equipped with at least the defined **standard** rights. The group "Everyone" can be renamed, but not deleted. No members from this group can be deleted.
- There is always automatically a group **Owner** containing a user **Owner**. More users can be added to this group and users can also be removed. However, there has to be at least one member. This group can also not be deleted and **has all access rights at all times**. This prevents a project from being rendered unusable by denying all access rights to all groups. Group and user "Owner" can be renamed.
- If the programming system or a project is restarted, no user is initially logged in to the project.

However, the user can then log in via a certain user account with user name and password to obtain certain access rights.

- **Each project has its own user management!** Therefore, in order to obtain certain access rights to a library integrated into the project, for example, the user has to explicitly and separately log into this library. Users and groups defined in different projects are not the same, even if they have the same names.

“Rights” command

The **Rights** command opens the **Rights** dialog used to configure the access rights to objects or the rights to execute specific actions in the project.

At first, only the “Owner” (group “Owner”) has the right to change the current user, group and rights configuration in a new project. To assign this right to another group, log into the project with the user name “Owner” and an empty password.



Note that all settings in the “Rights” dialog are applied immediately.

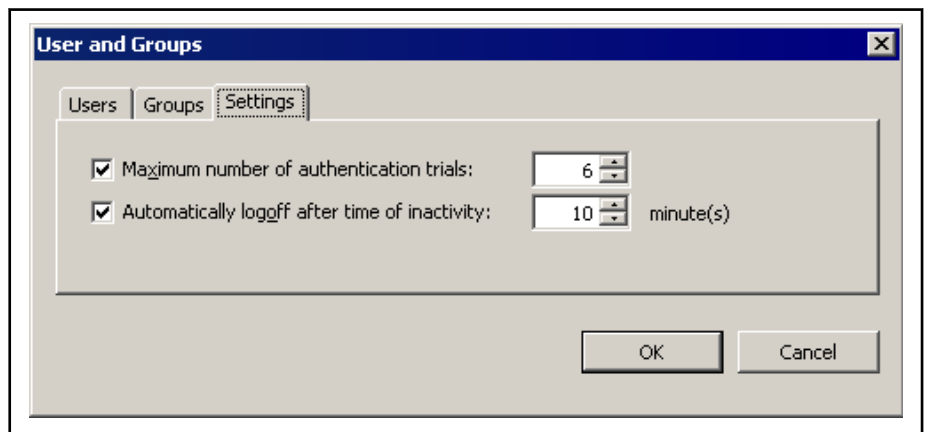


Fig. 18-48: “Rights” menu from the user management

The **Actions** window in the left section of the dialog displays all possible rights, that means all actions that can be executed on objects of the project. The tree is structured as follows:

- On top level: The name of some categories used to structure the tree as per the right allocation to execute **commands**, to modify **user settings and group settings**, to create **object types** and to view, modify, remove and handle child objects of **project objects**. **Project objects** always refer to the objects currently created in the project. This display changes when adding and deleting objects.
- The entries for the executable **actions** are located below the category entries. These nodes are also only used to structure the tree.

Possible actions:

- Executing (a menu command)
- Create (creating a new object in the project)
- Adds or removes child objects (of existing objects)
- Edit (an object in the editor or user settings and group settings in the respective dialog)

-  Remove (deleting or cutting an object)
-  View (opening the view of an object in the editor)
- ➡ There are possible "**Targets**" of the action (such as the individual project objects or menu commands) below each "Actions" node. These entries can be individually select to assign rights.

The **Rights** window in the right section of the dialog displays a list of all currently defined user groups (except for the "Owner" group) and a toolbar to configure the rights of a group.

Highlight the group for which you currently wish to define the rights.

On the left of the group name, one of the symbols explained below, indicates the right applying for the currently selected "target" of an action in the "Actions" window. To change a right, click on "Target" entry in the "Actions" window. Right symbols:

-  The currently extended action  in the "Actions" window on the currently selected target(s) ➡ is **granted** for the currently selected group.
-  The currently extended action  in the "Actions" window on the currently selected target(s) ➡ is **Denied** for the currently selected group.
- : The currently extended action  in the "Actions" window on the currently selected target(s) ➡ was not explicitly granted, but it is **Granted by default**, as it was for example granted for the "Father object". Example: This right was assigned to the group for the "myplc" object. Thus, it is also the default right for the object "myplc.pb_1". "Granted" is the default setting for all rights which were not explicitly configured.
- : The currently extended action  in the "Actions" window on the currently selected target(s) ➡ was not explicitly denied, but it is **Denied by default**, as it was for example denied for the "Father object".
- If multiple actions are selected in the "Actions" window and the group does not have the same right for these actions, no symbol is displayed.

To **modify the rights for one group**, select the corresponding action  and the target(s) ➡ in the "Actions" window and the corresponding button in the toolbar of the "Rights" window:

-  **Grant** Explicit granting.
-  **Deny** Explicit denying.
-  **Clear** The right is reset to default.

"PLC compiler version" command

Function: In IndraWorks version <13V06, the PLC compiler version is linked to the IndraWorks compatibility version.

The PLC compiler version can additionally be selected in the IndraWorks compatibility version 13V06 and higher. This allows to adjust the IndraWorks compatibility mode without changing the PLC compilation. Even after changing the compatibility mode, login without new download to the device is possible.

Call: Menu **Project** ► **PLC compiler version...**

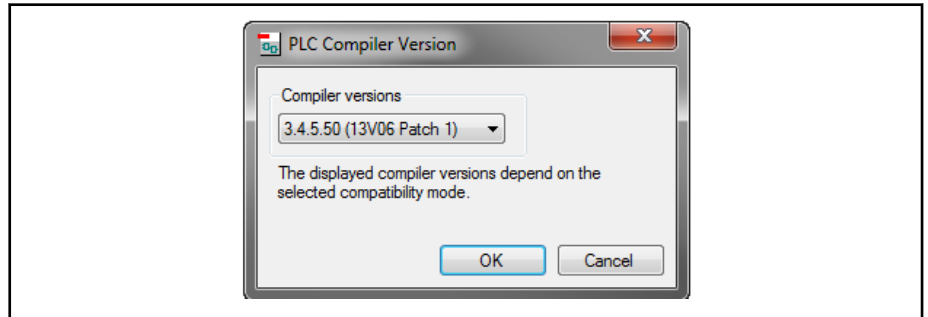


Fig. 18-49: Dialog of the PLC compiler version

Specify the PLC compiler version to be used when executing the "Create" command or while downloading for project compilation in the dialog.

The corresponding IndraWorks compatibility version is located behind the brackets of the PLC compiler version.

The latest Plc compiler version that can be set corresponds to the current compatibility mode of the project.

When creating a project again, the PLC compiler version is set to the corresponding IndraWorks compatibility version by default.

Command 'PLC project comparison'

Function: The "PLC project comparison" command allows to compare the currently loaded project to a second project and to merge part of the second project to the first one if required.

Call: IndraWorks menu **Project** ► **PLC project comparison**.

If the command is enabled, a selection window is displayed to specify the second project to be compared.

Both projects are compared to each other.

The comparison statistics outputs possible differences with regard to the boundary condition of the selected comparison options (**Tools** ► **Options** ► **Comparison**, also refer to IndraWorks, Options)

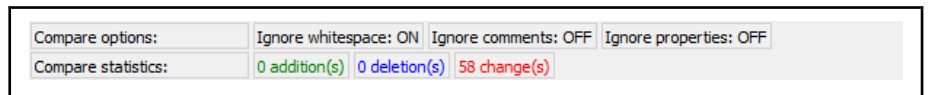
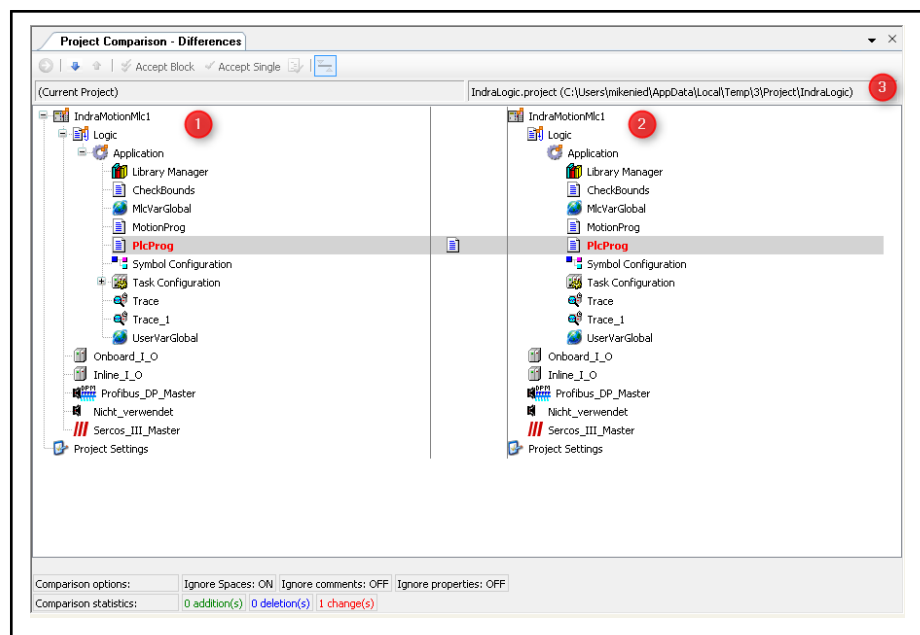


Fig. 18-50: Compare statistics

If the projects differ, click on one of the objects to enable the actual comparison.



- (1) Currently loaded project
- (2) Second project for comparison
- (3) Position of the object selected in the Navigator

Fig. 18-51:

Click on one of the objects and the toolbar of the window is enabled:

-  : Allow to switch between the “project differences” view and the “Internal differences of the selected object pair”
-  : Go to the next/previous difference
-  Accept Block  Accept Single : The complete block or only the highlighted line can be applied.

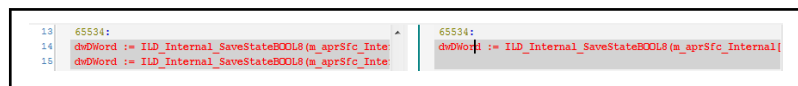


Fig. 18-52: Initial situation

Line 14 on the left and right differ in their content, line 15 was deleted.



Fig. 18-53: Accept block

Line 14 is highlighted on the right, but all lines (14 and 15) of the block are prompted for application from right to left. For the actual application, press <Apply>.



Fig. 18-54: Applying individually

Line 14 is highlighted on the right. Only this line of the block is prompted for application from right to left. For the actual application, press <Apply>.

- : Folders, access rights and properties of the object on the right are applied.
- : Showing ranges differently.



Fig. 18-55: Representation of different contents in the lines 14/15

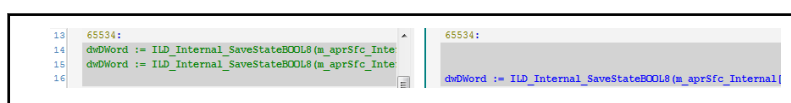


Fig. 18-56: Showing different operations, green (14/15) inserted, blue (16) deleted

- : Comparison status and application status



Fig. 18-57: The contents of the objects differ

- : Comparison status and application status

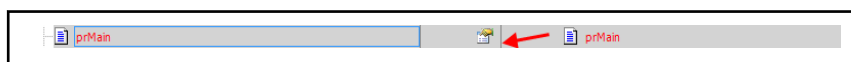


Fig. 18-58: The properties of the objects differ

“Add” command

Function: The command opens a submenu with objects. This submenu contains all objects that can be inserted with regard to the current position in the Project Explorer.

Call: Context menu.

Prerequisite: If IndraLogic should insert the object into the Project Explorer, select an already existing object below which the new one should be inserted and indented.

If IndraLogic should insert the object into the "General module folder", set the focus to the folder.

Also refer to

- [chapter 2.2 "Adding objects" on page 8](#)
- [chapter 2 "Creating and configuring projects" on page 7](#)

“Add folder” command

Symbol: 

Function: The command opens a dialog to define a new folder in the Project Explorer.

Call: Context menu



The device nodes and device objects cannot be structured by self-created folders.

The command inserts the folder below the currently selected object into the Project Explorer. If no object is selected, IndraLogic inserts the folder on the top on the tree directly below the node.

Also refer to

- [chapter 2.2 "Adding objects" on page 8](#)

Command "Display PLC objects of an external IndraWorks project"

Symbol:

Call: Menu **Project ► Display PLC objects of an external IndraLogic project**

Function: The command allows to open an external project and to copy PLC components from its content.

Click on the command and the standard dialog to select a project opens. The file filter settings ensure that either an IndraWorks project (storage location of the directory *.iwp or single-file storage location *.xiwp) or (directly) an IndraLogic project (Indralogic.project) can be selected. Navigate to the corresponding file for all project types. Then, select them and press "Open".

To directly open an IndraLogic project file for example, navigate to the respective IndraLogic project, e.g. **First steps Motion ► IndraLogic ► IndraLogic.project**.

Select for example the file **First steps Motion ► First steps Motion.iwp** to open an IndraWorks project located in the directory structure .

To open an IndraWorks project in a single-file storage location, the respective file **<ProjectName>.xiwp** has to be selected.

For all project types, a dialog appears to select the objects to be opened:

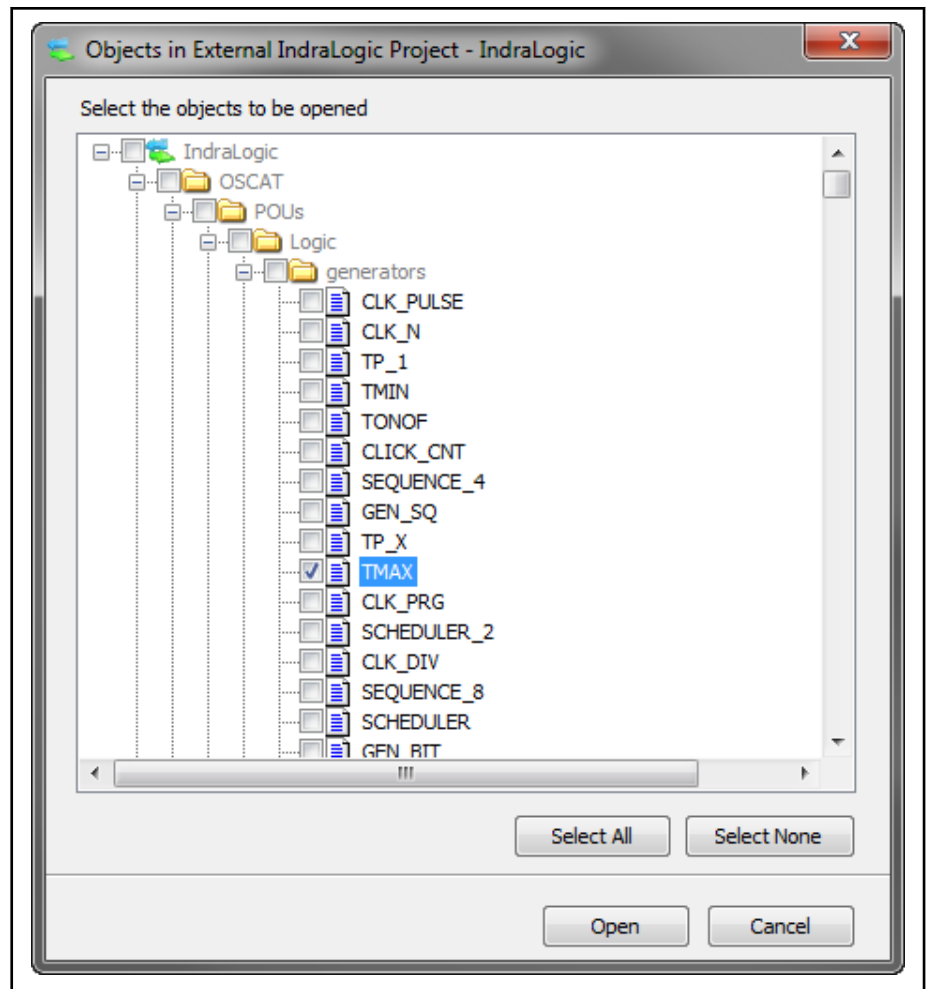


Fig. 18-59: Dialog: PLC object selection

Objects that can be selected (i.e. displayed) have black font, all others are shown in gray font.

Use <Select all> or <Select none> to mark all objects for opening or remove their marking.

Clicking the button in front of a selectable object with the mouse marks individual objects for opening.

If the user presses the <Open> button, the selected objects are opened for read access. The user can now e.g. mark text passages in these objects and copy them to objects of the open project.

The "External:" entry in the title of the document window shows the user that the opened object comes from an external project.

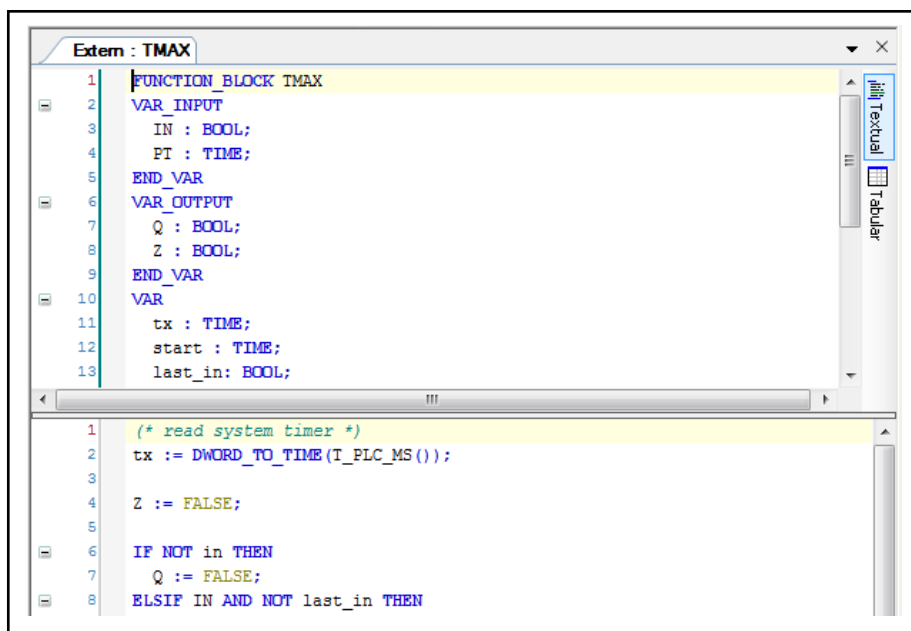


Fig. 18-60: Opened external PLC object

“Edit object (offline)” command

Function: The command opens the object offline in its editor.

Call: Context menu of an offline editable object

Prerequisite: The application is in online mode. An object is selected in the Project Explorer or in the "General module folder".

The object can thus also be edited in online mode.

Apply the changes to the control with **Debug ► Online change** or **Debug ► Load** from the context menu.

Also refer to

- [chapter "Command 'Online change'" on page 619](#)
- [chapter "Command 'Load'" on page 618](#)

Command 'Set active application'



Online actions apply only to the active application.

Examples:

- Breakpoints
- Trace
- Force

Function	The command sets the selected application as the "active application".
Call	Open the application object Set active application via the context menu of the application object.
Prerequisites	• The project is provided with at least two applications • The selected application is not active
Representation in the user interface	The name of an active application is displayed in bold typeface in the Project Explorer. Additionally, the active application is displayed in the IndraLogic symbol bar.

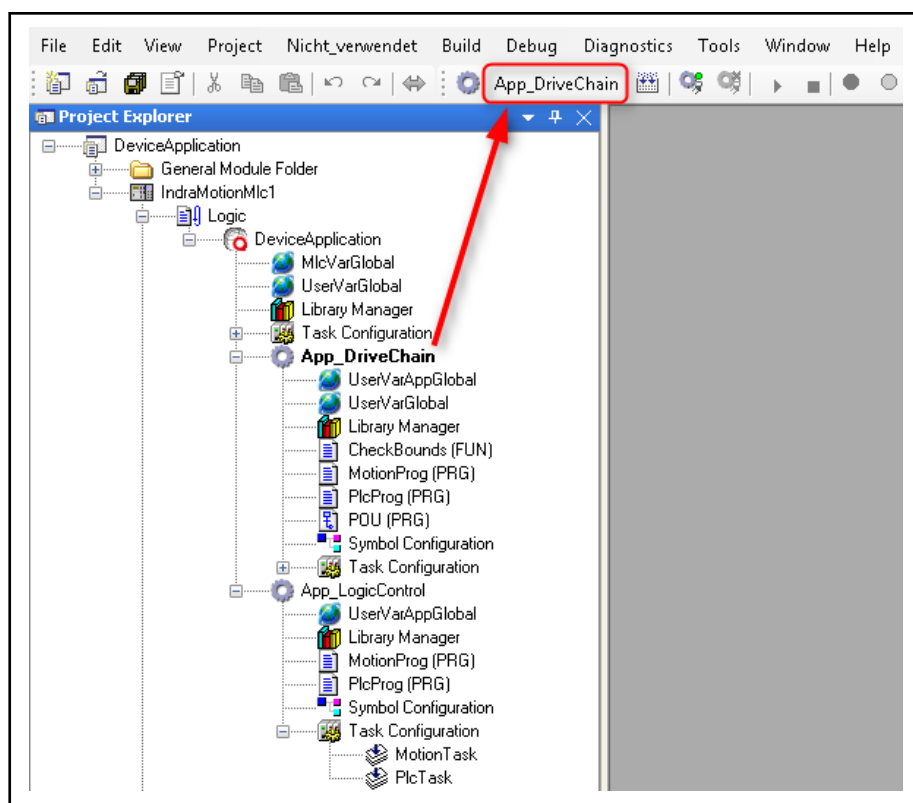


Fig. 18-61: Representation of an active application in the user interface

"Edit I/O mapping" command

Function: The command opens the **Edit I/O mapping** dialog. This displays all the I/O "mappings" of the currently selected device object plus the I/O mappings of all further device objects that are inserted in the Project Explorer under this object.

Call: Context menu of a device object in the Project Explorer.

"Edit I/O mapping" dialog

You can edit the I/O mapping in this dialog in exactly the same way as in the dialog **I/O mapping** of the individual device editors. The respective other dialogs are directly updated accordingly.

Search	Input field for a character string to search for the mapping table. The search results are marked in yellow.
Filter	<i>Selection list with filters for the I/O assignments shown in the mapping table:</i> • Show all • Show only outputs • Show only inputs • Show only mapped variables • Show only mappings to existing variables • Show only mappings to new variables • Show only unmapped variables

In the context menu you will find among other things the following commands:

Export I/O mapping to CSV: stores the mappings of a device and its sub-devices in an external file. To do this, select the device in the Project Explorer or in the mapping list.

Import I/O mapping from CSV: inserts mappings from a file created beforehand by export.

Also refer to

- [chapter 8 "Configuring an I/O link" on page 69](#)
- [chapter ""Export I/O mapping to CSV" command" on page 610](#)
- [chapter "Command 'Import I/O mapping from CSV'" on page 610](#)

Command 'Import I/O mapping from CSV'

Function: this command opens the standard dialog for searching for a file in the local file system. The filter is set to file format `csv`, for the selection of a file to which the I/O mapping configuration of a device was previously exported. IndraLogic loads the configuration back into the current project.

Call: Context menu of the **I/O mapping** tab in the device editor of a device.

Prerequisite: A project with a device with I/O mapping configuration is open. The device matches the exported `csv` file.



Edit the `csv` file externally before import. Note the following: The entries of the file to the I/O mapping of the device are assigned via the device name and the parameter name. Parameter names that are not unique are numbered sequentially in this file `@<n>`.

Fields without contents in the `csv` file are ignored during the import. To remove an existing entry in the I/O mapping by importing, add a space in the respective field in the `csv` file.

Also refer to

- [chapter ""Export I/O mapping to CSV" command" on page 610](#)
- [chapter 8.3 "Configuring devices and I/O mapping" on page 73](#)

"Export I/O mapping to CSV" command

Function: this command opens the standard dialog for saving a file in the local file system. The filter is set to the file format `csv`. After specifying a name and a location, IndraLogic stores the I/O mapping configuration in a `csv` file.

Call: context menu of the **I/O mapping** tab in the device editor of a device.

Prerequisite: A device object with an I/O mapping configuration is selected in the Project Explorer.



Parameter names that are not unique are numbered sequentially in this file `@<n>`.

Also refer to

- [chapter "Command 'Import I/O mapping from CSV'" on page 610](#)
- [chapter 8.3 "Configuring devices and I/O mapping" on page 73](#)

"Replace Global Variable List (GVL) in XML format" command

A Global Variable List in the IndraLogic project should be exportable and re-importable as `xml` to be readable and compilable.

Execution It is called via the context menu of a selected Global Variable List

- Storing data as XML file...
- Data transfer from XML file...

The file storage location for export and import is selected via standard Windows mechanism.

"*.gvl.xml" is specified as file extension.

Format description

This data is saved in xml format.

The <GVL> tag is created behind the <File> tag.

The <Interface> tag includes the actual content of the "Global Variable List" in the same format as shown in the editor in IndraWorks.

XML file structure

```
<file>
  <GVL>
    <interface>VAR_GLOBAL CONSTANT
      (* define all possible axis,
        set to maximum axis index to preserve resources *)
      // set to a maximum of 64 for the L65
      uiMAX_AXIS_INDEX:   UINT      := INT_TO_UINT(MB_MAX_AXIS_NUMBER);
    END_VAR
  </interface>
</GVL>
<path>Application.UserConstGlobal</path>
</file>
```

18.2.4 Menu 'Create'

The menu item includes command for compilation, code generation and cleaning. The first three command are default command. Assign further commands via **Tools** ► **Customize**, tab **Commands**, category "Create".

"Build" command

Symbol: , shortcut: <Shift>+<F11>

Function: The command starts the build process for the active application.

Call: **Create** menu

During the compilation, IndraLogic syntactically validates of all objects in the application. However, code is not generated like at login to the target system or download of the application.

The compile process is always performed automatically when you login with a changed program.

When the check is complete, IndraLogic displays any error messages or warnings in the messages view (**Build** category).

If the program has not been changed since it was compiled without errors the last time, then it is not recompiled. The message "The application is current" is displayed. If the syntactic validation is repeated, then you must execute the **Rebuild** command.

Also refer to

- [chapter ""Rebuild" command" on page 612](#)

"Generate code" command

Function: The command starts the code generation for the active application.

Call: **Create** menu

When generating code with this command, code is generated as when "downloading the application to the control", but the code is not transferred to the control. The source code is checked further.

Thus, the code can be checked for bugs not detected by the compiler and for fixing any bugs before the code is used in online mode.

Also refer to

- [chapter 11.2 "Generating application code" on page 175](#)

"Clean" command

Function: The command deletes the build information for the active application.

Call: **Create** menu

During the last download, the build information was created in the local file system and saved to a file "*.compileinfo" (C:\ProgramData\IW-Projects\Project\IndraLogic\IndraLogic.IndraMotionMlc1.Application.*.compileinfo).

After a cleaning process, an online change is no longer possible for the affected application. The application has to be downloaded completely to the control again.

Also refer to

- [chapter ""Clean all" command" on page 613](#)

"Rebuild" command

Function: The command starts the build process for the active application, even if the last build contained errors.

Call: Enable the command via **Tools ► Customize ► Commands ► Create category**.

Also refer to

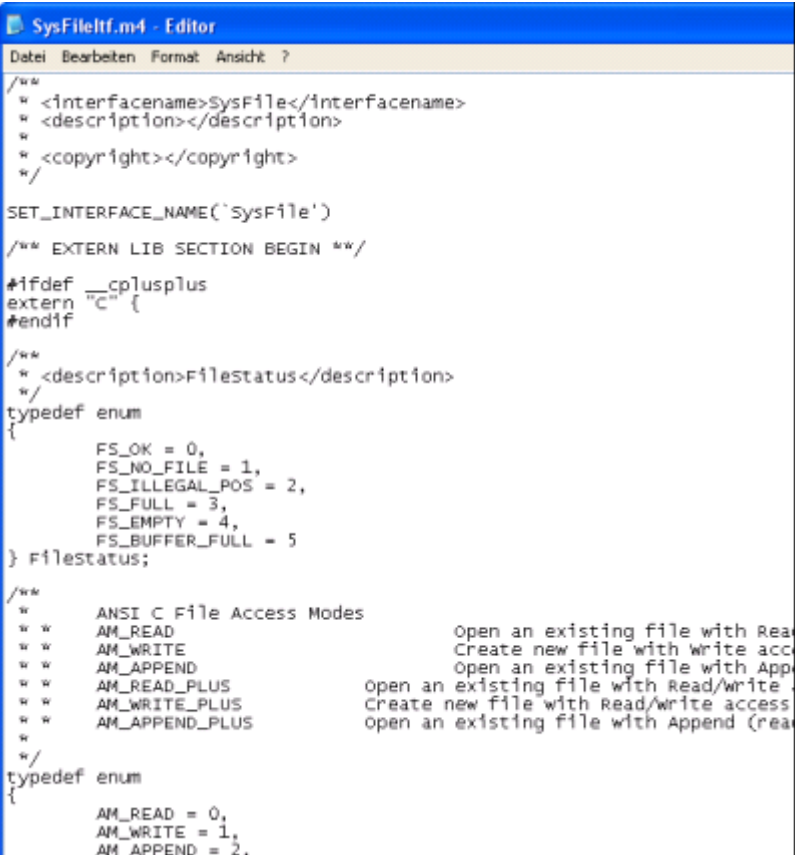
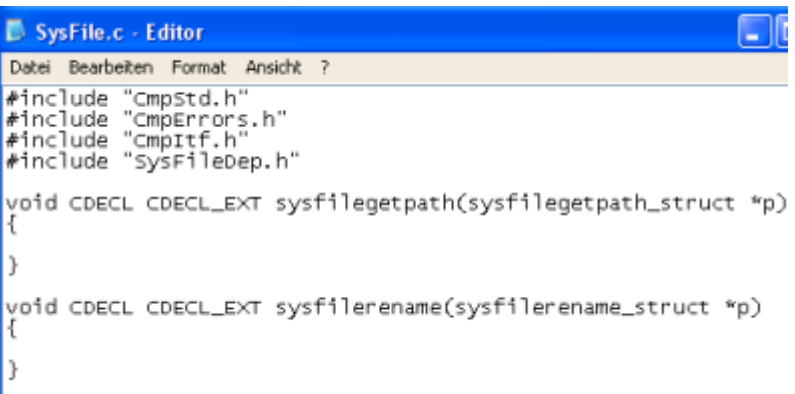
- [chapter ""Build" command" on page 611](#)

"Generate runtime system files" command

Function: The command generates a C stub file and an M4 interface file from the current library project. These files are used as the basis for creating an external library file.

Call: Enable the command via **Tools ► Customize ► Commands ► Create category**.

The command opens the **Generate Files For Runtime System** dialog.

Output directory	Directory where IndraLogic creates the runtime system files. Click the button  to open the default dialog for browsing the file system.
M4 interface file	: Interface file <projectname>Itf.m4 with definitions. Example of m4 file:  <pre> SysFileItf.m4 - Editor Datei Bearbeiten Format Ansicht ? /** * <interfacename>SysFile</interfacename> * <description></description> * <copyright></copyright> */ SET_INTERFACE_NAME('SysFile') /** EXTERN LIB SECTION BEGIN **/ #ifdef _cplusplus extern "C" { #endif /** * <description>Filestatus</description> */ typedef enum { FS_OK = 0, FS_NO_FILE = 1, FS_ILLEGAL_POS = 2, FS_FULL = 3, FS_EMPTY = 4, FS_BUFFER_FULL = 5 } Filestatus; /** * ANSI C File Access Modes * AM_READ Open an existing file with Read * AM_WRITE Create new file with Write acc * AM_APPEND Open an existing file with App * AM_READ_PLUS Open an existing file with Read/write * AM_WRITE_PLUS Create new file with Read/write access * AM_APPEND_PLUS Open an existing file with Append (read */ typedef enum { AM_READ = 0, AM_WRITE = 1, AM_APPEND = 2, </pre>
C-stub file:	: Stub file for reprogramming the library in C. Example of stub file:  <pre> SysFile.c - Editor Datei Bearbeiten Format Ansicht ? #include "CmpStd.h" #include "CmpErrors.h" #include "CmpItf.h" #include "SysFileDep.h" void CDECL CDECL_EXT sysfilegetpath(sysfilegetpath_struct *p) { } void CDECL CDECL_EXT sysfilerename(sysfilerename_struct *p) { } void CDECL CDECL_EXT sysfiledelete(sysfiledelete_struct *p) { } </pre>

Tab. 18-11: Which files do you want to create?

"Clean all" command

Function: The command deletes the build information for all applications in the project.

Call: Enable the command via **Tools ▶ Customize ▶ Commands ▶ Create category**.

During the last download, the build information was created in the local file system and saved to a file `"*.compileinfo"` (C:\ProgramData\IW-Projects\X\Project\IndraLogic\IndraLogic.IndraMotionMlc1.Application.*.compileinfo).

With this command, a download is required before the next login. An online change is no longer possible. As compared to the **Clean** command (only the active application), IndraLogic regenerates the language model for all objects, which is very time-consuming.



Reconsider carefully whether executing this command is really necessary. If you only want to rebuild and download the active application, then execute the **Clean** command.

Also refer to

- [chapter "Clean" command" on page 612](#)

18.2.5 Menu 'Debug'

Menu 'Login'

Symbol



Function

The command connects the application to the target system (control to simulated device) and starts the online mode.

Call

Menu **Debugging**, or via the via context menu of the **Application** object or via the shortcut: <Alt> + <F8>.

Prerequisite

The application contains no errors and the communication settings are configured.

If the communication settings are incorrect, a dialog is displayed. You can switch directly to the **Communication settings** of the control.

Click on **Login** in the "Debug" menu and the currently active application is connected to the target system.

If you choose this command from the shortcut menu (right-click) while an application is selected in the Project Explorer, then that application is logged in, even if it is not set as the "active application".

If an online user management is configured on the target device, then you are prompted for user data when you log in. The **Device user login** dialog opens.

CAUTION

Check options to access the control

Controls can under no circumstances be accessible from the Internet or untrustworthy networks!

Specifically, the programming ports of the control can under no circumstances be unprotected and accessible from the Internet (usually UDP ports 1740 - 1743 and TCP ports 1217 and 11740, or control-specific ports)!

However, if access from the Internet must be possible, then a secure procedure must be selected for connecting to the control (e.g., VPN).



If a safety controller is configured below a controller, then this command can interrupt the communication connections temporarily.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to chapter [chapter 10.5 "Subordinate safety control" on page 172](#)

Possible situations when logging in:

- A later version of the device description (than in the project) is on the control. A warning prompt is displayed with the option to cancel the process.
- The application does not exist yet on the control: You are prompted to confirm the download.
- The application is already on the control and has not been changed since the last download. The login continues without any more prompts.
- The application exists on the control, but it has been changed since the last download.

You are prompted to select one of the following options:

- Login with online change
- Log in with download
- Login without any change

At this point, there is also the option of updating the boot application on the control.

- There is an unknown version of the application on the control. You are prompted whether IndraLogic should replace them.
- There is a version of the application on the control and it is running. Prompts you whether IndraLogic should still log in and overwrite the currently running application.
- The application on the control is currently stopped at a breakpoint.

You have changed the program and logged out:

If you have made an online change or download, then IndraLogic warns you that the PLC will be halted completely. This also happens if several tasks exist and the breakpoint affects only one of them.

Click **Details** in the dialogs above to open the **Application Information** dialog.

Refer to

- [chapter "Command 'Online change'" on page 619](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)

"Application information" dialog (details)

This dialog provides information on the application changed in the development system and its previous version currently on the control. There are two tabs:

- **Application information:** The application properties of the **Application in the IDE** (Integrated Development Environment) are compared with those of the **Application in the PLC**: Project name, last modification, IDE

version, author, description. In addition, IndraLogic shows the objects that have changed since the last download.

- **Application content:** When the option **Download Application Info** is selected, the contents of the applications on both the (1) development system and (2) PLC can be compared. The **Download Application Info** option is located in the **Application Build Options** tab of the application properties.

If the code in the development system is not current, the following command is displayed in the lower left **Application not up to date. Generate code now?** (3). Execute this command to update the application source code.

This detailed information can help you to better assess the effects of login in the current situation and to make a decision about downloading the new application.

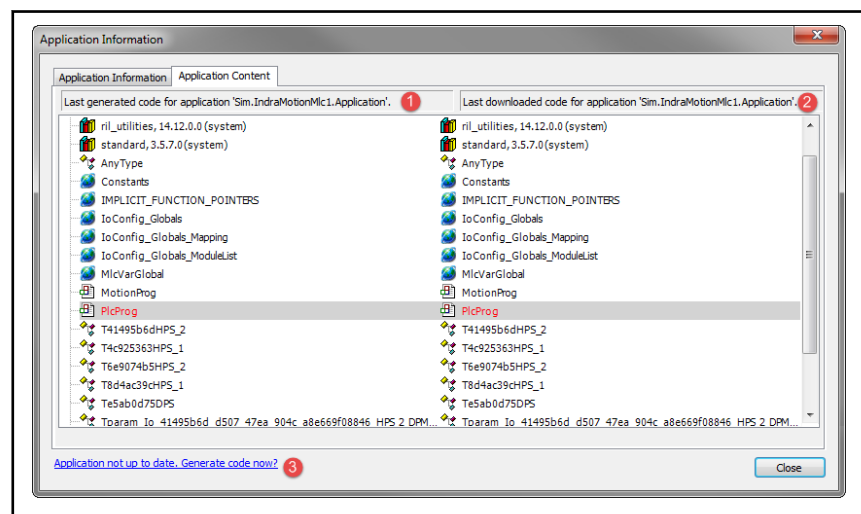


Fig. 18-62:



The comparison can also be displayed in the device editor (**Applications** tab) by clicking **Content**.

Also refer to

- [chapter ""Applications" tab" on page 491](#)
- Working with IndraWorks, comparing project data

Unknown applications on the PLC

If one application is already on the control, but are not in the project, IndraLogic opens a dialog with a list of these applications.

You can then define whether an application should be deleted before loading the current application from the PLC.

Compiling the project before login

If an application program has not been build since the last change, IndraLogic compiles the project before login. This process is the same as the **Build** command when logged out.

If errors occurred during the build process, a message dialog is displayed. The errors are output to the messages view in category **Build**. Decide whether to log in without downloading the program to the control.

Also refer to

- [chapter ""Build" command" on page 611](#)

Error upon login

If an error occurs when logging in to the control, then IndraLogic cancels the loading process with an error message.

Information on the download	The error dialog gives you the options of showing the error details. If an exception was thrown and the text *SOURCEPOSITION* is included in the log, then you can display the affected function in the editor by clicking Show in Editor. The cursor jumps to the line containing the error.
	<i>If IndraLogic downloads the project to the control at login, then the following information is printed to the messages view:</i> • Generated code size • Size of global data • Resulting memory requirement on the control • List of the affected blocks (for online change)
	In online mode, settings of the devices or modules cannot be changed. You must log out of the application for changing device parameters. Depending on the bus system, there may be some special parameters that you can also change in online mode. IndraLogic saves the view configuration separately in online and offline mode. In addition, views are closed that cannot be used in any operating mode. Due to this, the view can change automatically at login.

Command 'Logout'

Symbol	
Function	The command disconnects the application from the target system (control or simulated device) and returns to offline mode.
Call	In menu Debugging , or via the via context menu of the Application object or via the shortcut: <Ctrl> + <F8>.

Command 'Create boot application'

Function	The command generates a boot application.
Call	Debug menu A boot application is the application that is started automatically when the control is switched on or started. In offline mode, you can save the boot application in any directory. In online mode, IndraLogic saves the boot application to the target device. The file name is <ApplicationName>.app. <i>Also refer to</i> • chapter 11.4 "Generating boot applications" on page 177

Command 'Log out current device user'

Symbol	
Function	The command logs out the user currently logged in to the control. If IndraLogic is still connected to the control, it is disconnected.
Call	IndraWorks menu Debugging
Prerequisites	• The application is in online mode. • A user management has been established.



Use the online user management **Project ► User management** to protect the target device.

Also refer to

- IndraWorks user management

Command 'Load'

Function The command compiles the active application and a subsequent download to the control.

Call **Debug** menu

Prerequisite The application is in online mode.

When you execute this command, IndraLogic performs a syntax check and generates the application code.

This code is loaded to the control.

IndraLogic generated a build protocol in the project directory:

Example:

```
<Projectname>.<Devicename>.<Projectname>.<Application> .compileinfo
C:\ProgramData\IW-Projects\x\Project\IndraLogic\IndraLogic.IndraMo-
tionMlc1.Application.*.compileinfo).
```



During loading all variables are re-initialized with the exception of persistent variables.



If a safety controller is configured below a controller, then this command can interrupt the communication connections temporarily.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to chapter [chapter 10.5 "Subordinate safety control" on page 172](#)

The description of the **Login** command describes the possible situations when logging in and loading.

If you attempt to load an application while the same version of this application is already on the controller, the following message appears: "Program is unchanged. Application was not loaded".

IndraLogic does not load the application to the PLC.

During loading a record of the actions being executed (generation of code, execution of initialization, etc.) appears in the Message window in the message category **Compile**. Furthermore, information is displayed regarding the memory ranges, the size of the code, the global data and the allocated memory.

For the purpose of clarity, as opposed to the online change, the modified function blocks are no longer listed.

Also refer to

- [chapter "Menu 'Login'" on page 614](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)

Command 'Online change'

Function	The command initiates an online change on the currently active application. When this is done, IndraLogic re-downloads only the changed parts of an application that is already running on the control.
Call	Menu Debugging , or via the via context menu of the Application object.
Prerequisite	The application is in online mode and is selected in the Project Explorer. The command directly executes an online change on the selected application, even if the selected application is not configured as "active application". An online change is not possible after the Create ► Clean and Create ► Clean all commands. The cleaning process deletes the compile information (build log) that was generated automatically upon each code generation. This log is the basis for an online change.

CAUTION

An online change changes the running application program and requires a restart.

Make sure that the new application code still has the required effect on the controlled system.

Depending on the controlled plant, the plant and workpieces may be damaged or the health and life of persons could be endangered.



1. Outputs retain their values permanently from the last cycle if a change was made in the PLC program and thus the outputs are not used anymore and the change is loaded to the control via online change.
2. If an online change is performed, then the application-specific initializations (for example, reference run) are not executed as the machine retains its status. Due to this reason, the new program code does not have the desired effect.
3. Pointer variables retain their value from the last cycle. If a pointer refers to a variable whose value was changed in an online change, then the variable no longer yields the correct value. Make sure that pointer variables are re-assigned in each cycle.

An online change is automatically provided if login occurs with an application that is already running on the control but has been modified since the most recent download in the programming system.

At the time of download, IndraLogic also lists the changed interfaces, affected variables, and all blocks with new generated code in the "Build" category of the messages view. If memory locations change, a dialog box will inform you of possible problems in conjunction with pointers.



IndraWorks 14VRS and later are provided with a faster version of the online change for small changes:

If only code was changed, only the affected POU's are compiled and subsequently loaded. More code is not executed.

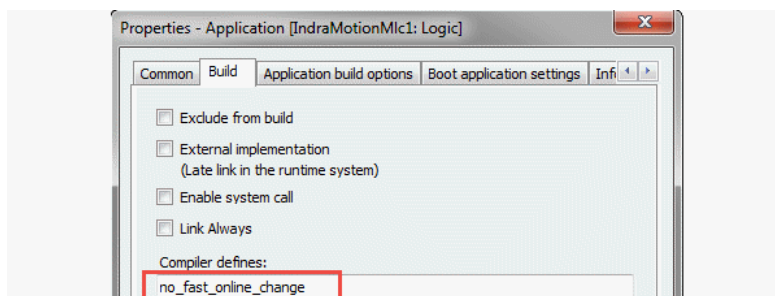
No initialization code is executed.

This means that also no code is generated when variables are initialized with the `init_on_onlchange` attribute.

As a rule, this has no effect, as the attribute is primarily used for initializing variables with addresses. However, a variable cannot change its address during an online change.



To ensure the effect of the `init_on_onlchange` attribute in the entire application code, disabled the fast online change in general for the application by using the compiler definition `no_fast_online_change`.



Insert the definition into the application "Properties" ("Build" tab).

Also refer to

- [chapter "Attribute "init_on_onlchange" on page 392](#)
- [chapter 11.2 "Generating application code" on page 175](#)

Command 'Start'

Symbol



Function This command starts the application (status: "RUN").

Call In menu **Debugging**, or via the via context menu of the **Application** object or via the shortcut: <Alt> + <F5>.

Prerequisite The application is in online mode and its status is "STOP".

Calling the command via the "Debug" menu affects the application, see [chapter "Command 'Set active application'" on page 608](#)

Command 'Stop'

Symbol



Function This command stops the application (status: "STOP").

Call In menu **Debugging**, or via the via context menu of the **Application** object or via the shortcut: <Ctrl> + <F5>.

Prerequisite The application is in offline mode and its status is "RUN".

Calling the command via the "Debug" menu affects the application, see [chapter "Command 'Set active application'" on page 608](#)

Command 'Single cycle'

- Function** The command executes the active application for one cycle.
- Call** Menu **Tools** ► **Customize** ► **Commands**, category "Online".
- Prerequisite:** The application is in online mode and the program is stopped at a program step.

Command 'Multiple download'

- Function** The command results in the generation of code of the applications contained in the project as well as loading of the applications to the respective controls.
- Call** In menu **Debugging**.
- The command opens a dialog with a list of the applications.
- In this dialog, select the applications that are to be loaded.
- Then, IndraLogic performs the syntax check of these applications and generates the respective code. The code is downloaded to the respective control.
- In the project directory (C:\ProgramData\IW-Projects\x\Project\IndraLogic), IndraLogic generates a compilation log for each selected application with the name <project name>.<device name>.<application ID> .com-
pileinfo.



If a safety controller is configured below a controller, then this command can interrupt the communication connections temporarily.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to chapter [chapter 10.5 "Subordinate safety control"](#) on page 172

Dialog 'Multiple download'

Please select the applications	 : Selection of the applications. The applications are thereby also loaded to different controllers.
Up, down	Change of the download order of the applications. The applications are downloaded to the PLCs in the order of this list. By default, this list is alphabetically sorted. Parent-child relationships of applications are thereby taken into account.
OK	Checks the syntax of all selected applications. Afterwards, the communication with the associated controller is verified for each application before the download takes place.

Online change options

If an earlier version already exists on the control and is different from the current version, then the following options are provided:	
Try online change. If not possible, execute download	Enabled by default. If an online change cannot be executed for one of the applications, then a download is performed.

Forcing an online change. If this is not possible, cancel the operation	If an online change cannot be performed for (at least) one of the applications, then no download is performed and the online change is terminated (for example, if you have executed the command "Clean All" beforehand).
Always execute complete download	Downloads all parts of the applications to the PLC, regardless of any existing versions.

For selected applications that do not exist on the control yet, IndraLogic performs a download automatically to the control.

Additional options

Delete all applications from the controller that are not contained in the project	 : Corresponding applications are deleted
Start all applications after download or online change	 : The applications are started after the download/online change.



Note that variables with the key attribute `PERSISTENT RETAIN` are not generally initialized. If you change the data layout, however, the persistent variables are automatically re-initialized.

After completion of the download a listing of all selected applications appears in the download order that you configured. In addition, you are shown information on the success of the download for each application.

Also refer to

- [chapter "Command 'Load'" on page 618](#)
- [chapter 11.2 "Generating application code" on page 175](#)

Command 'Reset warm'

Function	The command results in a warm start of the active application on the controller.
Call	In the Debug menu
Prerequisite	The application is in online mode. If you have not defined any initialization values, then IndraLogic resets the variables to the default initialization value (for example, integer variables to zero). As a safety precaution, IndraLogic requires a user confirmation before all variables are overwritten. The situation is comparable to a loss of power or switching the control off and back on while the application is running ("warm start"). Breakpoints of the application enabled before the active application was reset, are still active after executing the command. Breakpoints that were disabled before are still disabled after the command execution. When executing the command while the program sequence just stops at a breakpoint, you are prompted whether the current cycle should really be ended. Alternatively, IndraLogic executes the reset immediately.



If a safety controller is configured below a controller, then this command can interrupt the communication connections temporarily.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to chapter [chapter 10.5 "Subordinate safety control" on page 172](#)

After restarting with "Reset Warm", the following happens:

- Application code remains loaded on the controller.
- Variables are initialized (with the initialization value or the default initialization value 0).
- Retain variables are retained with values.
- Persistent variables are retained with values.
- Breakpoints that were set in the code are retained with their status (for example, activated or deactivated).
- The application goes into the "STOP" state.

You can also select the command while debugging the application when it halts at a breakpoint in the "HALT ON BP" state. Then either the warm start is executed immediately, or the remaining statements of the current cycle are processed. Therefore, a message window opens for you to select the next action. However, the message window opens only if the runtime system is capable of restarting the cycle without terminating it first.

After the reset, you can run the application as usual and, for example, start execution by using the **DebugStart** command.

Also refer to

- [chapter 12.5 "Resetting applications" on page 185](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)
- [chapter "Command 'Reset cold'" on page 623](#)
- [chapter "Command 'Reset origin'" on page 624](#)

Command 'Reset cold'

Function	The command results in a cold start of the active application on the control.
Call	In menu Debugging .
Prerequisite	The application is in online mode.



If a safety controller is configured below a controller, then this command can interrupt the communication connections temporarily.

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to [chapter 10.5 "Subordinate safety control" on page 172](#)

After restarting with "Reset Cold", the following happens:

- Application code is retained on the controller.
- Variables are initialized (with the initialization value or the default initialization value 0), and the previous values are lost.
- Retain variables are initialized, and the previous values are lost.
- Persistent variables are retained with values.
- Breakpoints that were set in the code are retained with their status (for example, activated or deactivated).
- The application goes into the STOP state.

You can also select the command while debugging the application when it halts at a breakpoint in the "HALT ON BP" state. Then either the warm start is executed immediately, or the remaining statements of the current cycle are processed. Therefore, a message window opens for you to select the next action. However, the message window opens only if the runtime system is capable of restarting the cycle without terminating it first.

After the reset, you can run the application as usual and, for example, start execution by using the **DebugStart** command.

Also refer to

- [chapter 12.5 "Resetting applications" on page 185](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)
- [chapter "Command 'Reset warm'" on page 622](#)
- [chapter "Command 'Reset origin'" on page 624](#)

Command 'Reset origin'

Function	The command results in a reset origin of the active application on the controller.
Call	In the Debug menu
Prerequisite	The application is in online mode.



If a safety controller is configured below a controller, then this command can interrupt the communication connections **permanently!**

Connections of the safety controller to other safety controllers (via safety network variables), to field devices, and to the development system are affected. The safe field devices or other safety controller can enter the safe state as a reaction. The connection to the development system is affected only when a safety controller that is connected to the main controller via a fieldbus.

For more information, refer to chapter [chapter 10.5 "Subordinate safety control" on page 172](#)

After restarting with "Reset Origin", the following happens:

- The application code is deleted, and as a result the application has no state.
- Variables are deleted, and the values are lost.
- Retain variables are deleted, and the values are lost.
- Persistent variables are deleted, and the values are lost.
- Breakpoints that were set in the code are lost.

Also refer to

- [chapter 12.5 "Resetting applications" on page 185](#)
- [chapter 9.16.1 "Retaining data using persistent variables" on page 140](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter 10.5 "Subordinate safety control" on page 172](#)
- [chapter "Command 'Reset warm'" on page 622](#)
- [chapter "Command 'Reset cold'" on page 623](#)

Command 'Breakpoints'

Symbol



Function

The command opens the **Breakpoints** view.

Call

In the **Debug** menu.

This dialog shows an overview of all defined breakpoints of an application. Within the view, all breakpoint commands are available.

Application	Select the application from the list.
POU	Name of the function block containing the breakpoint.
Position	<i>Breakpoint position in the POU</i> • Textual editor: Line number and column number • Graphical editor: Network or element number For function blocks, "(Impl)" indicates that the breakpoint is located in the implementation of the function block, not in an instance.
Instance path	Complete object path of the breakpoint position.
Tasks	Names of tasks that will be effective when the breakpoint is executed. If there are no restrictions, "(all)" is displayed.

Condition	• Always break: No additional enable condition defined; the breakpoint is always enabled. • Boolean expression. The expression has to be TRUE for an enabled breakpoint.
Hit count condition	When the breakpoint should be in effect (depending on the hit count).
Current hit count	Indicates how often the breakpoint has been ("hit") during the execution up to this point.

Tab. 18-12: Table of current breakpoints

	New breakpoint	The command opens the Breakpoint properties dialog
	Clear breakpoint	Removes the breakpoint (not the same as disable)!
	Enable/disable breakpoint	<i>Toggles the status of the breakpoint between "enabled" and "disabled".</i> •  Breakpoint enabled •  Breakpoint disabled As opposed to "Clear breakpoint", a disabled breakpoint remains in the list and can be enabled again.
	Properties	The Breakpoint properties dialog opens to edit the breakpoint parameters. This dialog corresponds to the New breakpoint dialog.
	Go to source position	Opens the online view of the respective function block. The cursor is set at the breakpoint position.
	Clear all breakpoints	Deletes all breakpoints of the application. The list is cleared. Do not confuse this with disabling!
	Enable all breakpoints	Enables all currently disabled breakpoints.
	Disable all breakpoints	Disables all currently enabled breakpoints. The points remain in the list and can be enabled again.

Tab. 18-13: Toolbar

Also refer to

- [chapter 18.3.2 "'Breakpoint properties' dialog" on page 701](#)
- [chapter "Command 'New breakpoint'" on page 626](#)
- [chapter "Command 'Enable breakpoint'" on page 627](#)
- [chapter "Command 'Disable breakpoint'" on page 627](#)
- [chapter "Command 'Switch breakpoint'" on page 627](#)

Command 'New breakpoint'

Symbol



Function The command opens the **Breakpoint properties** dialog.

Call In menu **Debugging**.

Prerequisite The application has to be in online mode.



With the **Toggle Breakpoint** command, you can set a new breakpoint directly at the current cursor position in online mode.

Also refer to

- [chapter 18.3.2 ""Breakpoint properties" dialog" on page 701](#)
- [chapter 12.2 "Using breakpoints" on page 179](#)

Command 'Edit breakpoint'

Symbol



Function The command opens the **Breakpoint properties** dialog.

Call In menu **Debugging**.

Prerequisite The application is in online mode and the cursor is stopped at a breakpoint.

Also refer to

- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter 18.3.2 ""Breakpoint properties" dialog" on page 701](#)

Command 'Switch breakpoint'

Function The command sets a breakpoint or clears an existing breakpoint.

Call In menu **Debugging**, or via the shortcut <Alt> + <F9>.

Prerequisite The application is in online mode and the cursor is stopped at a breakpoint.

Also refer to [chapter 12.2 "Using breakpoints" on page 179](#)

Command 'Enable breakpoint'

Function The command enables a disabled breakpoint.

Call In menu **Debugging**.

Prerequisite The application is in online mode and the cursor is halted at a disabled breakpoint.

Also refer to [chapter 12.2 "Using breakpoints" on page 179](#)

Command 'Disable breakpoint'

Function The command disables an enabled breakpoint.

Call In menu **Debugging**.

Prerequisite The application is in online mode and the cursor is halted at an enabled breakpoint.

Also refer to [chapter 12.2 "Using breakpoints" on page 179](#)

Command 'Call list'

Symbol



Function The command opens the **Call stack** view.

Call In menu **Debugging**.

This view "Call list" is very useful when you want to execute programs incrementally.

The call list shows the current location with the complete call path.

Application	Name of the active application that controls the block that has just been reached.
Task	Name of the task that controls the block that has just been reached.
POU	Name of program block in which the program execution is contained. The first line in the list describes the current execution location and is indicated by a yellow arrow. If this position is located in a function block that is called by another function block, the position of the call is described in the second line. If the caller is then called by another function block, this call position is described in the third line, etc.
Position	<i>Position within the block where program execution has halted</i> - Line number and column number for text editors - Network number or element numbers for graphical editors
Instance path	Instance, where program execution has stopped.

The call list is in offline mode and is available in offline mode if no debugging functions are used. In this case the call list contains the most recent position indicated during an incremental execution, displayed as "grayed out".



The "call tree" view provides call information about a function block at any time, in contrast to the "call list".

Also refer to

- [chapter 12.2 "Using breakpoints" on page 179](#)
- [chapter "Command "Call tree"" on page 592](#)

Command 'Monitoring' - 'Monitoring list <n>'

Symbol



Function

The command opens the **Watch list <n>** view.



<n> can be 1, 2, 3, or 4 for a total of up to four watch lists.

Call

In menu **Debugging**.

Also refer to [chapter 13.1.2 "Using watch lists" on page 196](#)

The monitoring list can be assigned with a project variable and facilitates the following function within a single overview:

- Monitoring of variables
- Forcing variables
- Writing of variables

Command 'Monitoring' - 'Show all forces'

Symbol



Function

The command opens the **"Show all forces"** view.

Call

In menu, select **Debug ► Monitoring ► Show all forces**

Prerequisite

A project is open in the Project Explorer (in online or offline mode).

The view contains a list with all application variables, prepared for forcing or that are already forced.

The list provides all actions that can also be executed in other monitoring lists.

Additionally, the list provides the following commands in the drop-down menu **Release force**:

- **Unforce and retain all selected values:**
The variables are set to the forced values and it is then unforced for all selected list entries.
- **Unforce and restore all selected values:**
The variables are set to the value before forcing and forcing is canceled for all selected list entries.

Also refer to

- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter 13.1.2 "Using watch lists" on page 196](#)

Command 'Add all forces to monitoring list'

Function	The command adds all application variables to the monitoring list prepared for forcing or that have already been forced.
	The command is only supported by the active monitoring lists!
Call	In context menu of the "Watch list" view.
Prerequisites	• The application is in online mode. • At least one monitoring list is active
	The monitoring list "Watch all forces" provides an overview of all application variables prepared for forcing or that have already been forced. The monitoring list provides additional commands to cancel forcing.

Also refer to

- [chapter 13.1.2 "Using watch lists" on page 196](#)
- [chapter "Command 'Monitoring- 'Monitoring list <n>'" on page 628](#)
- [chapter "Command 'Monitoring- 'Show all forces'" on page 628](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

Command 'Procedure step'

Symbol	
Function	The command executes the statement where the program is currently located and halts before the next statement in the POU.
Call	In menu Debugging , or via the shortcut <F12>.
Prerequisite	The application is in online mode and the program is halted at the current break position (debug mode). If the executed statement contains a call (from a program, function block instance, function, method, or action), then the subordinate POU is processed completely in one step and returned to the call. Then it halts before the next statement (in the next line of code).



Click "Step Into" to jump to a subordinate POU and process it in single steps.

Also refer to

- [chapter 12.3 "Stepping a program" on page 181](#)
- [chapter "Command 'Single step'" on page 630](#)

Command 'Single step'

Symbol

Function The command executes the statement where the program is currently located and halts before the next statement.

Call In menu **Debugging**, or via the shortcut <Alt> + <F12>.

Prerequisite The application is in online mode and the program is halted at the current break position (debug mode).

If the executing statement contains a call (from a program, function block instance, function, method, or action), then the program execution jumps to this subordinate POU. Its code opens in a separate editor. The first statement is executed and the program execution is stopped before the next statement. The new current breakpoint position is then in the called POU.



Click "Step Over" to remain in the currently active POU and execute the call in one step.

Also refer to

- [chapter 12.3 "Stepping a program" on page 181](#)
- [chapter "Command 'Procedure step'" on page 629](#)

Command 'Execute to return'

Symbol

Function The command executes the program until the next return and stops afterwards.

Call In menu **Debugging**, or via the shortcut <Shift> + <F10>.

Prerequisite The application is in online mode and the program is halted at the current break position (debug mode).

If the current breakpoint position is contained in a subordinate POU, this POU run is completed. Then the program execution jumps back to the calling point in the calling POU and halts there (in the line with the call).

If the current breakpoint position is contained in the main program, this POU run is completed. Then the program execution jumps back to the beginning (to the program start at the first line of code in the POU) and halts there.

Also refer to

- [chapter 12.3 "Stepping a program" on page 181](#)

Command 'Execute to cursor'

Symbol

Function The command executes a program until a specified position as marked by the cursor.

Call In menu **Debugging**.

Prerequisite	The application is in online mode and the program is halted at the current break position (debug mode). Moreover, you have marked any line of code in any POU with the cursor. The statements between the current breakpoint position and the cursor position are executed in one step. Then the execution halts at the cursor position, which then becomes the next breakpoint position. Remember that the line of code where you placed the cursor is reached but not executed. <i>Also refer to</i> • chapter 12.3 "Stepping a program" on page 181

Command 'Determine next statement'

Symbol	
Function	The command specified the instruction executed next.
Call	In menu Debugging .
Prerequisite	The application is in online mode and the program is halted at the current break position (debug mode). Moreover, you have marked any line of code in any POU with the cursor. The line of code marked with the cursor becomes the current breakpoint position without executing the statements in between or the statement that jumped to it. Also refer to chapter 12.3 "Stepping a program" on page 181

Command 'Show next statement'

Symbol	
Function	Jumps to the program statement executed in the next step.
Call	In menu Debugging .
Prerequisite	The application is in online mode and the program is halted at the current break position (debug mode). The break position is in a line of code that you cannot see. The command makes the window with the current breakpoint position active in the code highlighted in yellow and marked with the  symbol and makes the breakpoint position to become visible. This is useful if you have multiple editors open and the breakpoint position is hidden in an inactive editor. Also refer to chapter 12.3 "Stepping a program" on page 181

Command 'Force values'

Function	The command sets a permanent predefined value to a variable on the control.
Call	In menu Debugging , or via the shortcut <F7>.
Prerequisite	The application is in online mode.

CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Evaluate possible risks before forcing variable values and take corresponding precautions. Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

With this command, IndraLogic permanently sets one or more variables of the active application to defined values on the control.

This is done at the beginning and end of a processing cycle.

Processing sequence:

1. Read inputs
2. Force values
3. Process code
4. Force values
5. Write outputs

Values can be prepared by:

- Click in the **Prepared value** field in the declaration section. Enter the new value here.
For Boolean variables, you change the value by clicking the field.
- Click in the inline monitoring field in the implementation section of the FBD/LD/IL editor. Enter the new value here
- Click in the **Prepared value** field in the monitoring window. Enter the new value here.

A forced value is marked with the forced symbol .

IndraLogic forces the value until it is canceled by the user. Cancelling takes place by:

- Click on **Cancel forcing for all values**
- Cancel force in the **Prepare value** dialog
- Log out of the application

Also refer to

- [chapter 18.3.3 ""Prepare value"" on page 701](#)
- [chapter "Command 'Cancel forcing for all values'" on page 633](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

Command 'Write values'

Function	The command sets a predefined value to a variable on the control one time.
Call	In menu Debugging , or via the shortcut <Ctrl> + <F7>.
Prerequisite	The application is in online mode.

CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Evaluate possible risks before forcing variable values and take corresponding precautions. Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

With this command, one or more variables of the active application are set to defined values on the control one time. It is written once at the beginning of the next cycle.

Values can be prepared by:

- Click in the **Prepared value** field in the declaration section. Enter the new value here.
For Boolean variables, you change the value by clicking the field.
- Click in the inline monitoring field in the implementation section of the FBD/LD/IL editor. Enter the new value here.
- Click in the **Prepared value** field in the monitoring window. Enter the new value here.

Also refer to

- [chapter "Command 'Force values'" on page 631](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

Command 'Cancel forcing for all values'

Function	The command resets the forcing of all variables. The variables then receive their current value(s) from the control.
Call	In menu Debugging , or via the shortcut <Alt> + <F7>.
Prerequisite	The application is in online mode.



The "Remove force list" command has the same functionality as this command with one difference:

If the "Remove force list" command cannot be executed for all forced values, then no message is displayed.

CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Evaluate possible risks before forcing variable values and take corresponding precautions. Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

Also refer to

- [chapter "Command 'Force values'" on page 631](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)

Command 'Switch sequence control mode'

Symbol	
Function	The command enables and disables the sequence control.
Call	In the Debug menu
Prerequisite	The application is in online mode.



An active flow control extends application runtime!

When **Safe online mode** is activated in the communication settings, a dialog is displayed when switching on the sequence control to cancel the process.

When flow control is activated, it is not possible to use break-points or step through the program.

Also refer to [chapter 12.6 "Sequence control" on page 186](#)

Command 'Operating mode'

- Function** Via the operating modes of the "operating mode" command, the control can be switched to a state that prevents an accidental project change.
- Call** In menu **Debug ► Operating mode**.
- Prerequisites**
- A boot application has to be available on the control for the application
 - No active breakpoints may be set
 - The application has to run
 - No forced values may exist

Use the operating mode command to lock the state of a control. For example, to prevent that the control changes the state while programming takes place at a different control.

After programming has been completed, the control has to be in a defined state, which is visible from outside and which sets itself again after a reboot.

The following operating modes are available:

 Debug	No restrictions
 Blocked	The current state of debugging on the application is locked. No breakpoint can be set and no variables can be forced. However, writing variables is possible.
 Ready for operation	Nothing can be changed. Breakpoints cannot be set. However, writing variables is possible.

In the status line, the symbols ,  and  show the current operating mode. Double-click on one of these symbols to open a help window.



The operation mode between **Blocked** and **Ready for operation** cannot be switched.

Command 'Core dump'

Command 'Create core dump'

- Function** The command triggers the creation of a core dump.
- Call** In menu **Debug ► Core Dump**.
- Prerequisite** The application is in online mode.
- Effect of the command: IndraLogic checks whether a core dump file is already available on the control.
- If a core dump file is already available, this file can be loaded to the project directory in IndraLogic.
- Under the following preconditions, IndraLogic creates a new dump file with the current application data:
- A core dump file is not yet available or IndraLogic has rejected loading of a core dump file
 - AND: The application is currently stopped at breakpoint or an exception has occurred
- The generated core dump file is saved directly to the project directory:

Example:

```
<Projectname>.<Devicename>.<Applicationname>.<Applicationguid>.core
```

(e.g. C:\ProgramData\IW-Projects\x\Project\IndraLogic\IndraLogic.IndraMotionMlc1.Application.*.core).

You can cancel the file generation by clicking the button in the status bar.

The runtime system support decides how detailed the core dump is. A core dump is always created in case of an exception.

Log out from the PLC.

A core dump can then be loaded.

The core dump output by using the **Load core dump** command can be used for error analysis.

Also refer to

- [chapter 13.6 "Error analysis using core dump" on page 208](#)
- [chapter "Command 'Load core dump'" on page 635](#)

Command 'Load core dump'

Function	IndraLogic scans the project directory for core dump files.
Call	In menu Debug ► Core Dump .
Prerequisite	The application is in offline mode.

When a new core dump is forced with the **Create Core Dump** command, the dump file is automatically loaded from the control to the project directory.

If multiple core dump files are available, a prompt is displayed, asking the user to select whether the latest file should be opened in the project. You can also select one of the other files.

When a file is loaded into the project, an online view of the application appears with state of the application at the time when the core dump was generated. Then, view the variable values afterwards. Finally, the call tree is also available.



You can only close the core dump view via the **Close core dump** command.

The **Logout** command is not applied to this view!

Also refer to

- [chapter 13.6 "Error analysis using core dump" on page 208](#)
- [chapter "Command 'Create core dump'" on page 634](#)
- [chapter "Command 'Close core dump'" on page 636](#)

Command 'Load device logbook from core dump'

Function	The command loads the control log list to the project that was saved with the last generated core dump.
Call	In menu Debug ► Core Dump .
Prerequisites	• The application is in offline mode • A core dump is open in the project The log list is displayed in the same view as in online mode in the Log tab of the device editor.

Also refer to:

- [chapter 13.6 "Error analysis using core dump" on page 208](#)
- [chapter ""Log" tab" on page 495](#)

Command 'Close core dump'

Function	The command closes the core dump view of the application, opened in the project.
Call	In menu Debug ► Core Dump .
Prerequisites	• The application is in offline mode • A core dump file was loaded from the control to the project

Also refer to [chapter 13.6 "Error analysis using core dump" on page 208](#)

Command 'Display mode'

Function	The "display mode" command has a submenu for the display formats binary, hexadecimal and decimal. By selecting one of the three formats, the value representation during monitoring in online mode is specified.
Call	In menu Debugging .
Prerequisite	The project is in online or in offline mode.

Supported display formats

Binary	unsigned
Hexadecimal	unsigned
Decimal	signed

Also refer to [chapter 13.1.1 "Calling monitoring in programming object" on page 191](#)

18.2.6 Menu 'Tools'

Libraries

"Library repository" command

Symbol: 

Function: The command opens the **Library repository** dialog. In this dialog you define which libraries are installed on the local system and are thus available for your application.

Call: Menu **Tools ► Libraries**.

"Library repository" dialog

Display of the directory on the local system in which the library files are located. The libraries in this location are listed in the Installed libraries area.	
Edit	Opens the Edit repository locations dialog.

Tab. 18-14: **Location**



1. You can only use empty directories for new repositories.
2. You can also use existing repositories as locations.
3. The "system" repository is not editable; IndraLogic indicates this by the italic lettering of the entry.

Edit repository storage locations

List of the repositories with storage location and name .	
Add	Creates a new repository. Opens the storage location for repository dialog. The selected directory (input field Place) must either be empty or an already existing valid repository. Name is the input field for a symbolic repository name.
Edit	Opens the Location for repository dialog (see Add)
Remove	A query appears, asking whether only the entry is to be removed from the list of repositories, or whether the directory with the library files is to be deleted from the file system. If you wish to delete the directory, you must confirm this.

Installed libraries

List of the libraries in a tree structure. Display of each library with category, name, company and version.	
Company:	Selection list for filtering the displayed libraries.
Install	Opens the Select library dialog. By default the filter is set to IndraLogic libraries, which have the file extension "*.library" as standard. The filter "IndraLogic libraries (before 2G)" displays all libraries with the extension "*.lib".
Uninstall	Uninstalls the selected library.
Export	Opens the standard dialog to save the library project to the local file system. The file type is Libraries (*.library) or Compiled libraries (*.compiled-library) .
Search	Searches for libraries and function blocks. Opens the Browse library dialog. If you enter a string in the input field, IndraLogic displays the libraries that it finds with a corresponding string.
Details	Opens the Details dialog with details from the "Library info" of the library for the selected version of a library. For the following additional information, click on the More button in the Details dialog: • Size: Specified in bytes • Created: Creation date • Changed: Date of the last change • Last access: Date • Attributes • Properties
Dependencies	For the selected library the Dependencies dialog opens, showing the dependencies on other libraries. Title , Version and Company are shown for each library reference. References that function via placeholders are displayed according to the syntax: #<placeholder name>.
Group by category	• : Grouping according to library categories • : Alphabetical sorting The categories are defined by external description files "*.libcat.xml".

Library profiles

A library profile defines the library version with which IndraLogic resolves a library placeholder if a certain compiler version is set in the project.

Import	Imports a *.libraryprofile file. If the import already contains existing placeholder entries, a query appears asking whether IndraLogic should overwrite it.
Export	Exports an xml file with the extension ".libraryprofile" with the assignments of the selected placeholder entries; you can only select a single entry of a compiler version .



Placeholder resolutions can also be defined in the target device currently in use and even by a specific local specification in the Placeholders dialog in the library manager.

Also refer to

- [chapter 18.1.20 "Library info" object](#) on page 545
- [chapter 18.1.14 "Object 'Library manager'"](#) on page 516

Generating a Library in IndraWorks

The libraries of the IndraLogic PLC programming system contain function blocks, functions, data types, variable lists and constants provided by the control manufacturer. The libraries are either directly installed during the installation of "IndraWorks Engineering" and are visible in the library manager of the relevant application or they can be added from the library repository.

Also refer to

- [chapter 18.1.14 "Object 'Library manager'"](#) on page 516
- [chapter ""Library repository" command"](#) on page 636

General information on the library creation

Function: It is described in the following how to create a "library" or "compiled library" from function blocks, functions, data types, etc. which is equal to the manufacturer's libraries.

Call: Menu **Tools ► Libraries**.

1. Creating new project
Create a new project as "Library project".
Specify a suitable project name.
The project contains the "General module folder" with library info and project settings.
2. Structurally arrange the library content
Function blocks, functions, data types, etc. are to be arranged structurally in the "General module" folder.
3. Selecting the type of library creation
Select one of the three following options:
 - ["Generate library..."](#) on page 639
 - ["Generate compiled library..."](#) on page 640
 - ["Generate and install compiled library..."](#) on page 640



All menu items can only be used if the opened project contains a "Library info" object.

Also refer to

- [chapter 18.1.20 "Library info" object](#) on page 545

- If no "Library info" object is available and one of these menu items is selected, the following message is output:

Error message, library info object not (yet) available:

The "Library info" object does not exist.

In this case, use the context menu to add a "Library info" object.

- In this object, at least complete the mandatory fields "Company", "Title" and "Version".

In the example: "BRC", "TestLib", "1.0.0"

If this was not performed or not performed correctly, the following message is output.

Message: Mandatory parts of the library info object are missing

The "Library info" object is not complete. Company, title and version are required.



This mandatory data is later used by the library in the library repository.

The name of a library or compiled library is generally displayed in the details of the library repository (normally, title and name should match):

From now, the process and the result of the three menu items differ:

Generate library...

Symbol:

Call: Context menu of the "General module folder" **Generate library...**

If the folder is highlighted: **Tools ► Libraries ► Generate library...**

Creating a folder in the file system of your computer to integrate a library

- Select the directory to store the library. It has to differ from the IndraLogic directory of the currently opened project. If this is not the case, the user is requested to select a different directory. (The library name can be changed when storing it)

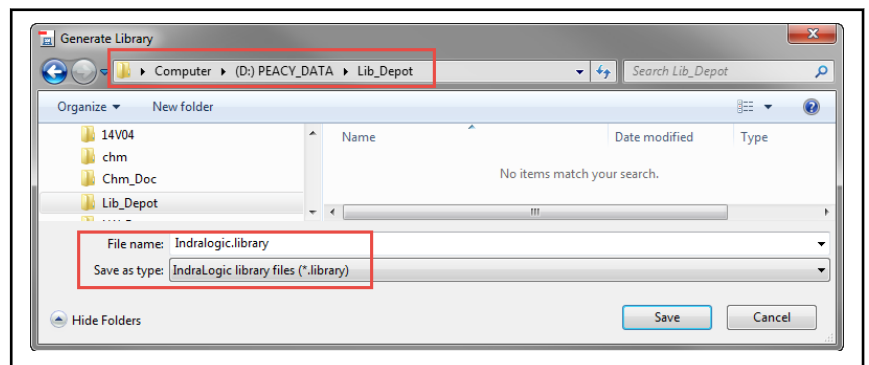


Fig. 18-63: Dialog: Selection of the storage location and the name of the library to be generated

After confirming this dialog, the open project is saved first and then, the library file is generated in the selected directory.

Path "D:\LibDepot", file "Indralogic.library").



The "storage" name of the library is only used for finding and management outside the library repository. It does not affect the title in the "Library info".

Generate compiled library...

Symbol: 

Call: Context menu of the "General module folder" **Generate compiled library...**

If the folder is highlighted: **Tools ► Libraries ► Generate compiled library...**

Creating a folder in the file system of your computer to integrate a library

1. Select the directory to store the library. It has to differ from the IndraLogic directory of the currently opened project. If this is not the case, the user is requested to select a different directory. (The library name can be changed when storing it)

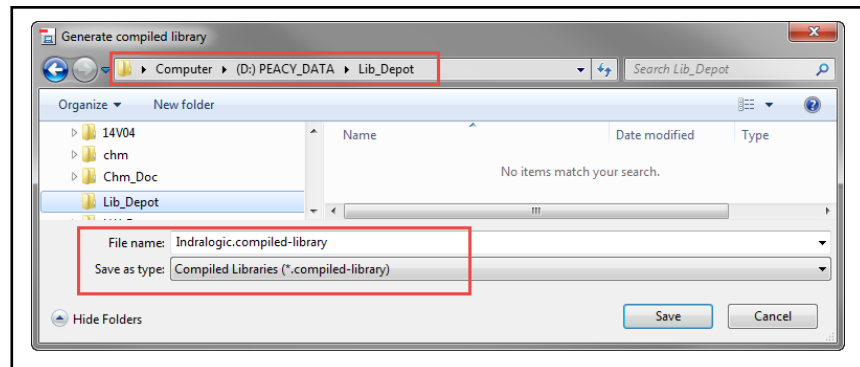


Fig. 18-64: Dialog: Selection of the storage location and the name of the "compiled library" to be generated

After confirming this dialog, the open project is saved first and then, the library file is generated in the selected directory.

Path "D:\LibDepot" the file "Indralogic.compiled-library").



The "storage" name of the "compiled library" is only used for finding and management outside the library repository. It does not affect the title in the "Library info".

Generate and install compiled library...

Symbol: 

Call: Context menu of the "General module folder" **Generate and install compiled library...**

If the folder is highlighted: **Tools ► Libraries ► Generate and install compiled library...**

Create library and store in library repository

1. IndraLogic generates the desired library and installs it immediately in the library repository.

The data is taken from the mandatory fields "Company", "Title" and "Version" of the library info.

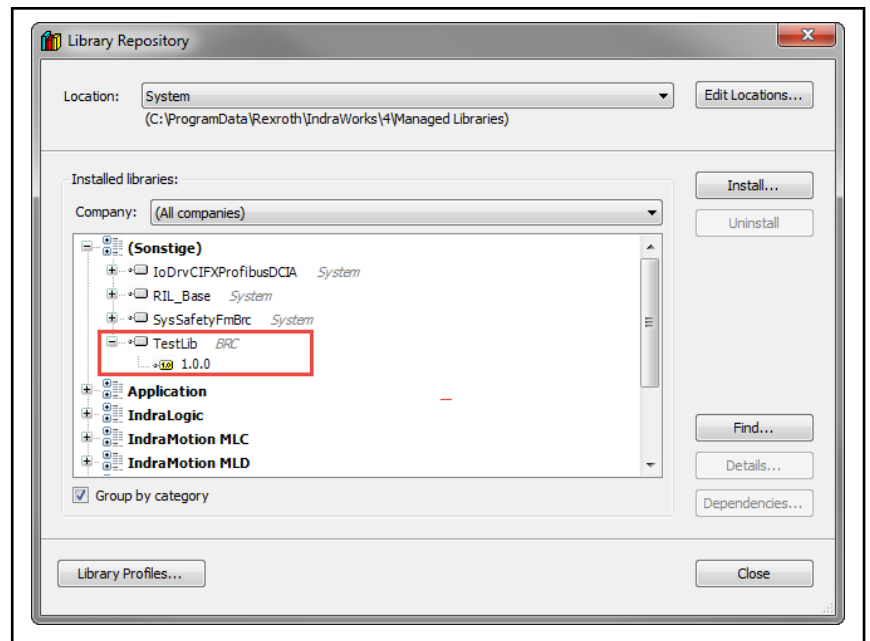


Fig. 18-65: Dialog: Library repository, new library "TestLib"

"Check for library compatibility" command

Function: The command starts checking whether the currently open library project is compatible to the latest installed library version (preceding version number).

Call: Menu **Tools** ► **Libraries**

Prerequisite: A library project is open. The library project is provided with a "Library info" object.

Differences in the implemented interfaces of a method and part of the check. After the test, error messages are displayed in the following cases in the message window:

- Adding or removing inputs or outputs of function blocks, functions or methods
- Changing the data type of inputs or outputs
- Changing the implemented interfaces of a method

Also refer to

- Customize
- [chapter 16.1 "Information for library developers" on page 219](#)

"Customize" command

Function: The command opens the **Customize** dialog. The "Customizing" dialog provides the option of editing the menu and the toolbars. The dialog is part of the Engineering tool IndraWorks.

Call: **Tools** menu

"Options" command

Function: The command opens the dialog **Options** for the configuration of the IndraWorks/IndraLogic options.

These options define the behavior and appearance of the IndraWorks user interface.

IndraWorks saves the settings in your current user profile on your local system. The current profile specifies the default settings.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**

Also refer to

- [chapter ""Options" dialog - "General Settings"" on page 720](#)
- [chapter ""Options" - "CFC editor" dialog" on page 721](#)
- [chapter ""Options" – "Declaration editor" dialog" on page 722](#)
- [chapter "Dialog 'Options' - 'Device editor' " on page 722](#)
- [chapter ""Options" - "FBD, LD and IL" dialog" on page 723](#)
- [chapter ""Options" dialog - "IndraLogic 1x converter"" on page 725](#)
- [chapter ""Options" – "Libraries" dialog" on page 726](#)
- [chapter ""Options" - "Refactoring" dialog" on page 726](#)
- [chapter ""Options" dialog - "Safe FBD options"" on page 727](#)
- [chapter ""Options" - "SFC editor" dialog" on page 728](#)
- [chapter ""Options" dialog - "Smart coding "" on page 730](#)
- [chapter ""Options" - "Syntax Highlighting" dialog" on page 731](#)
- [chapter ""Options" dialog - Text editor" on page 731](#)
- [chapter ""Options" dialog - Visualization" on page 734](#)
- [chapter ""Options" dialog - "Visualization style"" on page 735](#)

18.2.7 Menu 'AS'

"Init step" command

Symbol: 

Function: The command converts the selected step into an initial step.

Call: SFC menu

After you choose this command, the borders of the step element change to a double line. The step that was previously the initial step automatically becomes a "normal step" and is displayed with a simple frame.

You can also activate and deactivate the property **Init step** in the properties dialog of a step. However, IndraLogic does not automatically adjust the settings of other steps.

The command can be useful if converting the chart. When a new SFC object is created, it automatically contains an initial step, followed by a transition (TRUE) and a jump back to the initial step.



Please note: In online mode, it is possible to reset the diagram to the initial step using the `SFCInit` and `SFCReset` flags.

Also refer to

- [chapter "SFC flags" on page 241](#)

"Insert step-transition" command

Symbol: 

Function: The command inserts a step and a transition before the selected point.

Call: SFC menu

When a step is selected, IndraLogic inserts a new combination of step and transition. When a transition is selected, a new combination of transition and step is inserted.

By default, the new step is named `Step<n>`. `n` is a consecutive number, starting with 0 for the first step, that is added to the initial step.

By default, the new transition is accordingly named `Trans<n>`. Click on the name to directly edit the default name.

Also refer to

- [chapter ""Init step" command" on page 642](#)
- [chapter ""Insert step-transition after" command" on page 643](#)
- [chapter "SFC elements "steps" and "transition"" on page 245](#)

"Insert step-transition after" command

Symbol: 

Function: The command inserts a step and a transition after the selected point.

Call: SFC menu

When a step is selected, IndraLogic inserts a new combination of transition and step. When a transition is selected, a new combination of step and transition is inserted.

By default, the new step is named `Step<n>`. `n` is a consecutive number, starting with 0 for the first step, that is added to the initial step. By default, the new transition is accordingly named `Trans<n>`. Click on the name to directly edit the default name.

Also refer to

- [chapter ""Insert step-transition" command" on page 642](#)

"Parallel" command

Symbol: 

Function: The command converts the selected alternative branch into a parallel branch.

Call: SFC menu

Prerequisite: The horizontal connecting line of a branching is selected.

Please note that after you convert a branch, you must check and modify the layout of the steps and transitions before and after the branch.

Also refer to

- [chapter ""Alternative" command" on page 643](#)

"Alternative" command

Symbol: 

Function: The command converts the selected parallel branch into an alternative branch.

Call: SFC menu

Prerequisite: The horizontal connecting line of a branching is selected.

Please note that after you convert a branch, you must check and modify the layout of the steps and transitions before and after the branch.

Also refer to

- [chapter ""Parallel" command" on page 643](#)

"Insert branch" command

Symbol: 

Function: The command inserts a branch to the left of the selected point.

Call: SFC menu

This command functions similar to the **Insert branch right** command.

Also refer to

- [chapter "SFC element "Branching"" on page 249](#)
- [chapter ""Insert branch right" command" on page 644](#)

"Insert branch right" command

Symbol: 

Function: The command inserts a branching to the right of the selected point.

Call: SFC menu

The type of the inserted branching depends on the selected element:

- If the top element of the selected elements is a transition or an alternative branch, then IndraLogic inserts an alternative branch.
- If the top element of the selected elements is a step, a macro, a jump, or a parallel branch, then IndraLogic inserts a parallel branch with the Branch<x> jump marker, where x is an incremental number.

This default name of the jump marker can be edited.

The jump marker can be specified as the target of a jump.

- If a common element of an existing branch (horizontal line) is selected, then IndraLogic inserts the new branch line as a branch line on the far right.

If an entire branch line of an existing branch is selected, then IndraLogic inserts the new branch line directly to the right as a new branch line.



Please note: You can convert a branch into another type with the **Alternative** and **Parallel** commands.

Example:

Parallel branch:

The following image shows a new inserted parallel branch generated by the **Insert branch right** command while the Step11 step was selected. IndraLogic automatically inserts a step (Step2 in the example).

Processing in online mode: If t2 provides TRUE, IndraLogic executes Step2 immediately after step11, before t3 is analyzed.

In this case, in contrast to alternate branches, IndraLogic executes both branchings.

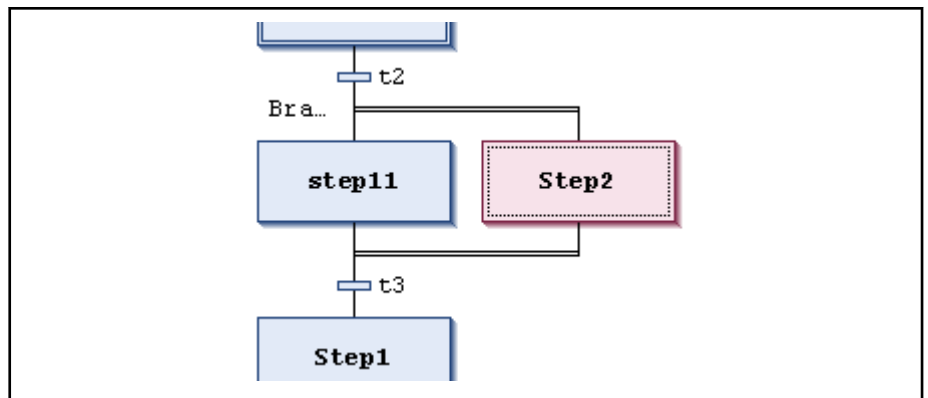


Fig. 18-66:

*Example:***Alternative branch**

The following image shows a new inserted alternative branch generated by the **Insert branch right** command while the t4 transition was selected. IndraLogic automatically inserts a step (Step32 in the example), a preceding and a following transition (t41, t42).

Processing in online mode: When Step3 is active, IndraLogic evaluates the subsequent transitions (t4, t41) from left to right. The first branch of the branching in which the first transition delivers the value **TRUE** is executed. In this way, in contrast to a parallel branch, only one branch is executed.

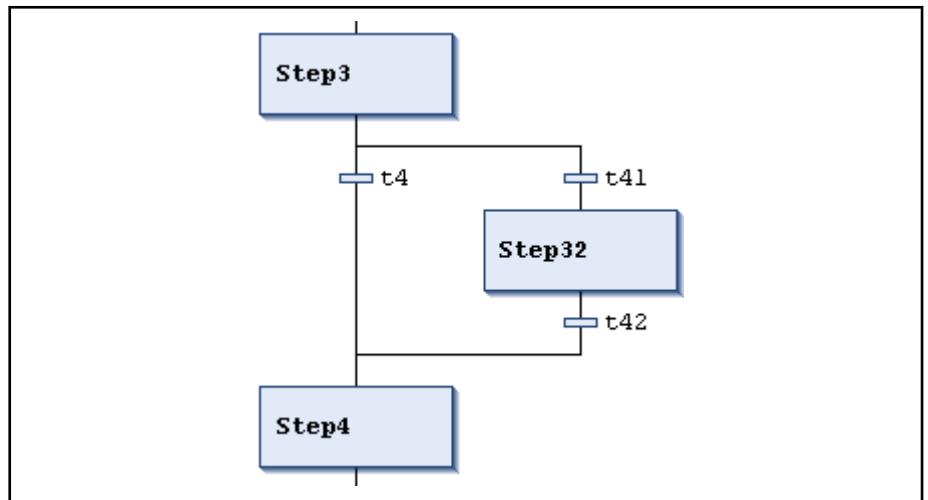


Fig. 18-67:

Also refer to

- [chapter "SFC element "Branching"" on page 249](#)
- [chapter ""Insert branch" command" on page 644](#)
- [chapter ""Parallel" command" on page 643](#)
- [chapter ""Alternative" command" on page 643](#)

"Insert action association" command

Symbol:

Function: The command assigns an IEC action to a step.

Call: SFC menu

Prerequisite: A step is selected.

IndraLogic inserts the action element to the right of the selected step element.

If the step has been assigned one or more actions, they are displayed in an "actions list".

The new action is positioned as follows:

- If the step element is selected, it is positioned as the first action of the step, i.e. in the top position in the action list.
- If one of the actions in the action list of the step is selected directly in front, i.e. above the action.

The left section of the action element includes the qualifier (N by default), and the action name is typed in the right section. To do this, click in the box to obtain an editing frame. This action has to be created before as POU in the project.

The qualifier can also be edited. A list of valid qualifiers is described under "Qualifiers for actions in SFC".

Also refer to

- [chapter ""Insert action association after" command" on page 646](#)
- [chapter "Qualifiers for SFC actions" on page 239](#)

"Insert action association after" command

Symbol: 

Function: The command assigns an IEC action to a step.

Call: SFC menu

Prerequisite: A step is selected.

The command corresponds to the **Insert action association** description. The difference is that IndraLogic does not place the new action in the first position, but instead in the last position in the actions list.

When an action is selected in the selection list, IndraLogic position the new action below and not above.

Also refer to

- [chapter ""Insert action association" command" on page 645](#)
- [chapter "Qualifiers for SFC actions" on page 239](#)

"Insert jump" command

Symbol: 

Function: The command inserts a jump element before the selected element.

Call: SFC menu

Prerequisite: A step is selected.

IndraLogic automatically inserts the jump with the `Step` destination. This jump target has to be subsequently replaced by a real jump target. Select the target using the input assistant.

Also refer to

- [chapter "SFC element "Jump"" on page 251](#)
- [chapter ""Insert jump after" command" on page 647](#)

"Insert jump after" command

Symbol: 

Function: The command inserts a jump element after the selected element.

Call: SFC menu

IndraLogic automatically inserts the jump with the `Step` destination. This jump target has to be subsequently replaced by a real jump target. Select the target using the input assistant.

Also refer to

- [chapter "SFC element "Jump"" on page 251](#)
- [chapter ""Insert jump" command" on page 646](#)

"Insert macro" command

Symbol: 

Function: The command inserts a macro element before the selected element.

Call: SFC menu

The new macro is named `Macro<x>` by default, where `x` is an incremental number beginning at 0 for the first macro. Click on the name to directly edit the default name.

To edit the macro, click **Zoom into macro** in the macro editor.

Also refer to

- [chapter ""Insert macro after" command" on page 647](#)
- [chapter ""Zoom into macro" command" on page 647](#)

"Insert macro after" command

Symbol: 

Function: The command inserts a macro element after the selected element.

Call: SFC menu

The command corresponds to the **Insert macro** description.

Also refer to

- [chapter ""Insert macro" command" on page 647](#)
- [chapter ""Zoom into macro" command" on page 647](#)

"Zoom into macro" command

Symbol: 

Function: The command opens a macro in the macro editor for processing.

Call: SFC menu

Prerequisite: A macro is selected.

IndraLogic uses this command to close the main view of the SFC editor and the macro editor instead. This is also an SFC editor for editing the section of the SFC diagram that is displayed as a macro box in the main view.

Click **Zoom out of macro** to return to the main view.

Also refer to

- [chapter ""Zoom out of macro" command" on page 648](#)

"Zoom out of macro" command

Symbol: 

Function: The command closes the macro editor and returns to the main view of the SFC editor.

Call: SFC menu

Prerequisite: A macro is open in the macro editor.

Also refer to

- [chapter ""Zoom into macro" command" on page 647](#)

'Insert after' command

Symbol: 

Function: The command pastes the elements from the clipboard after the selected point.

Call: SFC menu

18.2.8 Menu 'CFC'

"Edit worksheet" command

Function: The command opens the **Edit worksheet** dialog to define the size of the worksheet.

Call: CFC menu

Prerequisites: A CFC editor is active.

"Edit worksheet" dialog

Use the following dimensions	The size of the worksheet can be set here. The change is only applied if the size of the existing program is sufficient.
Adapt dimensions automatically	Customizes the size of the worksheet automatically to the size of your program.
Move the working sheet origin relatively	Moves the worksheet on the x-axis or y-axis. Entering negative numbers is allowed.

"Edit page size" command

Function: The command opens the **Edit page size** dialog. The size of the page-oriented CFC editor can be adjusted here.

Call: CFC menu

Prerequisites: A page-oriented CFC editor is active.

"Edit page size" dialog

Width	Page width (minimum 24, maximum 1024). Elements outside the workspace are highlighted in red.
Height	Page height (minimum 24, maximum 1024). Elements outside the workspace are highlighted in red.
Margin width	Margin width of the column (minimum 6, maximum 25% of the page width).
Set as default for new CFC objects	 The current settings are selected as default settings for new CFC objects.

Also refer to

- [chapter "CFC element "Page"" on page 277](#)

“Negate” command

Symbol: 

Function: The command negates the selected function block input or function block output.

Call: Menu **CFC**, context menu

Prerequisites: A CFC editor is active. A function block input or function block output is selected.

“Command EN/ENO” command

Symbol: 

Function: The command adds a Boolean input **EN** (Enable) and a Boolean output **ENO** (Enable Out) to the selected function block.

Call: Menu **CFC**, context menu

Prerequisites: A CFC editor is active. A function block is selected.

The added input **EN** enables the function block. The function block is executed when the value becomes **TRUE**. The value of this signal is output at the **ENO** output.

“None” command

Symbol: 

Function: The command removes a Reset (R), Set (S) or REF (reference assignment) from the input of the **Output** element.

Call: Menu **CFC** ► **Set/reset**, context menu **Set/reset**

Prerequisites: A CFC editor is active. The input of an **Output** element is selected.

“R-Reset” command

Symbol: 

Function: The command adds a reset to the input of a Boolean element **Output**.

Call: Menu **CFC** ► **Set/reset**, context menu **Set/reset**

Prerequisites: A CFC editor is active. The input of an **Output** element is selected.

If an element **Output** is provided with an “Reset” input, the Boolean output value is set to **FALSE**, when the value of the input becomes **TRUE**. **FALSE** remains at the output even if the input value changes again.

Also refer to

- [chapter “S-SET” command on page 649](#)
- [chapter “None” command on page 649](#)

“S-SET” command

Symbol: 

Function: The command adds a Set (S) to the input of a Boolean element **Output**.

Call: Menu **CFC** ► **Set/reset**, context menu **Set/reset**

Prerequisites: A CFC editor is active. The input of an **Output** element is selected.

If an element **Output** is provided with an "Set" input, the Boolean output value is set to **TRUE**, when the value of the input becomes **TRUE**. **TRUE** remains at the output even if the input value changes again.

Also refer to

- [chapter "'R-Reset" command" on page 649](#)
- [chapter "'None" command" on page 649](#)

"Ref= (reference assignment)" command

Symbol: 

Function: The command assigns a reference to an **Output** element.

Call: Menu **CFC ► Set/reset**, context menu **Set/reset**

Prerequisites: A CFC editor is active. The input of an **Output** element is selected.

Example:

Declaration:

```
ref_int : REFERENCE TO INT;
a : INT;
```

CFC:

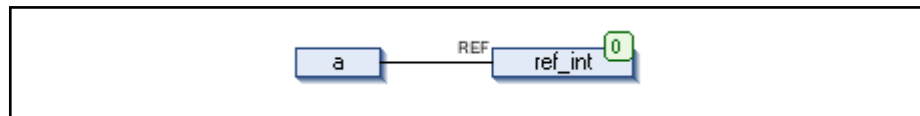


Fig. 18-68:

This corresponds to the ST code `ref_int REF= a;`

For further information, refer to the description on the `REFERENCE TO` data type.

Also refer to

- [chapter "Programming in the CFC editor" on page 97](#)
- [chapter 17.5.13 "Data type 'REFERENCE'" on page 361](#)
- [chapter "'None" command" on page 649](#)

Command 'Send to front'

Symbol: 

Function: The command sets the selected elements to the beginning of the execution sequence.

Call: Menu **CFC ► Execution order**, context menu **Execution order**

Prerequisites: One CFC editor is active and the "explicit execution order mode" property is selected. At least one element is selected.

The selected elements get the lowest numbers beginning at 0 while keeping the previous order. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

Also refer to

- [chapter "'Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to back'Symbol:" on page 651](#)

- [chapter "Command 'One ahead'" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

Command 'Send to back' Symbol:

Symbol: 

Function: The command numbers the elements so that the selected elements are located at the end of the execution order.

Call: Menu **CFC ▶ Execution order**, optionally in the editor context menu **Execution order**

Prerequisites: One CFC editor is active und "The explicit execution order mode" property is selected. At least one element is selected.

The selected elements get the highest numbers while keeping the previous order. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

Also refer to

- [chapter ""Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to front'" on page 650](#)
- [chapter "Command 'One ahead'" on page 651](#)
- [chapter "Command 'One back'" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

Command 'One ahead'

Symbol: 

Function: The command numbers the elements so that the selected elements are located one position forward.

Call: Menu **CFC ▶ Execution order**, optionally in the editor context menu **Execution order**

Prerequisites: A CFC editor is active and at least one element is selected. The "Explicit Execution Order Mode" property is selected.

The selected elements get a numbering decreased by one while keeping the previous order. The selected elements are processed one position earlier. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

Also refer to

- [chapter ""Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to front'" on page 650](#)
- [chapter "Command 'Send to back' Symbol:" on page 651](#)
- [chapter "Command 'One back'" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

Command 'One back'

Symbol: 

Function: The command numbers the elements so that the selected elements are located one position backward.

Call: Menu **CFC ► Execution order**, optionally in the editor context menu **Execution order**

Prerequisites: A CFC editor is active and at least one element is selected. The "Explicit Execution Order Mode" property is selected.

The selected elements get a numbering increased by one while keeping the previous order. The elements are processed one position later. The remaining elements are numbered so that their execution order remains the same. The topological positions of the elements are retained anyway.

Also refer to

- [chapter ""Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to front'" on page 650](#)
- [chapter "Command 'One ahead'" on page 651](#)
- [chapter "Command 'Send to back'Symbol:" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

Command 'Order by data flow'

Function: The command numbers the elements in the program by data flow, or in the case of multiple networks by their topological position in the editor.

Call: Menu **CFC ► Arrange by data flow**, optionally in the editor context menu **Arrange by data flow**

Prerequisites: One CFC editor is active und "The explicit execution order mode" property is selected.

The execution order is determined by data flow. In the case of multiple networks, it is determined by their topological position of the networks. All numbered elements of the POU are set accordingly. Afterwards, the execution order is identical to that in auto data flow mode. The topological positions of the elements are retained anyway.

Also refer to

- [chapter ""Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to front'" on page 650](#)
- [chapter "Command 'Send to back'Symbol:" on page 651](#)
- [chapter "Command 'One ahead'" on page 651](#)
- [chapter "Command 'One back'" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

Command 'Order by topology'

Function: The command orders the execution order of the elements by their topological position from right to left and from top to bottom.

Call: Menu **CFC ► Arrange topologically**, optionally in the editor context menu **Arrange topologically**

Prerequisites: One CFC editor is active und "The explicit execution order mode" property is selected. At least one element is selected.

The command applies to all elements in the program, even if not all elements are selected when the command is executed. The topological positions of the elements are retained anyway.

Also refer to

- [chapter ""Properties" dialog - "CFC execution order" tab" on page 709](#)
- [chapter "Command 'Send to front'" on page 650](#)
- [chapter "Command 'Send to back' Symbol:" on page 651](#)
- [chapter "Command 'One ahead'" on page 651](#)
- [chapter "Command 'One back'" on page 651](#)
- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

"Set execution order" command

Function: The command opens a dialog to set the execution position of an element to an arbitrary value.

Call: Menu **CFC** ► **Set execution order**, context menu **Set execution order**

Prerequisites: A CFC editor is active. An element is selected.

The command only changes the chronological order in the program sequence. The position of the elements is not changed. If the command is used for multiple elements at the same time, the order within the selected elements remain.

Also refer to

- [chapter "Command 'Order by data flow'" on page 652](#)
- [chapter "Command 'Order by topology'" on page 652](#)

"Edit parameters" command

Function: The command opens the **Edit Parameters** dialog, where you change the constant input parameters of a function block.

Call: Menu **CFC** ► **Edit parameters**, or **right-click** ► **Edit parameters**, click the **Parameter** function block.

Prerequisites: A CFC editor is active. An instantiated function block has VAR_INPUT CONSTANT variables in its declaration.



This functionality applies only to blocks that are inserted in a CFC with IndraLogic >= V3.5 SP4!

IndraLogic displays blocks with VAR_INPUT CONSTANT variables by the word **Parameter** in the lower left corner of the block.

"Edit parameters" dialog

Parameters	Variable name
Data type	Variable data type
Value	Click into the field to enter a value.
Initial value	Initialization value

Category	Additional information on the parameters. These values are defined by attributes and cannot be changed in this dialog
Unit	
Min	
Max	
Delete prepared parameters	The command is active when you write a prepared value (Debug ► Write value)

When you exit the field and the dialog by clicking **OK**, the value changes are applied to the project.

Example of a function block with constant inputs

```
FUNCTION_BLOCK FB1
VAR_INPUT CONSTANT
    {attribute 'parameterCategory':='General'}
    {attribute 'parameterUnit':='m/s'}
    {attribute 'parameterMinValue':='0'}
    {attribute 'parameterMaxValue':='100'}
    fbin1:INT;
    fbin2:DWORD:=24354333;
    fbin3:STRING:='abc';
END_VAR
```



This functionality and the declaration of variables with keyword `VAR_INPUT CONSTANT` applies only to the CFC editor. In the FBD editor, IndraLogic always shows all input parameters on the block, regardless of whether they are declared as `VAR_INPUT` or `VAR_INPUT CONSTANT`. IndraLogic does also not make a distinction about this in text editors.

Also refer to

- ["Changing constant input parameters of function block instances" on page 275](#)
- [chapter ""Save prepared parameters to project" command" on page 654](#)

"Save prepared parameters to project" command

Function: The command applies the prepared parameter values to the project.

Call: CFC menu

Prerequisites: A CFC editor is active. Parameter values of function block instances are changed in online mode. You are in offline mode.

If the values of constants on the control are different from the values in the application, then this is indicated by a red asterisk next to the parameter field. Clicking **Incur Prepared Parameters** saves the control values to the application.

Also refer to

- ["Changing constant input parameters of function block instances" on page 275](#)
- [chapter ""Edit parameters" command" on page 653](#)

"Connect selected connections" command

Symbol:

Function: The command establishes a connection between the selected connections.

Call: Menu **CFC ▶ Connect selected connections**, context menu **Connect selected connections**

Prerequisites: A CFC editor is active. One output and multiple inputs are selected.

To select the connections, hold down the <Ctrl> key when clicking on the connections. Then, execute the command.

Also refer to

- [chapter ""Select linked connections" command" on page 655](#)

"Unlock connection" command

Symbol: 

Function: This command unlocks a locked connection.

Call: **CFC ▶ Unlock connection**, context menu **Unlock connection**

Prerequisites: A CFC editor is active. A connection or a connection mark is selected.

There is a locked connection if the connections of the automatic routing are changed. To automatically route before, delete a locked connection.



Click on the  icon of a locked connection and this connection can also be unlocked.

Also refer to

- [chapter "Programming in the CFC editor" on page 97](#)
- [chapter "CFC element "Connection mark source/target"" on page 279](#)

"Show next collision" command

Function: The command shows the next collision in the editor and highlights the respective location.

Call: Menu **CFC ▶ Routing**, context menu **Routing**

Prerequisites: A CFC editor is active and there is at least one connection with collision.

This function is very useful when working with big networks and if you also see a section. A collision is additionally signaled with a red-framed icon in the right upper corner of the editor.

"Select linked connections" command

Symbol: 

Function: The command selects connections either connected to the currently selected line or to the currently selected connection mark in the site-oriented CFC.

Call: Menu **CFC ▶ Select linked connections**, context menu **Select linked connections**

Prerequisites: Either a CFC editor or a page-oriented CFC editor is active. A connection or a connection mark is selected.

Also refer to

- [chapter ""Connect selected connections" command" on page 654](#)

“Reset connections” command

Symbol: 

Function: The command restores the deleted connections of a function block.

Call: Menu **CFC ▶ Reset connections**, context menu **Reset connections**

Prerequisites: A CFC editor is active and a function block is selected.

The command restores all inputs and outputs of the function block as defined in its implementation.

Also refer to

- [chapter “Remove unconnected connections” command on page 656](#)

“Remove unconnected connections” command

Symbol: 

Function: The command removes all unused connections of the selected element.

Call: Menu **CFC ▶ Remove disconnected connections**, context menu **Remove disconnected connections**

Prerequisites: A CFC editor is active. An element is selected.

Also refer to

- [chapter “Reset connections” command on page 656](#)

“Add input connection” command

Symbol: 

Function: The command adds another input to the selected function block.

Call: Can be configured via **Tools ▶ Customize ▶ Commands ▶ CFC**

Prerequisites: A CFC editor is active. A function block is selected.

Also refer to

- [chapter “Add output connection” command on page 656](#)

“Add output connection” command

Symbol: 

Function: The command adds another output to the selected function block.

Call: Can be configured via **Tools ▶ Customize ▶ Commands ▶ CFC**

Prerequisites: A CFC editor is active. A suitable function block is selected.

Also refer to

- [chapter “Add input connection” command on page 656](#)

“Route all connections” command

Symbol: 

Function: The command undoes all manual changes at the connections in the program and restores the original state.

Call: Menu **CFC ▶ Route all connections**, context menu **Route all connections**

Prerequisites: A CFC editor is active.

IndraLogic cannot automatically route connections fixed with control points. Remove the control points before executing the command. Use the command **Remove control point**. Manually changed connections with the icon  have to be unlocked. Use the command **Unlock connection**.

Also refer to

- [chapter ""Remove control point" command" on page 657](#)
- [chapter ""Unlock connection" command" on page 655](#)

"Remove control point" command

Function: The command removes a control point.

Call: CFC ► **Remove control point** or context menu **Remove control point**

Prerequisites: A CFC editor is active. A connecting line is selected.

When moving the mouse pointer over a selected connecting line, the existing control points are displayed with yellow circle symbols. Set the cursor to the control point to be deleted and execute the command from the context menu.

Also refer to

- [chapter "CFC element "Control point"" on page 277](#)
- [chapter ""Create control point" command" on page 657](#)

"Create control point" command

Symbol: 

Function: The command generates a control point at a connector.

Call: CFC ► **Create control point** or context menu **Create control point**

Prerequisites: A CFC editor is active. The cursor is on a connection.

The control point is created at the at the cursor position of the connection when the command is called. The command corresponds to the **Control point element** in the **Tools** window.

Also refer to

- [chapter "CFC element "Control point"" on page 277](#)
- [chapter ""Remove control point" command" on page 657](#)

"Connection mark" command

Symbol: 

Function: The command toggles the representation of the connection between two elements and the use of connection marks.

Call: CFC ► **Connection mark**, context menu **Connection mark**

Prerequisites: A CFC editor is active. A connection or a connection mark is selected.

After selecting a connecting line, the command removes this line and adds a **Connection mark - Source** to the output of one element and a **connection mark - Target** to the input of the other element. Both are equally named "C-<n>". n is a consecutive number.

When selecting a connection mark pair, the command converts these marks into a connecting line.

Also refer to

- [chapter "CFC element "Connection mark source/target"" on page 279](#)

“Create group” command

Symbol: 

Function: The command groups the selected elements.

Call: Configure via **Tools ▶ Customize ▶ Commands ▶ CFC**

Prerequisites: A CFC editor is active. Multiple elements are selected.

Grouped elements can only be moved together. The position of the elements is not affected by the grouping.

Also refer to

- [chapter “Ungroup” command on page 658](#)

“Ungroup” command

Symbol: 

Function: The command removes a previously created grouping.

Call: Configure via **Tools ▶ Customize ▶ Commands ▶ CFC**

Prerequisites: A CFC editor is enabled. A grouping is selected.

Also refer to

- [chapter “Create group” command on page 658](#)

Command 'Prepare function block element for forcing'

Function This command activates and deactivates the forceability of the inputs for a function block element.

Call The menu is not part of the standard command block of IndraWorks and can be moved to the symbol bar via the IndraWorks menu **Tools ▶ Adjust ▶ Commands ▶ CFC**.

Prerequisites The CFC editor is in offline mode and a function block element is selected. After executing the command, the "command Force FB input" is available in online mode to open a dialog for forcing the block input values.

Also refer to:

- [chapter “Force FB input’ command” on page 658](#)
- [chapter 18.2 “Menu commands” on page 572](#)
- [chapter 12.4 “Forcing and writing variables” on page 183](#)

‘Force FB input’ command



This kind of forcing uses a data breakpoint internally and is therefore different from forcing with the "Force values" command or [F7].

Values that were forced by the command "Force FB input" do not respond to the commands "Show all forces" or "Unforce values".

Function This command opens the "Force value" dialog to force the selected input of a function block. Forcing can be canceled with the same command and dialog.

Call: The menu is not part of the standard command block of IndraWorks and can be moved to the symbol bar via the IndraWorks menu **Tools ▶ Adjust ▶ Commands ▶ CFC**.

Prerequisites The "Forcibility" of the function block is enabled via the command "Prepare function block element for forcing". The CFC editor is in online mode and the function block input is selected.

In the "Force value" dialog, either specify a value that the input of the function block should be forced, or remove the currently forced value.

After forcing, the input is highlighted in green again. Boolean inputs are assigned a small monitoring window with the forced value. The forced value is displayed in the "Value" column of monitoring views (in the declaration part of the programming function block or in a watch list).

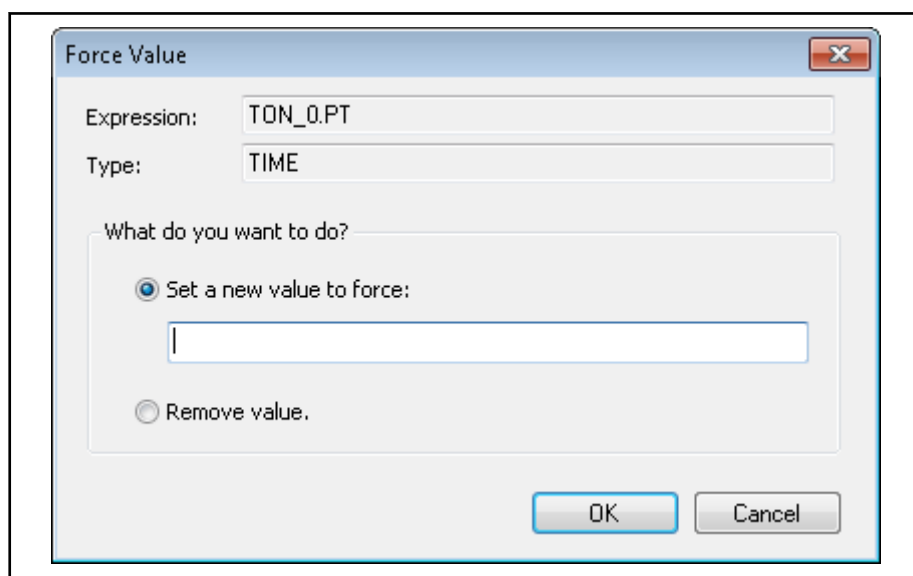


Fig. 18-69: Dialog "Force value"

Dialog 'Force value'

Expression	Name of the function block input. Example: TON_0.PT.
Type	Data type of the input

"What do you want to do?"

Set a new value to force	 : Specify a new value in the input field. The format has to correspond to the data type.
Remove value	 : Forcing at the input is canceled.

Also refer to:

- [chapter 18.2 "Menu commands" on page 572](#)
- [chapter 12.4 "Forcing and writing variables" on page 183](#)
- [chapter "Command 'Prepare function block element for forcing'" on page 658](#)

Command 'Use attributed member as input'

Symbol 

Function The command allows for connecting a structure member to a scalar type input.

Call: Menu CFC ► Pins.

Via the context menu in the CFC editor.

Prerequisites

A CFC editor is active and a function block input is selected

The member of the structure that is connected to the input of the subsequent function block must be provided with the pragma {attribute 'ProcessValue'}.

The data type of the structure member has to be compatible with the data type of the subsequent input. Inputs connected in this way are flagged with the "V" symbol.

Example: Data type declaration "QINT" with attribute

```
TYPE QINT :
STRUCT
    Status : STRING;
    {attribute 'ProcessValue'}
    Value1 : INT;
    Value2 : INT;
END_STRUCT
END_TYPE
```

Example: Declaration in the program and highlighting of "intValue" input

```
PROGRAM PLC_PRG
VAR
    input1: QINT;
    output1: QINT;
    intValue: INT;
END_VAR
```

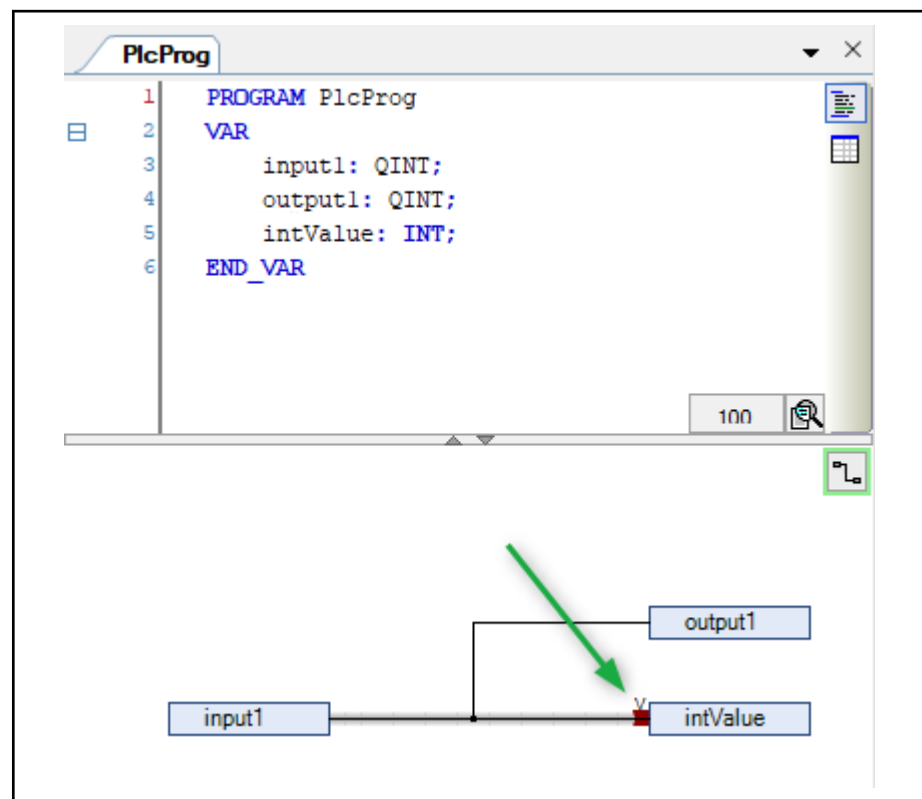


Fig. 18-70:

Also refer to

- chapter "Attribute 'ProcessValue'" on page 412

18.2.9 Menu 'FBD/LD/IL'

"Insert network" command

Icon: , shortcut: <Ctrl>+<Shift>+

Function: this command inserts a further network in the FBD/LD/IL editor.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL element "Network"" on page 263](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Insert network below" command" on page 661](#)

"Insert network below" command

Icon: , Shortcut: <Ctrl>+<Shift>+<A>

Function: this command inserts a further network in the FBD/LD/IL editor below the selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Network"" on page 263](#)
- [chapter ""Insert network" command" on page 661](#)

"Comment in/out" command

Icon: , shortcut: <Ctrl>+<Shift>+<C>

Function: The command comments the selected network in or out.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert assignment" command

Symbol , shortcut: <Ctrl> + <A>

Function: The command inserts an assignment in the FBD or LD editor.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected, but no function block is selected.



In IL an assignment is programmed via the operators **LD** and **ST**.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

'Insert function block' command

Symbol: , Shortcut: <Ctrl> +

Function: The command inserts a function block that is available in the project at the end of the selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected, but no function block is selected.

Select the command and the input assistant opens. Select the function block there.

Also refer to

- [chapter "FBD/LD/IL element "Function block"" on page 264](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter 9.5 "Using input assistants" on page 107](#)

"Insert function block with EN/ENO" command

Symbol: , shortcut: <Ctrl> + <Shift> + <E>

Function: The command inserts a function block with a Boolean input **ENable** and a Boolean output **ENable Out** at the end of the selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected, but no function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter 9.5 "Using input assistants" on page 107](#)

"Insert empty function block" command

Symbol: 

Function: this command inserts an empty function block at the end of the currently selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A network is selected, but no function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert function block with EN/ENO" command

Symbol: 

Function: The command inserts an empty function block with a Boolean input **ENable** and a Boolean output **ENable Out** at the end of the selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: AN FBD editor, an LD editor or an IL editor is active. A network has to be selected. No other function block may be selected.

If at the time of the function block call **Enable** has the value **FALSE**, the operations defined in the function block are not executed. Otherwise, i.e. if **Enable** has the value **TRUE**, these operations are executed. The ENO output acts as repeater of the EN input.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert jump" command

Symbol , shortcut: <Ctrl>+<J>

Function: The command inserts a jump element before the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A connecting line is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Jump"" on page 265](#)

"Insert label" command

Symbol: 

Function: The command inserts a jump label into the currently selected network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD, LD or IL editor is active. A network is selected. No jump label is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Jump label"" on page 265](#)

"Insert return" command

Symbol: 

Function: this command inserts an element **Return** in the selected place.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or IL editor is active. A function block output is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Return"" on page 265](#)

"Insert input" command

Symbol: , shortcut: <Ctrl>+<Q>

Function: The command adds a further input to an extendable function block (AND, OR, ADD, MUL, SEL) above the selected input.

Call: **FBD/LD/IL** menu

Prerequisites: The FBD or LD editor is active. A function block input is selected.

If a function block is selected, the command **Append input** is available in the context menu. The input is inserted at the lower end of the function block.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert coil" command

Symbol: , Shortcut: <Ctrl> + <I>

Function: The command inserts a coil into the network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A network, a coil or a connector is selected. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Coil"" on page 267](#)

"Insert set coil" command

Symbol: 

Function: The command inserts a set coil into the network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A network, a coil or a line is selected. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert reset coil" command

Symbol: 

Function: The command inserts a reset coil into the network.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A network, a coil or a line is selected. No function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Coil"" on page 267](#)

"Insert contact" command

Symbol  shortcut: <Ctrl> + <K>

Function: The command inserts a contact to the left of the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line or a contact is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert contact (right)" command

Symbol: , shortcut: <Ctrl> + <R>

Function: The command inserts a contact to the right of the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line, a contact or a function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert contact in parallel (below)" command

Symbol: , shortcut: <Ctrl> + <D>

Function: The command inserts a contact with lines in parallel with and below the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line or a contact or a function block is selected.



Closed parallel branchings in an LD network can be programmed as "Short Circuit Evaluation" (SCE) or OR constructs. SCE branches are shown with double vertical lines, OR branches are with single lines. Refer to the help site "Closed line branchings".

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter ""Insert contact in parallel \(above\)" command" on page 665](#)
- [chapter "Closed line branching" on page 268](#)

"Insert contact in parallel (above)" command

Symbol: , shortcut: <Ctrl> + <E>

Function: The command inserts a contact with lines in parallel with and above the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line, a contact or a function block is selected.



Closed parallel branchings in an LD network can be programmed as "Short Circuit Evaluation" (SCE) or OR constructs. SCE branches are shown with double vertical lines, OR branches are with single lines. Refer to the help site "Closed line branchings".

Also refer to

- [chapter "LD element "Contact"" on page 266](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "Closed line branching" on page 268](#)

'Toggle parallel mode' command

Function: This command toggles a parallel branch between an OR construct and the Short Circuit Evaluation (SCE) .

Call: The menu is not part of the standard command block of IndraWorks and can be moved to the symbol bar via the IndraWorks menu **Tools** ► **Adjust** ► **Commands** ► **FBD/LD/IL**.

Prerequisites: The LD editor is active. A vertical line of a parallel branch is selected.



Closed parallel branchings in an LD network can be programmed as "Short Circuit Evaluation" (SCE) or OR constructs. SCE branches are shown with double vertical lines, OR branches are with single lines. Refer to the help site "Closed line branchings".

Also refer to:

- [chapter 18.2.9 "Menu 'FBD/LD/IL'" on page 661](#)
- [chapter "Closed line branching" on page 268](#)

"Insert negated contact" command

Symbol: 

Function: The command inserts a negated contact to the left of the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line or a contact is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert negated contact in parallel (below)" command

Symbol: 

Function: The command inserts a negated contact with lines in parallel with and below the selected element.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line, a contact or a function block is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert contacts" command: "Insert below"

Shortcut: <Ctrl>+<Shift>+<D>

Function: The command inserts a previously copied contact with lines below the selected element.

Call: Menu **FBD/LD/IL** ► **Insert contacts**, context menu

Prerequisites: The LD editor is active.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert contacts" command: "Insert above"

Shortcut: <Ctrl>+<Shift>+<E>

Function: The command inserts a previously copied contact with lines above the selected element.

Call: Menu **FBD/LD/IL** ► **Insert contacts**, context menu

Prerequisites: The LD editor is active. A line or a contact is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert contacts" command: Add right (after)

Shortcut: <Ctrl>+<Shift>+<R>

Function: The command inserts a previously copied contact to the right of the selected element.

Call: Menu **FBD/LD/IL** ► **Insert contacts**, context menu

Prerequisites: The LD editor is active. A line or a contact is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "LD element "Contact"" on page 266](#)

"Insert IL line after" command

Symbol: 

Function: The command inserts an instruction line below the selected line.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The IL editor is active. A line is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Delete IL line" command

Symbol: , shortcut: <Ctrl>+<Del>

Function: The command deletes the selected instruction line.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The IL editor is active. A line is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Negation" command

Symbol: , shortcut: <Ctrl>+ <T>

Function: The command negates the following elements:

- Input/output of a function block
- Jump
- Return
- Coil

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. The corresponding element is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Edge detection" command

FBD symbol: , LD symbol: , shortcut: <Ctrl> + <N>

Function: The command inserts an edge detector before the selected function block input or function block output.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. A function block input or function block output is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Set/reset" command

Symbol: , shortcut <Ctrl>+ <Shift><S>

Function: In the case of an element with a Boolean output, this command switches between reset, set and no mark.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. An element with a Boolean output is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Set output connection" command

Symbol: , shortcut <Ctrl> + <Shift> + <W>

Function: The command turns the selected function block output into the forwarding function block output.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. One of several function block outputs is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert line branching" command

Symbol: , shortcut <Ctrl>+ <Shift> + <V>

Function: The command creates an open line branch on the selected line.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. An input or an output of a function block is selected.

Also refer to

- [chapter "FBD/LD/IL element "Line branching"" on page 265](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)

"Insert line branching above" command

Symbol: 

Function: The command inserts a line branch above the selected open line branch.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. An open line branching is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Line branching"" on page 265](#)

Command "Insert line branching below" command

Symbol: 

Function: The command inserts a line branch below the selected open line branch.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. An open line branching is selected.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "FBD/LD/IL element "Line branching"" on page 265](#)

"Set line branching start point" command

Symbol: 

Function: The command sets the starting point of a line branch on the selected line.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line is selected.

Also refer to

- [chapter "Closed line branching" on page 268](#)
- [chapter "FBD/LD/IL Editor" on page 254](#)

"Set line branching end point" command

Function: The command sets the end point of a line branching at the selected line.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The LD editor is active. A line is selected. A starting point of the line branching was set.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)
- [chapter "Closed line branching" on page 268](#)

"Update parameters" command

Function: The command enters changes to the declaration of the selected element in the diagram.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: FBD, LD or CFC editor is active. A function block is selected. An extending change has been made to the declaration.

The command checks whether a function block and its declaration in the declaration editor correspond. The change is accepted for the function block only if the declaration was extended. Deletions and overwrites are not updated.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Remove unused FB call parameters" command

Symbol: 

Function: The command deletes inputs and outputs of the selected function block to which no variable and no value were assigned. However, the default inputs and outputs are always retained.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. A function block is selected. The function block is provided with interfaces without assigned values.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Repair block" command

Symbol: 

Function: The command repairs internal inconsistencies in the selected function block.

Call: Menu **FBD/LD/IL**, context menu

Prerequisites: The FBD or LD editor is active. The defective function block is selected. The editor detected internal inconsistencies in the programming function block that can possibly be resolved automatically. IndraLogic reports inconsistencies in the message window.

This situation is conceivable when editing a project that was created with an older programming system version that did not yet handle the inconsistency concerned as an error.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Display as function block diagram" command

CAUTION

Loss of data!

An error-free conversion requires syntactically correct code. Otherwise, parts of the implementation might be lost.

Shortcut: <Ctrl>+<Shift>+<F>

Function: The command converts the active instruction list or the active ladder diagram into the function block diagram.

Call: Menu **FBD/LD/IL** ► **View**

Prerequisites: LD or IL editor is active.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Display as ladder diagram" command

CAUTION

Loss of data!

An error-free conversion requires syntactically correct code. Otherwise, parts of the implementation might be lost.

Shortcut: <Ctrl>+<Shift>+<L>

Function: The command converts the current function block code or the active instruction list into a ladder diagram.

Call: Menu **FBD/LD/IL ▶ View**

Prerequisites: The FBD or IL editor is active.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Display as instruction list" command



Enable the IL in the IndraWorks options if required.

Tools ▶ Options ▶ IndraLogic 2G ▶ FBD, LD and IL ▶ IL ▶ Enable IL

CAUTION

Loss of data!

An error-free conversion requires syntactically correct code. Otherwise, parts of the implementation might be lost.

Shortcut: <Ctrl> + <Shift> + <I>

Function: The command converts the active function block code or the active ladder diagram into an instruction list.

Call: Menu **FBD/LD/IL ▶ View**

Prerequisites: The LD or FBD editor is active.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

"Go to" command

Symbol:

Function: The command allows you to jump to any network.

Call: **FBD/LD/IL** menu

Prerequisites: the LD, FBD or IL editor is active. A network is selected.

The command opens a dialog with an input field. Enter the number of the desired network in the input field.

Also refer to

- [chapter "FBD/LD/IL Editor" on page 254](#)

18.2.10 Menu 'Library'

The commands for the library manager can be called from its editor window (library manager).

If required, the menu configuration can be changed in the dialog under **Tools ▶ Customize**.

“Add library” command

Function: The command opens the **Add library** dialog. In this dialog, you can add libraries to the library manager and then link them to your application.

Prerequisite: The library manager is open in the editor.

Call: Menu **Library manager** ► **Add library** or **Add library** in the header of the editor

“Add library” dialog

In the line above the library list, you can search for library names or library function blocks by typing an appropriate string.	
Library	List all the libraries installed in the library repository grouped in categories.
Company	Library creator.
Advanced	Opens the advanced “Add library” dialog
	The displayed libraries are grouped into categories
	The displayed libraries are listed in alphabetical order
	All available libraries are displayed



Specific libraries can be “blacklisted” in a device description. These libraries cannot be added below this device in the library manager.

Also refer to

- [chapter “Library repository” command on page 636](#)

“Add library” - “Advanced” dialog

You should use this dialog only if you have expert knowledge of library referencing. Using this dialog, you can link special versions or change placeholder definitions.



We recommend seriously that you follow the guidelines for the creation of libraries when developing and referencing libraries.

“Library” tab

Company	Filtering the list according to vendor.
Group by category	☒ : Display of the libraries in a tree structure grouped in categories. ☐ : Display of the libraries in alphabetical order in a flat structure.
Display all versions	☒ : Display of all versions of the libraries. Version specification “*” means the latest version available in the repository. ☐ : Display of only the latest versions of the libraries. A multiple selection of libraries is possible in this display: Keep the <Shift> key pressed while selecting the entries.
Details	Opens a detailed view with the library function block.
Library repository	Opens the Library repository dialog. There, you can install more libraries on your local system.

Also refer to

- [chapter “Library repository” command on page 636](#)

"Placeholder" tab

Placeholder name	The input field provides a combo box for entering the valid placeholder names that are read from the currently accessible device descriptions. You can also enter a new placeholder name in order to define a "free placeholder", which is not resolved by the device or by the library profile.
Standard library	IndraLogic uses this library when for any reason no device is available that the resolution defines. In this way it is possible to compile the current project without errors.

**Note about placeholder resolution**

If the selected library is developed in compliance with the "Guidelines for developing libraries", then we do not recommend that you change the following settings.

Also refer to

- [chapter "'Placeholder" command" on page 675](#)

"Try to reload library" command

Function: The command tries to reload the selected library.

Call: Menu **Library manager** ► **Try to reload library**.

Prerequisite: A library is selected that failed to load.

If for any reason a library is not available in the defined repository location when a project is opened, IndraLogic displays a corresponding error message.

Once you have rectified the error, i.e. when the library is properly available again, you can reload the library with this command without having to leave the project.

Also refer to

- [chapter "'Placeholder" command" on page 675](#)

"Properties" command

Function: The command opens the **Properties** dialog for the library selected in the library manager.

Call: Menu **Library manager** ► **Properties** or **Properties** in the header of the editor

Prerequisite: A library is selected.



This dialog is intended for library developers. Use it only if you have profound knowledge of library referencing. In addition, follow the guidelines for library developers.

"Properties" dialog

General information

Namespace	Namespace of the selected library. By default, this is identical to the library name, unless it was defined explicitly in the library info when the library was created. You can change the namespace for the open project. Example: LA
Default	Library that triggers the placeholder when no other trigger is defined or is possible. Prerequisite: The selected library is a library placeholder, and therefore the setting is available. Note: For the compiler version 3.5.8.0 and later, the following applies to library placeholders with a resolution in the device description located in the library manager of the "General module folder" view: > This placeholder is always resolved automatically according to the description of the device that compiles the application.



Note about placeholder resolution

If the selected library is developed in compliance with the "Guidelines for developing libraries", then we do not recommend that you change the following settings.

Version

Selection of version constraint Prerequisite: The settings are available only if the selected library is not a library placeholder. Note: Container and interface libraries are created automatically with library references with version constraint. As far as possible, do not create libraries that include library references with version constraint. Otherwise, you reference the libraries by placeholders. Edit a placeholder resolution in the "Placeholders" dialog.	
Exact version	 (selected from list box) Version is integrated into the project. Note: This option is strongly recommended for container libraries, and it is usually preset for this library type.
Always latest version	 The library repository is scanned and the latest detected version is integrated. Note: If a newer library version is available, the actually used library function blocks can change. This option is strongly recommended for interface libraries, and it is usually preset for this library type.

Visibility

Only allow qualified access to all identifiers	 Library function blocks (and variables) are called in the project only with prepended namespace paths.
If the current project is referenced as a library by another project	Note: Changing the following settings makes sense only if you created a library with your project and therefore opened a library project. In this way, the selected library is referenced in the new library.

Make all IEC symbols in the project visible, as if this reference had been directly integrated here	: As a container library, the selected library makes the contents of the referenced library visible at the top level (later in a project). Prerequisite: A container project is created with a library project. A container library does not implement its own function blocks, but references other libraries exclusively. It bundles libraries. A container library can be employed sensibly to bundle multiple libraries (in a reference) in a project. This option must be activated for each library reference. Symbolic access to library function blocks: <code><NamespaceContainerLibrary>.<BlockName></code> : The contents of the referenced library is accessed uniquely by means of the namespace. The path name consists of the library name and the unique name (library reference), and it is prepended to the function block name. Prerequisite: No container project is created with a library project. Unique symbolic access to a library function block: <code><NamespaceLibrary>.<NamespaceSublibrary>.<BlockName></code>
Hide this reference in the dependency tree	: The selected library is not displayed in the library manager as a library reference (later in a project). The library is a hidden reference. Warning: If there are compile errors resulting from hidden library errors, then detecting the errors may be difficult. : The selected library is displayed as a library reference (later in a project).
Optional (no error is displayed if the library is missing):	: The selected library is treated as optional. When downloading the project that references the library, no error is reported, even if the library is not available in the library repository.

Also refer to

- [chapter "Placeholder" command" on page 675](#)

"Placeholder" command

Function: The command opens the **Placeholder** dialog. The dialog notifies about the selected placeholder library and allows for the assignment of a valid resolution especially for the project.

Call: Menu in the header of the editor **Placeholder**.

Prerequisite: The library manager is open in the editor.

A placeholder library implemented below the device node „Application“ is resolved as follows:

- If you have assigned a special resolution to the placeholder library in the "Placeholders" dialog, it is applied.
- If no special resolution is given, then a check is performed as to whether a resolution is specified in the device description of the application.
- This is followed by a check as to whether a resolution is specified in the library profile.
- The result is shown in the library manager below the "Effective version".

A placeholder library implemented in the "General module folder" is resolved as follows:

- Any special resolution assigned in the "Placeholders" dialog is **ignored**.

- An application-dependent check is performed as to whether a resolution is specified in the associated device description.
- Then the library profile is checked.
- The result is shown in the tooltip of the symbol 

Also refer to

- [chapter 16.1 "Information for library developers" on page 219](#)
- [chapter 18.1.14 "Object 'Library manager'" on page 516](#)

"Placeholders" dialog

Name	Identifier of the placeholder
Library	Current resolution for the project Double-click the entry to edit the placeholder resolution. A selection list of the available versions of the library appears. In addition, the "Other libraries" command is displayed
Command "Other library"	The "Search libraries" dialog opens for searching for and installing libraries. Select this command to redirect to a special library, and not to switch to another version.
Info	Type of placeholder resolution • Resolved by device description • Resolved by library profile • Resolved by <SpecialLibrary>

"Library info" command

Symbol: 

Function: The command opens the **Library info** dialog.

Call: **General module** folder

Prerequisite: IndraLogic automatically creates the object in the library projects. In other projects, the dialog can be created via the context menu item **Add ► Library info**.

Also refer to

- [chapter 2.5.1 "Retrieving and editing library information" on page 11](#)
- [chapter 2.5.2 "Applying project settings" on page 13](#)
- [chapter 18.1.20 "'Library info' object" on page 545](#)

18.2.11 Menu 'Image pool'

"Insert image"

Symbol: 

Function: The command inserts a new line into an image pool.

Call: Menu **Image pool** or context menu

Prerequisites: An image pool is active and a line is selected in the image pool.

Also refer to

- [chapter 18.1.13 "'Image pool' object" on page 515](#)

18.2.12 “Declarations” menu

“Edit declaration header” command

Function: The command opens the **Edit declaration header** dialog used to configure the header of a POU in the tabular declaration editor.

Call: Menu item **<pou-name>** or context menu of the tabular declaration editor.

Prerequisite: The focus is set on the tabular declaration editor.

Also refer to

- [“Declaring in the tabular declaration editor” on page 85](#)

“Edit declaration header” dialog

Function: Configure the header of the tabular declaration section of a POU in the header.

Call: click on the header of the tabular declaration editor, context menu in the tabular declaration editor.

Declaration	Selection list to change the POU type • PROGRAM • FUNCTION_BLOCK – EXTENDS: Input field for a basic function block – IMPLEMENTS: Input field for an interface • FUNCTION – Return type Input field with current POU name: The POU name can be changed.
Attributes	The Attributes dialog opens to enter attributes and pragmas.

Also refer to

- [chapter 9.2.1 “Using the declaration editor” on page 85](#)
- [chapter 17.6 “Pragmas” on page 378](#)

“Shift downwards” command

Symbol: 

Function: The command shifts a variable declaration downwards by one row.

Call: Menu item **<pou-name>** or context menu of the tabular declaration editor.

Prerequisite: A row with a variable declaration is selected in the tabular declaration editor.

Also refer to

- [chapter 9.2.1 “Using the declaration editor” on page 85](#)

“Shift upwards” command

Symbol: 

Function: The command shifts a variable declaration upwards by one row.

Call: Menu item **<pou-name>** or context menu of the tabular declaration editor.

Prerequisite: A row with a variable declaration is selected in the tabular declaration editor.

Also refer to

- [chapter 9.2.1 "Using the declaration editor" on page 85](#)

18.2.13 Menu 'Declarations' (persistence)

"Rearrange list and clean gaps"

Function: The command cleans the gaps that can result when making changes to the declaration of persistent variables. The memory requirement is reduced by this cleaning.

When executing the command, IndraLogic displays a warning regarding possible loss of data.

Call: **PersistentVars** or context menu

Prerequisite: the persistence editor (persistent variable list) is active.



Before cleaning you should consider saving the current values of the persistent variables to a recipe (command **Save current values to recipe**). Then you can load the values to the controller again after the next download.

Troubleshooting

- F: Upon (initial) saving of current values in a recipe, saving is denied and an error message is output.
- A: Check whether a "Library info" object ("General module folder") is used as "project information" in your project (library information). Disable the option "Released" under "Summary" and save them again.

Also refer to

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter ""Save current values to recipe" command" on page 678](#)
- [chapter ""Restore values from recipe" command" on page 679](#)

"Save current values to recipe" command

Function: The command creates a new recipe definition in the recipe manager and stores the current values of the persistent variables in it. You should execute this command before the command **Rearrange list and clean gaps** in order to avoid a possible loss of data.

You can subsequently restore the data with the context menu item **Restore values from recipe**.

Call: **PersistentVars** or context menu

Prerequisite: the application is in online mode and the persistence editor (persistent variable list) is active.



If there is already a list with respective names in the recipe manager when saving a persistent variable list, the current persistent variables are sorted into that list:

- New persistent variables are added to the list
- Variables that are not included in the list of persistent variables are deleted

It is thus possible to add more recipes to be retained to the list in the recipe manager. However, if new variables are added to the list, they are deleted upon the next execution of the command "Save current values in recipe".

Also refer to

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter ""Rearrange list and clean gapson page 678](#)
- [chapter ""Restore values from recipe" command" on page 679](#)
- [chapter 18.1.12 "'Persistent variable list' object" on page 514](#)

"Restore values from recipe" command

Function: The command restores the values of the persistent variables that you have stored in a recipe using the command **Save current values to recipe**.

You would normally select this command after executing the context menu command **Rearrange list and clean gaps**.

Call: **PersistentVars** or context menu

Prerequisite: the persistence editor (persistent variable list) is active, the application is in online mode

Also refer to

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter ""Save current values to recipe" command" on page 678](#)
- [chapter ""Rearrange list and clean gapson page 678](#)
- [chapter 18.1.12 "'Persistent variable list' object" on page 514](#)

Command 'Add all instance paths'

Function:

- When you execute the command in the persistence editor, the application is searched for declarations of persistent variables with the `PERSISTENT` keyword which are outside of the persistence editor. For each declaration found, an instance path of this variable is added in the persistence editor.
- When you execute the command in a variable configuration, an instance path is added for each variable with an incomplete address. All function blocks of the application are considered in this case.

Call: Menu **PersistentVars**, or via the via context menu of the persistence editor

Prerequisites:

- the persistence editor (global persistent variable list) is active or a variable configuration (global variable list with `VAR_CONFIG` declarations) is opened.
- The application was compiled successfully.

Also refer to

- [chapter 9.16 "Data persistence" on page 137](#)
- [chapter 17.2.11 "Persistent variables - PERSISTENT" on page 288](#)
- [chapter 9.10.1 "Variable configuration - VAR_CONFIG" on page 123](#)

18.2.14 Menu 'Recipes'

"Insert variable" command

Symbol: 

Function: The command inserts a variable into the currently opened recipe definition before the selected position.

Call: **Recipes** menu

Prerequisite: You have opened a recipe definition in the editor and selected the normal view

IndraLogic inserts the default text "NewVariable" in the column **Variable**. Replace this name with the respective variable name. To do this, open the input assistant by clicking  or enter the variable name directly into the table element.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Add a new recipe" command

Symbol: 

Function: The command opens a dialog for adding a new recipe (new column) to the recipe definition.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: A recipe definition is open in the editor.

After choosing the command, a dialog opens for you to define the name of the new recipe. The dialog also provides the capability of copying existing recipes into the new recipe.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Remove recipe" command

Symbol: 

Function: The command removes a recipe from the currently opened recipe definition.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: You have selected a field in the recipe column of a recipe definition.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Load Recipe" command

Symbol: 

Function: The command loads a recipe from a file.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: You have selected a field in the recipe column of a recipe definition.

The command overwrites the values of the selected recipe of the recipe definition.



If the option **Recipe management in the PLC (Recipe manager ► General)** is selected, note the following:

If you change recipes in the project by choosing the command **Load Recipe** or **Read Recipe**, then an online change is required when logging in again.



If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definitions are not read, and therefore updating leaves these variables unchanged on the control and in the project.

For values of the data type `REAL/LREAL`, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Save recipe" command

Symbol: 

Function: The command saves the variable values of a recipe to a file.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: You have selected the value of a recipe in the recipe definition.

When executing the command, IndraLogic saves the values of the selected recipe to a file. You can define the format in the settings for the recipe manager in the tab **Storage**.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Read recipe" command

Symbol: 

Function: The command reads the variable values of a recipe from the control.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: The application is in online mode and the value of a recipe is selected in the recipe definition.

When executing the command, IndraLogic overwrites the values of the selected recipe with the values read from the control.



If the option **Recipe management in the PLC (Recipe manager ► General)** is selected, note the following:

If you change recipes in the project by choosing the command **Load Recipe** or **Read Recipe**, then an online change is required when logging in again.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2 "Menu commands" on page 572](#)

"Write recipe" command

Symbol:



Function: The command writes the values of a recipe to the variables in the control.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: The application is in online mode and the value of a recipe is selected in the recipe definition.

When executing the command, IndraLogic overwrites the values in the control with the values of the selected recipe.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Load and write recipe" command

Symbol:



Function: The command loads a recipe from a file and writes the values to the variables in the control.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: The application is in online mode. You have selected the value of a recipe in the recipe definition.

After choosing the command, you are prompted either to write the variable values also to the recipe in the project or only to write them to the PLC. Updating the values in the recipe could require an online change when logging in again.

When you choose this command, IndraLogic overwrites the values of the selected recipe of the recipe definition. In addition, these recipe values overwrite the variable values in the control.



If you want to overwrite only individual recipe variables with new values, then remove the values for the other variables before loading to the recipe file. Entries without value definitions are not read, and therefore updating leaves these variables unchanged on the control and in the project.

For values of the data type `REAL/LREAL`, the hexadecimal value is also written to the recipe file in some cases. This is necessary so that the exact identical value is restored when converting back. In this case, change the decimal value and delete the hexadecimal value.

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Read and save recipe" command

Symbol:

Function: The command reads the variable values of a recipe from the control and saves them to a file.

Call: Menu **VI-Logic recipe definition** or context menu

Prerequisite: The application is in online mode and the value of a recipe is selected in the recipe definition.

After choosing the command, you are prompted either to read the variable values to the recipe or only to save them. Updating the values in the recipe could require an online change when logging in again.

The values are saved with the default name for recipe files according to the settings for the recipe manager (tab **Storage**).

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Remove variable" command

Symbol:

Function: This command removes the selected variables from a "Recipe Definition".

Call: Menu **Recipes** after applying from **Tools ► Adjust**, command category "Recipe"

Prerequisite: You have opened a recipe definition in the editor and selected the normal view

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

"Update structured variables" command

Symbol:

Function: The command opens the **Update structured variables** dialog.

Call: Menu **VI-Logic recipe definition** or context menu

In this dialog, you can update recipe definitions if the declaration of a structured variable or a block has changed. For example, if the dimension of an array is changed, then you can automatically add or remove the entries in the recipe definition.

Remove non-existing variables	 Variables are removed from the recipe definition when they no longer exist in the project due to a change to a structured element.
Update instances of structures and function blocks	 If the declaration of a structure or function block is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
Update array dimensions of array instances	 If the dimension of an array is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
Update included Global Variable Lists	 If the declaration of a global variable list is extended and available in the recipe definition with an instance, then the respective variables are added to the recipe definition.
Update contained programs	 If the declaration of a program is extended and instanced in the recipe definition, then the respective variables are added to the recipe definition.

Tab. 18-15: "Update structured variables" dialog

Also refer to

- [chapter 13.2 "Changing values with recipes" on page 198](#)
- [chapter 18.2.14 "Menu 'Recipes'" on page 680](#)

18.2.15 Menu 'Text list'

"Add language" command

Symbol: 

Function: The command adds another language column to the text list.

Call: Menu **VI-Logic text list** or context menu

Prerequisite: A text list or a global text list is open and active.

In the **Choose language** dialog, enter the country code for the new language, e.g. **en-US**. IndraLogic inserts the code as column header.

"Create global text list" command

Symbol: 

Call: Menu **VI-Logic visualization**, context menu

Prerequisites: A visualization is open.

Function: The "GlobalTextList" object is automatically created in the "General module folder" as soon as the first text is defined when configuring a visualization.

The menu item "Create global text list" is also available in the main menu. Go to **VI-Logic visualization ► Create global text list**.

Alternatively, the menu item is available in the context menu if the cursor is in the working area of the visualization editor.

Also refer to

- [chapter 18.1.10 ""GlobalTextList" object" on page 512](#)

"Export all as text" command

Symbol: 

Function: The command exports all text lists of the project.

Call: Menu **Text list**, context menu

Prerequisite

- A text list or a global text list is open and active.
- The visualization does not code the characters of the text in Unicode.

For each text list, IndraLogic creates a file as simple text in `.txt`. The text list name becomes the file name.

C:\ProgramData\IW-Projects\0\Project\IndraLogic\globaltextlist.txt or C:\ProgramData\IW-Projects\0\Project\IndraLogic\application.textlist.txt

A control can read and use this format. The file can for example be copied to the control and configured via a setting in the visualization manager to ensure that the text lists are not transferred again while loading the application.

"Export all as Unicode text" command

Symbol: 

Function: The command exports all text lists of the project.

Call: Menu **Text list**, context menu

Prerequisite:

- A text list or a global text list is open and active.
- The visualization codes the characters of the text in Unicode.
 - The **Unicode character string** option is selected in the visualization manager.
 - The compiler instruction `VISU_USEWSTRING` of the application is set. To check that, select the **Properties** command in the context menu of the application. Click on the **Build** tab. In the input field on the **Compiler definitions**, `VISU_USEWSTRING` is entered.

For each text list, IndraLogic creates a file as simple text in `.txt`.

The text list name becomes the file name. The directory to which files are exported, is set in **Project ► Project settings ► Visualization**, category **General** to **Text list files**

C:\ProgramData\IW-Projects\0\Project\IndraLogic\globaltextlist.txt or C:\ProgramData\IW-Projects\0\Project\IndraLogic\application.textlist.txt

A control can read and use this format. The file can for example be copied to the control and configured via a setting in the visualization manager to ensure that the text lists are not transferred again while loading the application.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

"Insert text" command

Symbol: 

Function: The command inserts a new line above the selected line in the text list. Enter the source text to open an input field under **Standard**.

Call: Menu **Text list** ► **Insert object**, context menu

Prerequisite: A text list, not a **GlobalTextList**, is open and active. A field is selected in the table.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

"Import/export text lists" command

Symbol: 

Function: The command exports an active text list, imports a file or compares a text list to a file. The file is in CSV format. The **Import/export** provides options.

Call: Menu **VI-Logic text list** ► **Import/export text lists**, context menu

Prerequisite: A text list or a global text list is open and active.

"Import/export" dialog

Select file for comparison or import	File read out by IndraLogic.  Opens the dialog Select text file list to select a file in this dialog.
Choose export file	File to which IndraLogic writes.  Opens the dialog Select text list file and select a file and a directory in this dialog.

Import/export type

Import	Prerequisite: Select a file in Choose file to compare or to import. The file can contains text list entries for the Global Text List as well as for text lists. <i>Global text list</i> • IndraLogic reads the file, compares the text list entries for the same source text and applies the differences in the translations. IndraLogic can overwrite translations in the project. <i>Text lists</i> • IndraLogic reads the file, compares the text list entries for the same IDs and applies the differences in the source text and the translations to the project. IndraLogic can overwrite text list entries in the project. • If the file contains a new ID, the text list entry in the text list of the project is imported and the text list is supplemented.
Import replacement file	Prerequisite: Select a replacement file in Choose file to compare or to import The replacement file contains replacements for the global text list. IndraLogic processes the replacement file line by line and performs the specified replacements in the global text list.

Export	Prerequisite: The file to which IndraLogic writes, is selected in Select export file. IndraLogic exports all texts from the text lists of the current project. All languages available in the project are added to the export file as columns. The file can be used to externally translate the language-dependent texts.
Export only text differences	Prerequisite: Select an import file in Choose file to compare or to import for the comparison. IndraLogic writes to an export file located in Select export file. IndraLogic reads the import file and compares it the lines of the active text list. If the lines match, IndraLogic ignores them. If lines differ from each other, IndraLogic writes the line to the export file and can apply translations from the text list. IndraLogic applies the translations from the import file and overwrites them.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

“Remove language” command

Symbol: 

Function: Removes the selected language column from the text list.

Call: Menu **VI-Logic text list** or context menu

Prerequisite: A text list or a global text list is open and active. A field in the column of the language to be removed, is selected.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

“Remove unused text list entries” command

Symbol: 

Function: The command checks whether a text list entry is used as static text in the project. If not, IndraLogic removes it from the text list.

Call: **VI-Logic text list** or context menu

Prerequisite: **GlobalTextList** is open and active. A field is selected in the table.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

“Check visualization text IDs” command

Symbol: 

Function: The command checks whether the ID of the text list entry is correct in the project and reports the result.

Call: **VI_Logic text list** or context menu

Prerequisite: **GlobalTextList** is open and active. A field is selected in the table.

If detects during the check that the global text lists and the static text lists of the visualizations do not match, a write-protection of the global text list can be the reason.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

“Update visualization text IDs” command

Symbol: 

Function: The command updates all inconsistent IDs in a static text list.

Call: Text list ► Insert text Menu, context menu

Prerequisite: GlobalTextList is open and active. A field is selected in the table. The object is read-only.

If detects during the check that the global text lists and the static text lists of the visualizations do not match, a write-protection of the global text list can be the reason.

Also refer to

- [chapter 9.8 "Managing text in a text list" on page 113](#)

“Add text list support” command

Symbol: 

Function: The command adds a text list support to the selected DUT object of the enumeration type.

Call: Context menu of standard DUT object of the type “Enumeration” (symbol:

)

The text list support allows the localization of enumeration component identifiers and a representation of the symbolic component value in a text output in the visualization.

Also refer to

- [chapter 18.1.8 "Object "DUT" on page 488](#)
- [chapter ""Remove text list support" command" on page 688](#)

“Remove text list support” command

Symbol: 

Function: The command removes the text list support from the selected enumeration object.

Call: Context menu of an object of a text list-supported enumeration (symbol:

)

The text list support allows the localization of enumeration component identifiers and a representation of the symbolic component value in a text output in the visualization.

Also refer to

- [chapter 18.1.8 "Object "DUT" on page 488](#)
- [chapter ""Add text list support" command" on page 688](#)

18.2.16 Menu 'Trace'

"Add variable" command

Function: The command adds a trace variable to the configuration.

Call: Trace menu

A new variable is displayed in the tree view of the trace configuration. The respective variable configuration is displayed on the right under **Variable settings**.

Also refer to

- ["Subdialog 'Variable settings'" on page 740](#)
- [chapter 13.3.2 "Creating the trace configuration" on page 204](#)

'AutoFit' command

Symbol: 

Function: This command scales the y-axis of the trace diagram for optimum display of all graphs, making sure that the y-values fit in the visible region of the diagrams. The command works with both single-channel and multi-channel displays.

Call: **Trace** menu, context menu

Trace in a single-channel display

When all trace variables are displayed in one diagram, the trace is in single-channel display.

Trace in a multi-channel display

When the trace variables are displayed in multiple diagrams, the trace is in multi-channel display.

Also refer to

- [chapter "'Trace configuration' dialog" on page 737](#)

Also refer to

- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

"Compress" command

Symbol: 

Function: The command compresses the trace graph by zooming into the displayed time range by a fixed percentage.

Call: **Trace** menu, context menu

Also refer to

- [chapter "'Stretch" command " on page 696](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

"Configuration" command

Function: The command opens the **Trace Configuration** dialog for enabling the configuration of the data recording.

Call: **Trace** menu, context menu

Also refer to

- [chapter "'Trace configuration' dialog" on page 737](#)
- ["Subdialog 'Variable settings'" on page 740](#)

"Cursor" command

Symbol: 

Function: *The command*

- adds a trace cursor to the trace diagram when **no** trace cursor is available

- adds a trace cursor to the trace diagram when **one** trace cursor is available
- removes the trace cursor when **two** trace cursors are available

Call: **Trace** menu, context menu

A trace cursor is a small black triangle with a vertical black line running parallel to the y-axis.

Trace diagram without trace cursor

In this mode, you can run through the trace diagram with the mouse pointer. The x-value that focuses on with the cursor is displayed in the status bar with normal style.

Example: **Time: 1m23s456ms - Value: 1**

Trace diagram with one trace cursor

In the status bar and y-value, IndraLogic prints the time that was marked by the trace cursor.

Example: **Time: 1m23s456ms**

Trace diagram with two trace cursors

In the status bar, IndraLogic prints the two times and the time interval that are marked by the two trace cursors.

Example: **Time: 1m23s456ms - Time: 1m24s456ms (Δ 1s)** .

User input in the trace diagram

If one or two trace cursors are available, then you can move them along the x-axis.

User-input via mouse	Symbol	Effect
Drag the triangle of a trace cursor to another position		While the mouse button is pressed, the cursor can be moved without restriction. The current Y-value is always displayed in the status bar. When the mouse button is released, the cursor jumps to the nearest measuring point

Keyboard shortcuts	Effect
<Left arrow> <Right arrow>	IndraLogic moves the black trace cursor to the next measuring point.
<Shift>+<Left arrow> <Shift>+<Right arrow>	IndraLogic moves the gray trace cursor moves the gray trace cursor to the next measuring point.

Also refer to

- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

'Download trace' command

Symbol: 

Function: This command transfers the trace configuration on the controller to the associated application and starts the data recording. The recorded data is retransferred to the development system. The trace diagram shows the current samples and continues to run.

Call: **Trace** menu, context menu

Prerequisite: The command is available when the assigned application is in online mode.

Also refer to

- [chapter 13.3.3 "Controlling data recording" on page 207](#)

“Export symbolic trace config” command

Function: The command exports a trace configuration to a *.traceconfig file.

Call: Trace menu, context menu

Prerequisite: The origin application includes a symbol configuration that defines the configured trace variables as symbols. Access to the IEC variables where data was recorded is therefore symbolic. Then you can use the trace configuration for various similar applications.

Using the configuration file

You can transfer this file to any runtime system. At runtime, its CmpTraceMgr runtime system component can access and perform data recording. The configuration file also includes information about the application context in addition to the configuration data.

The configuration file defines the following context

- Name of the application
- Trace name
- Name of the task

The application that is executed at runtime must fulfill the following conditions

- The application has the same name as the origin application
- The trace that is configured in the application has the same as the trace that is configured in the origin application
- The task that is running in the data recording has the same name as the task that is configured in the origin application



The configuration is not loaded automatically. You must execute the command explicitly.

Proceed as follows

- Access the trace manager programmatically via IEC code by using library interfaces
- Register the configuration file with the trace manager. Then the trace manager loads the configuration file when the application is started.

Example of a configuration file

Configuration file Trace_getting_started.traceconfig

```
[key]; [value]
Version; 0x03050000
Name; Application.Trace
ApplicationName; Application
ApplicationDataGuid; 00000000-0000-0000-0000-000000000000
IecTaskName; PlcTask
Comment;
Trigger.Flags; 0
Condition.Name;
Condition.AddrFlags; 0x00000000
Condition.Class; 0
Condition.Size; 0
EveryNCycles; 1
BufferEntries; 81
Flags; 16
0.Variable; PlcProg.rSin
0.Address.AddrFlags; 0x00000101
0.Class; 14
0.Size; 4
0.GraphColor; 4278190335
0.GraphType; 1
0.MinWarningColor; 4278190080
0.MaxWarningColor; 4294901760
0.CriticalLowerLimit; 0
```

```
0.CriticalUpperLimit; 0
0.ActivateMinWarning; 0
0.ActivateMaxWarning; 0
0.YAxis; 0
0.Data;
```

Also refer to

- chapter 13.3.6 "Managing trace" on page 207

“Load trace” command

Function: The command makes it possible to load a file, which contains the configuration and data, and was saved to the file system of the development system. The **Load trace** dialog opens.

Call: **Trace** menu, context menu

“Load trace” dialog

File name	Name of the file loaded
File type	File format *.trace: Trace file including the trace configuration and samples *.csv: Text file in CSV format that includes a trace configuration

Also refer to

- chapter ""Save trace" command" on page 695
- chapter 13.3.6 "Managing trace" on page 207

“Zoom with mouse” command

Symbol:  (Command disabled),  (Command enabled).

Function: The command enables and disables mouse zooming in the trace diagram.

Call: **Trace** menu, context menu

User input in the trace diagram If the command is enabled, stretch a rectangle with the mouse. When releasing the mouse button, the display zooms in on the box and the data is enlarged.

Also refer to

- chapter "'Trace configuration' dialog" on page 737
- chapter 13.3.5 "Navigating in the trace data" on page 207

Command 'Convert to multi-channel'

Function: The command toggles the display in the trace diagram between single-channel and multi-channel.

Call: **Trace** menu, context menu

Symbol: ☐ for single-channel display, ☒ for multi-channel display

Prerequisite: At least two trace variables have to be defined in the trace configuration.

Multi-channel display Multi-channel display means that the trace variables are displayed in multiple diagrams.

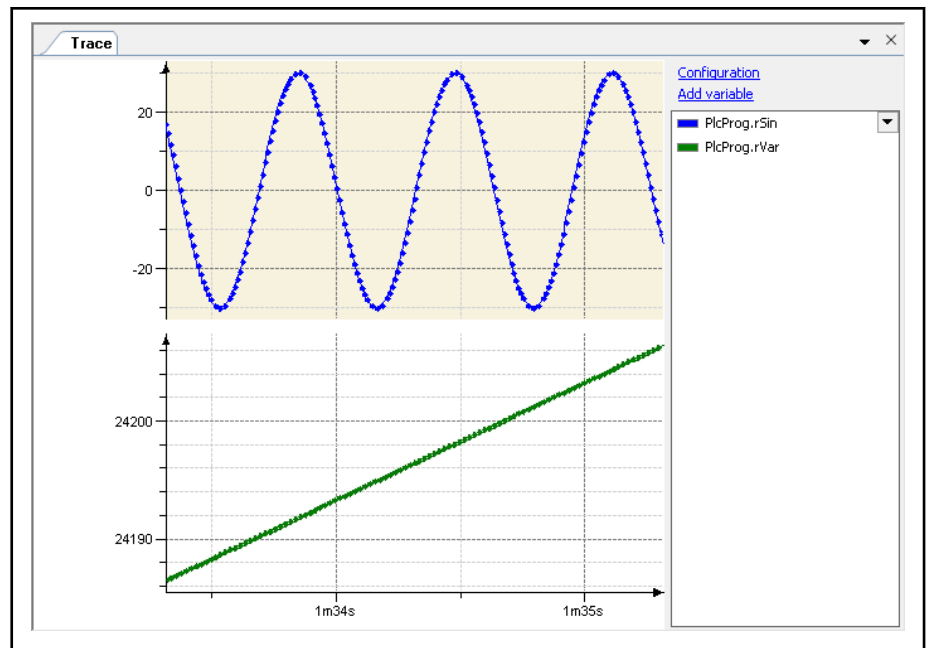


Fig. 18-71:

Also refer to:

- [chapter "Trace configuration' dialog" on page 737](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

Command 'Convert to single-channel'

Function: This command switches the display in the trace editor from multi-channel to single-channel.

Call: Trace menu, context menu

Symbol: ☐ (single-channel display), ☒ (multi-channel display).

Single-channel display

If a trace is displayed as single-channel, all trace variables are included in one diagram.

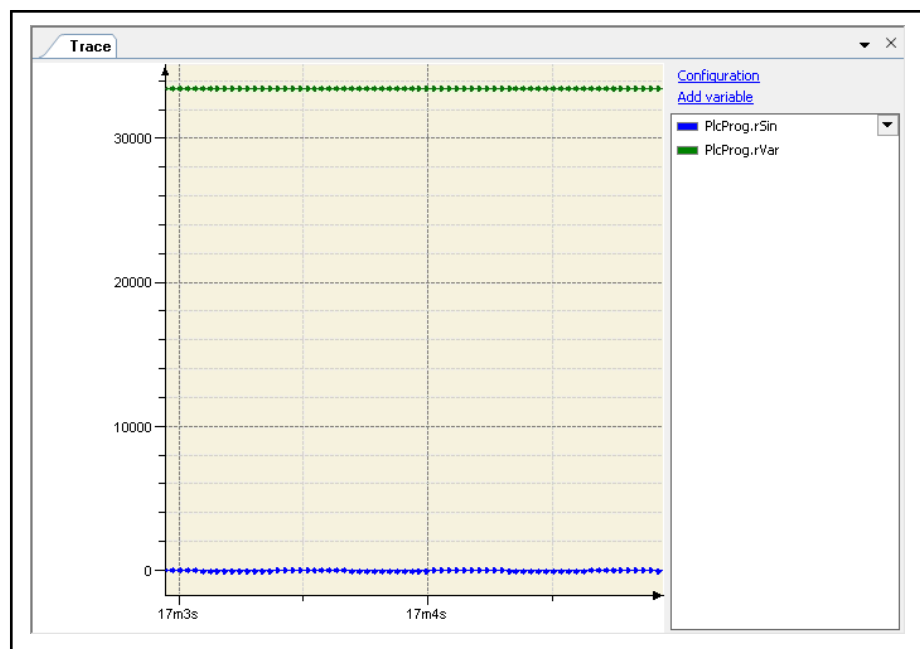


Fig. 18-72:

See:

- [chapter "'Trace configuration' dialog" on page 737](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

Command 'Online list'

Function: The command opens the "Online list" dialog. If the trace editor of a DeviceTrace object is active, then all traces that are running on the controller are displayed in a tree view. If the trace editor of an application-specific trace object is active, then only this trace is displayed if it is running on the controller.

Call: Menu **Trace**, optionally, context menu of the trace editor

Prerequisites: The runtime system uses the `CmpTraceMgr` component. An application belonging to the device is in online mode.



Closing the DeviceTrace editor terminates the connection to the control!

Note that the connection to the controller is also terminated when the last open DeviceTrace editor is closed.

To represent the device traces in the project again, reload the device traces in the DeviceTrace objects.

Currently, closing the editor is also the recommended procedure for deliberately terminating the connection to the controller. Logging out is not sufficient to close the connection to the control.

"Online list" dialog

Delete from runtime system	Stops and removes the selected trace from the running application.
Upload	The command is visible when a DeviceTrace is loaded in the trace editor. A DeviceTrace is a trace that runs on the controller. In the device tree, it can be represented with a DeviceTrace object directly below a device. When you execute this command, the trace that is selected in the tree view is loaded from the runtime system into the trace editor. Any existing configuration in the project is overwritten. For example, the device can provide traces for data of the processor load (cpuload, plcload), which you can track in the trace editor in IndraLogic. An individual DeviceTrace object is necessary in the device tree for each trace of the device that should be displayed in the project.

Also refer to

- [chapter 13.3.4 "Accessing all traces of the control" on page 207](#)
- [chapter 18.1.27 ""Trace" object" on page 568](#)
- [chapter " Command 'Upload trace'" on page 697](#)

"Reset trigger" command

Symbol: 

Function: The command resets the trace configuration after a triggered data recording. Then the application can record new data and react to a trigger again.

Call: **Trace** menu, context menu

Prerequisite: After triggering, the complete data is in the buffer of the development system.

Also refer to

- [chapter 13.3.3 "Controlling data recording" on page 207](#)

"Reset to default view" command

Symbol: 

Function: The command resets the trace diagram to the default view.

Call: **Trace** menu, context menu

Prerequisite: The display in the trace diagram has been changed by zooming, scrolling, or **AutoFit**.

Also refer to

- [chapter "'AutoFit' command" on page 689](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

"Save trace" command

Function: The command saves the data to a file on the development system. Depending on the file format, the configuration may also be saved. The **Save trace** dialog opens.

Call: **Trace** menu, context menu

"Save trace" dialog

File name	Name and location of the trace file
File type	<i>File type</i> • *.trace: Trace file contains the data and configuration. You can run the Load Trace command to load the file to the trace editor when offline. • *.txt: Text file with recorded data. To load and edit a file, use tools supporting .csv format. It cannot be loaded to the trace editor when offline because the trace editor cannot read this format. • *.trace.csv Trace CSV file (data only) contains the recorded data. Address information is provided for each trace variable. The created file can be read in the runtime system. The data is imported but the trace cannot be started because the variable addresses are not saved.

Also refer to

- [chapter ""Load trace" command " on page 692](#)
- [chapter 13.3.6 "Managing trace" on page 207](#)

"Start trace" command

Symbol: 

Function: The command starts the data recording on the control when it is stopped.

Call: **Trace** menu, context menu

Prerequisite: The assigned application on the runtime system is running and a trace configuration is loaded.

Also refer to

- [chapter 13.3.3 "Controlling data recording" on page 207](#)

"Stop trace" command

Symbol: 

Function: The command stops the data recording of a trace.

Call: **Trace** menu, context menu

Prerequisite: The assigned application on the runtime system is running and executes a trace.

Also refer to

- [chapter 13.3.3 "Controlling data recording" on page 207](#)

"Stretch" command

Symbol: 

Function: The command stretches the trace graph by zooming out of the displayed time range by a fixed percentage.

Call: **Trace** menu, context menu

Also refer to

- [chapter ""Compress" command" on page 689](#)
- [chapter 13.3.5 "Navigating in the trace data" on page 207](#)

Command 'Upload trace'

Function: The command establishes the connection to the PLC device, if not already connected. Then it opens the Online List dialog listing the traces running on the controller. Then the selected trace is loaded to the trace editor by means of the Upload command in the dialog.

Call: Menu **Trace**, optionally, context menu of the trace editor

Prerequisite: The editor of a DeviceTrace object is opened. The runtime system uses the `CmpTraceMgr` components (Trace Manager). At least one application in the runtime system is running. The communication settings for the PLC are configured correctly in the IndraLogic project.



Closing the DeviceTrace editor terminates the connection to the control!

Note that the connection to the controller is also terminated when the last open DeviceTrace editor is closed.

To represent the device traces in the project again, reload the device traces in the DeviceTrace objects.

Currently, closing the editor is also the recommended procedure for deliberately terminating the connection to the controller. Logging out is not sufficient to close the connection to the control.

Also refer to

- [""Online list" dialog" on page 694](#)
- [chapter 13.3.4 "Accessing all traces of the control" on page 207](#)

"Statistics" command

Function: The command opens the **Trace Statistics** dialog, which shows statistics about each trace variable.

Call: Menu **Trace**, context menu or for post-installation: **Tools** ► **Customize** ► **Commands** ► **Trace**

Prerequisite: The trace editor contains samples.

"Trace statistics" dialog

The analyzed time range and duration are shown in the caption.

The table contains one line per signal.

Signal	Name of the trace variable, e.g. <code>PlcProg.S1</code>
Index	0-based index of the signal order
N	Number of measurements for the calculations
Min	Minimum value
Avg	Average
Median	Middle value when the values are sorted by size.
RMS	Root mean square
StdDev	Standard deviation
Max	Maximum value

Integral	Integral
Min Δt [s]	Minimum value of time intervals for successive values
Avg Δt [s]	Average change of time intervals for successive values
Median Δt [s]	Median of time intervals for successive values
StdDev Δt [s]	Standard deviation of change of time intervals for successive values
Max Δt [s]	Maximum value of time intervals for successive values

Click on the table header in a certain column	IndraLogic sorts the table by that column, changing the order from ascending to descending and back. Default setting: The table is sorted ascending by the Index column. The signals are then sorted in the same order as in the signal tree.
Click into the line	The line is selected. You can select or clear other lines by pressing <Shift>+<arrow> up or down.
<Ctrl>+<C>	IndraLogic copies the selected lines as text to the clipboard. The values of the individual columns are tab-separated, and the lines are delimited with the control character <CR> or <LF>. Prerequisite: At least one line is selected.

Also refer to

- [chapter 13.3 "Data recording with "Trace"" on page 200](#)

18.2.17 Miscellaneous

"Simulation" command

Function: The command used to switch on and off the simulation mode of a control.

Call: Context menu of the respective control

Prerequisite: Control is logged off.

In the simulation mode, the active application can be started and debugged on a "simulated target device".

No real target device is required to test the online response of an application.

Upon initial login, It is prompted whether the application is to be created and loaded.

No communication settings have to be made for a simulated device.

Der simulation mode becomes visible in the status line by using the "Simulation" entry.



Use the PlcTask (cyclic tasks) and programs assigned to this task.

After successful login, the IndraWorks status line in the Project Explorer signals the simulation mode. Use the corresponding online commands to test the application.

To switch off the simulation mode, first log off from the control and then call the **Simulation** command again.

The command always affects the active application.



No drivers are currently provided for the periphery of the control.

"Add to watch list" command

Symbol: 

Function: The command adds the variable of the current location of the cursor to a watch list for the purpose of online monitoring.

Call: Right-click a variable in an editor when the application is in online mode. The command adds the variable to the currently opened watch list.

If a watch list is not open, then the variable is added to the **Watch 1** list and that view opens.

Also refer to

- [chapter 13.1.2 "Using watch lists" on page 196](#)
- [chapter 13.1 "Monitoring values" on page 191](#)

"Implement interfaces" command

Function: The command updates the implemented interfaces for a function block.

Call: Context menu if the function block (FB) is selected in the Project Explorer.

Prerequisite: The function block implements an interface that you have changed. For example, you have added a further method to the interface.



In object-oriented programming, if you derive a function block (FB) from a base function block, which implements one or more interfaces, for the purpose of inheritance, then the following applies:

When you execute the "Implement interfaces" command for the derived FB, all interface methods and interface attributes of the base FB are accepted into the derived FB in the form of "stubs" (without implementation). Then you are responsible for making sure that an "empty" method/attribute in the derived FB does not conflict with an implemented one in the base FB.

The following actions are taken to support you in this case: If there is a basic implementation for a method/attribute, IndraLogic adds a pragma attribute {error..} to the first line of the affected derived interface method or interface attribute that will generate the error message.

If there is no base implementation for the method/attribute, then there is a pragma attribute entry for a warning.

After editing the block, you must remove the error pragma attribute entry explicitly.

Also refer to

- [chapter ""Interface" object \(INTERFACE\)" on page 529](#)
- [chapter 9.18.2 "Implementing interfaces \(IMPLEMENTS\)" on page 147](#)

Command 'Limit results to current declaration'

Function: If several declarations are found, the command reduces the representation in the cross-reference list. It shows only the results for the declaration that is explicitly selected in the list.

Call: Context menu

Prerequisite: The cross-reference list is active. Multiple declarations for the searched symbol are listed as cross-references.

Also refer to

- [chapter 9.12.1 "Using the cross-reference list to find places of use" on page 129](#)

"Stop execution on handled exceptions" command

Function: The command stops the currently active application at the error location despite a programmed exception handling.

Call: The command is not available by default, but it can be configured via **Tools ▶ Customize**. For the command, go to the dialog **Customize ▶ Commands** in the **Online** category.

Prerequisite: The application is in online mode and contains a programmed exception handling with the `__TRY` and `__CATCH` operators.

Also refer to

- [chapter 17.3.61 "Operators `\_\_TRY`", `\_\_CATCH`", `\_\_FINALLY`", `\_\_ENDTRY`" on page 331](#)
- [chapter ""Customize" command" on page 641](#)

18.3 Dialogs

The IndraLogic interface dialogs are described on the help sites of the IndraLogic menu commands or IndraLogic objects.

The help on the **dialogs** only includes descriptions on dialogs, which

- are only displayed after a menu command or in a object editor due to further and cascaded calls
- or which are not part of the command help sites or object help sites due to their complexity (many subdialogs).

18.3.1 Dialog "Select function block"

Function: The dialog is used for selecting a function block for I/O mapping. The function block should be mapped to the I/O channel selected in the `<DeviceName> "I/O mapping"` tab or to the object selected in the `<Device-Name> "IEC objects"` tab.

Call: For objects in the Project Explorer that provide the following tabs in the generic device editor:

- **I/O mapping**, add via command button  FB for I/O channel
- **IEC objects**, add via command button 

The dialog provides all function blocks from the active application and the libraries included in the project which fulfill the following:

- The function block has the `{attribute 'io_function_block'}` attribute.
- The function block contains input or output parameters that match the channel type (input, output, data type) and has the `{attribute 'io_function_block_mapping'}` attribute.

When a function block is selected that provides multiple matching parameters, only the first one is mapped automatically to the channel. The others can only be assigned manually in the `<DeviceName> "I/O mapping"` tab.

After the function block is assigned, the parameter of the function block instance is entered in the Variable column of the mapping table. Then the path is composed as follows:

<ApplicationName>.<DeviceChannelName>.<Name of FB>.<continuous FB instance number>.<FB ParameterName>

Example: Appl.Out_4_Int_myScale_Output_Int_1.iOutput for the parameter iOutput of the first inserted instance of the function block myScale_Output.

Dialog elements

Search	Input field for searching for function block names
Type	Function blocks in the tree structure that match the channel type. Node: Application, library name(s)
Documentation	Shows the available documentation for the library selected in the tree or the library block

Also refer to [chapter "Attribute 'io_function_block', 'io_function_block_mapping'" on page 393](#)

18.3.2 “Breakpoint properties” dialog'

Function: The dialog is used to display or change the properties of the selected breakpoint in the "Breakpoints" view.

Symbol: 

Call: IndraWorks menu **Debugging ► Edit breakpoint**, or via the context menu of the breakpoint **Edit breakpoint**

Prerequisite: The application is in online mode and a breakpoint is set.

The dialog is identical to the "New breakpoint" dialog which is opened in the Debug menu by means of the respective commands, see [chapter 18.3.4 "Dialog 'New breakpoint'" on page 702](#)

18.3.3 “Prepare value”

Function: This dialog is used to prepare a value for a forced variable.

IndraLogic executes the prepared action with the next forcing.

IndraLogic opens the dialog in the following situations:

- When clicking in the field **Prepared value** of a forced variable in the declaration section
- When clicking in the inline monitoring field of a forced variable
- When clicking in the field **Prepared value** of a forced variable in the monitoring window

Prepare a new value for the next write or force operation	Value that IndraLogic writes to the variable with the next force operation
Cancel preparation with one value	IndraLogic deletes the prepared value.
Cancel the force without modifying the value.	IndraLogic retains the forced value and ends forcing. IndraLogic marks the variable with <Unforce>.
Release the force and restore the variable to the value it had before forcing it.	IndraLogic resets the forced value and ends forcing. The variable is marked with <Unforce and restore>.

Also refer to

- [chapter "Command 'Force values'" on page 631](#)

18.3.4 Dialog 'New breakpoint'

Function: In the dialog, you define the settings for a new breakpoint or data breakpoint. It is identical to the Breakpoint Properties dialog which is used in the Breakpoints view.

Symbol: 

Call:

- Via the IndraWorks menu **Debug ► New breakpoint...**
- Via the context menu of the corresponding line in the program code **New breakpoint...**

Prerequisite: The application is in online mode.

Tab "Condition"

The dialog defines the requirements for which program processing should halt at a breakpoint.



Using conditional breakpoints slows down code execution, even when the condition does not yield `TRUE`.

Tasks	
Only break if the breakpoint is hit in one of the following tasks	 : IndraLogic evaluates the breakpoint only if it is reached by specific tasks. The required tasks have to be enabled. For example, define a single "debug task" and prevent that other tasks are affected during debugging.
Number of hits	
Break always	The program always stops at this breakpoint.
Stop when the hit count is equal to	The program stops at the breakpoint if the breakpoint has been reached as often as selected or specified here.
Break when the hit count is a multiple of	
break when the hit count is greater than or equal to	
Condition	
Break, if true	 : IndraLogic analyzes the specified condition and only stops the program at the breakpoint when the result is <code>TRUE</code> . Define a condition as a valid Boolean expression. Examples: <code>x>100</code> , <code>x[y]=z</code> , <code>a AND b</code> , <code>boolVar</code> .

Tab "Data"

Prerequisite: This is used for the properties of a "data breakpoint".

On the tab, the variable or memory address is specified for which the data breakpoint is set or will be set.	
Break execution when the value of the variable or address changes	- Input of a qualified variable name : Selection of a variable in the Input Assistant dialog, in the "Watch variables" category Examples: Variable: <code>PLC_PRG.fb_DoSth.dwVariable</code>, address: <code>16#12A, 0x12A, 129</code>
Size	Number of bytes of the specified variable or memory address above which should be monitored for changes. When a new variable or memory address is specified, a value that matches the data type or memory is set automatically at first. Example: 4 for data type <code>DWORD</code>

Tab "Execution point settings"

In this tab, an existing breakpoint or data breakpoint can be converted into an execution point

Execution point (Execution does not stop at breakpoint)	: The breakpoint becomes an execution point. Processing does not halt at this point and the given code is executed. - Execution point of a breakpoint Enabled:  Disabled:  - Execution point of a data breakpoint Enabled:  Disabled: 
"Execute the following code"	Code that is executed when the execution point is reached. Loop structures (<code>FOR</code>, <code>WHILE</code>) and <code>IF</code> or <code>CASE</code> expressions are not valid.
"Output message in device log"	This option is only available, when enabling the option "Enable logging in breakpoints" in the settings of the compiler options. IndraLogic can output variables with the placeholder <code>{variable name}</code> in the message text.

Tab "Location"

Prerequisite: The command "New breakpoint" was selected.

POU	POU of the active application where the breakpoint is placed
Position	Position of the breakpoint in the POU. Entry as row and column numbers (text editor) or as network or element numbers
Instances	In case of function blocks, define whether the breakpoint should be set in the implementation or in an instance. : IndraLogic sets the breakpoint in the instance. For this option, select the "instance path". : IndraLogic sets the breakpoint in the implementation.
Enable breakpoint immediately	: The breakpoint is activated. : The breakpoint is not activated. For activation at a later point, click on the  button in the "Breakpoints" view.

Also refer to [chapter 12.2 "Using breakpoints" on page 179](#)

18.3.5 Dialog "Monitoring range"

Function: The dialog restricts the range of array elements whose values are displayed during monitoring.

Call: Click in the "Data Type" column field that belongs to the array variable.

Prerequisite: A POU is in online mode and is being monitored. In addition, a variable of the POU has the data type `ARRAY`.

Valid range	The scope of the array elements that are monitored. Example of a three-dimensional array: [1..10] [-3..3] [-10..10]
Maximum number of array elements	Number of elements of the array variables. Example: 1470
When you edit one of the settings "Start", "End", or "Scroll range of 1000 elements", both of the other settings are adapted automatically.	
Start	Index of the first array element whose value is displayed.
End	Index of the last array element whose value is displayed.
Scroll range of 1,000 elements	Scrollbar for selecting a range from the set of array elements.

Refer to [chapter 13.1.1 "Calling monitoring in programming object" on page 191](#)

18.3.6 "Properties" dialog

This dialog is used to configure the properties of an object in IndraLogic.

Depending on the object, the dialog contains different tabs, each handling a category of properties.

Call: Context menu of the object in the Project Explorer.

“Properties” - “General” dialog

Function: This dialog shows common information about the selected object.

Call: **View ► Properties** or context menu of the object, category **General**.

Prerequisite: An object is selected in the Project Explorer below the control or in the "General module folder".

Name	Object name as in device window or POU window
Object type	Type of object (for example, POU, application, or interface)
Open with	Type of editor to display or edit the object.

“Properties” - “Boot application”

Function: The settings in this tab define when and how a boot application is created from the application.

Prerequisite: The device supports the settings.

Call: Select application object, context menu **Properties** or menu **View ► Properties, Boot application** category.

<GroupName>

Create implicit boot application upon download	A boot application is created automatically when downloading the application.
Implicit boot application upon on-line change	A boot application is created automatically when for an online change.
Remind boot application on project close	Before closing the project, IndraLogic prompts to create the boot application.



Regardless of the presets defined here, you are always able to create a boot application explicitly when you log in.

Also refer to:

- [chapter 11.4 "Generating boot applications" on page 177](#)
- [chapter "Command 'Online change'" on page 619](#)

“Properties” - “Build” dialog

Symbol:

Function: This dialog includes options for compiling the object.

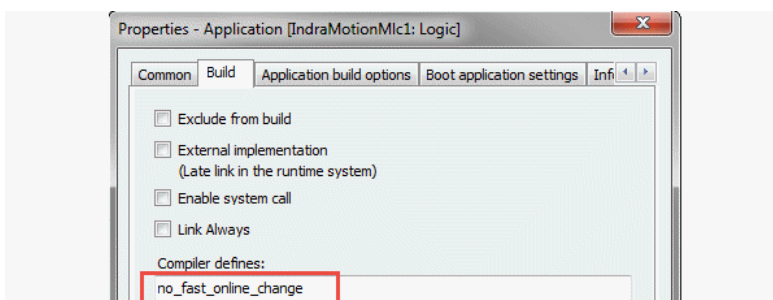
Call: Main menu **View ► Properties** or context menu of the object in the Project Explorer

Name	Description
Exclude from build	 : This object and inherently its child objects are not considered for the next compile process. The object entry in the views "Devices" or "POUs" is shown in green.
External implementation (Late linking in the runtime system)	 : IndraLogic does not generate any code for this object when compiling the project. The object is linked as soon as the project is running on the target system, provided it is available there (for example, in a library). The supplement (EXT) is provided for the object name in the "Devices" view or in the "POUs" view
Enable system calls	 : A system call (runtime system) for functions is possible. Background information: As opposed to IndraLogic 1x, the ADR operator in IndraLogic 2G can be used with function names, program names, function block names, and method names. It replaces the INSTANCE_OF operator. BUT: It is not possible to call function pointers within IndraLogic.
Link always	 : The object is marked by the compiler and therefore always included in the compile information. This means, that it is always compiled and loaded to the control.
Compiler defines	Here you can type in "Defines" or conditions for compiling the object (conditional compile). You can also type in the expression <code>expr</code> , which is used in these kinds of pragmas. Multiple entries are possible as a comma-separated list (see <code>{define}</code> instructions). Example: <code>hallo, test:='1'</code>

Using "compiler definitions"



To ensure the effect of the `init_on_onlchange` attribute in the entire application code, disabled the fast online change in general for the application by using the compiler definition `no_fast_online_change`.



Insert the definition into the application "Properties" ("Build" tab).

Also refer to

- [chapter "Attribute "init_on_onlchange" on page 392.](#)
- [chapter 17.6.3 "Conditional pragmas" on page 413](#)

"Properties" - "Access control" dialog

Function: The dialog is used to specify which user groups may execute which actions.

Call: Context menu **Properties**, **Access control** tab

Prerequisite: An object is selected in the Project Explorer below the control or in the "General module folder".

Groups, actions and rights	Table showing the right of the existing user groups to execute one of the following actions on the object: • View • Edit • Remove • Add/delete child objects Double-click on the "Rights" symbol to open a drop-down list of all rights.

Also refer to

- [chapter 6.3 "Protecting objects in the project via access rights" on page 61](#)

"Properties" - "Bitmap" dialog

Function: The dialog specifies whether and which bitmap file (image file) is added to the object. The bitmap is used in the graphic view of the library manger and the **Tools** view of the FBD/LD/IL editor.

Call: Main menu **View ► Properties** or context menu of the object in the Project Explorer.

Prerequisite: An object is selected in the Project Explorer below the control or in the "General module folder".

Transparency color	This color is shown transparently in the bitmap.

Also refer to

- [chapter 2.2 "Adding objects" on page 8](#)

"Properties" - "Application build options" dialog

Function: The dialog includes settings which IndraLogic uses when creating a boot application for the control.

Call: Main menu **View ► Properties** or context menu of the object in the Project Explorer

Download the application information	This feature requires compiler version $\geq 3.5.0.0$, runtime system version $\geq 3.5.0.0$ The information about the application contents is also downloaded to the PLC. We recommend that you keep this option activated because it enables a difference check between the current application and the application on the control. This compares the number of blocks, data, and memory locations. To receive information about the differences, click Details in the Applications tab of the device editor. This is also in the messages view that opens when you are downloading an application to the control when it is different from the one already on the control.
Use dynamic memory allocation	Memory is allocated dynamically for the application, for example when using the <code>__NEW</code> operator. In this case, define the Maximum size of memory (bytes). Note: The entire memory is not available for creating objects dynamically. Instead, the system always uses part of it for management information.

Also refer to

- [chapter 11.2 "Generating application code" on page 175](#)
- [chapter 11.4 "Generating boot applications" on page 177](#)

- [chapter 18.1.2 ""Application" object" on page 480](#)

The "Properties"- "Network variables" dialog

Symbol: 

Function: In this dialog, define network properties for the variable list selected in the Project Explorer. Furthermore, any variables in it that are declared as network variables are also available.

Call: Context menu of the variable list in the Project Explorer ► Properties and then the Network properties tab.

Network type	UDP
Task	Task of the current application that controls the variables to be sent. IndraLogic always sends the variables at the end of a task cycle.
Variable list ID	To identify the network variable list. Has to be unambiguous.
Pack variables	The size of the packages (telegrams) that are transmitted depends on the network type. In the case of "UDP", a package is 256 bytes. : IndraLogic bundles the variables for sending in packages in order to reduce the number of packages to send. In the case of variables of type <code>ARRAY</code> or structured data types, this can lead to the splitting of the variables into multiple telegrams. As a result, data inconsistencies are possible within these variables, even if the variable size is smaller than the package size. : IndraLogic generates one package per variable.
Transmit checksum	: A checksum is provided for each variable packet. The checksum is verified by the receiver in order to ensure that the variable definitions of sender and receiver comply with each other. A package with incorrect checksum is not accepted.
Confirmation	: IndraLogic sends an acknowledgment message for each received data package. If the sender does not receive a confirmation before it sends again, then an error is written to the diagnostic structure. Note: From version 3.5.7.0 of the <code>NetVarUdp</code> library, no reception channel is assigned anymore if no confirmed transfer is selected. Thus, network variables can also be exchanged between two controls on one hardware device.
Cyclic transmission, interval	IndraLogic sends the variables within the defined interval. Example of time definition: "T#70ms".
Transfer upon change, minimum gap	: IndraLogic sends the variables only if their values have changed. You can use "minimum gap" to define the least amount of time between two transmissions.

Transmit on event, variable	 : IndraLogic sends the variables as soon as the defined variable yields TRUE.
Settings	Log-specific settings; possible entries depend on the network library: Port: Number of the port that IndraLogic uses for data exchange with other network units. The Default value is "1202". You can change the current value in the Value field at any time: Select the field, press the <spacebar>, enter the value. Note: The other nodes in the network must define the same port! If more than one UDP connection is defined in the project, the port number is automatically adjusted in all configurations to the value set here. Broadcast Addr.: The preset value is "255. 255 . 255. 255" means that data is exchanged with all network devices. Change the current value in the Value field: Select field, press <spacebar>, enter the address or the address range of a subnetwork. Example: "197 . 200 . 100 . 255" in case of communication with all nodes having IP addresses in the range from "197 . 200. 100 . x".

Also refer to

- [chapter 10.4.4 "Editing communication" on page 169](#)
- [chapter 10.3.1 "Configuring a network variable exchange" on page 160](#)
- [chapter 18.1.16 ""Global Network Variable List \(receiver\)" object" on page 519](#)
- [chapter 18.1.15 ""Global Network Variable List \(sender\)" object" on page 519](#)

"Properties" - "Network properties" dialog

If the device supports the network functionality, then the current network settings for a GNVL (global network variable list) can be displayed and changed in the **Properties** dialog of the object.

These are the settings that were used when adding the GNVL in the **Add Network Variable List (Receiver)** dialog.

Also refer to:

- [""Add Global Network Variable List \(receiver\)" dialog" on page 520](#)
- [chapter 10.3.1 "Configuring a network variable exchange" on page 160](#)

"Properties" dialog - "CFC execution order" tab

Function	Configuration of execution order mode in CFC object networks.
Call	In the Project Explorer via the context menu "Properties" of a CFC object
"CFC execution order" tab	In the CFC editor, elements and networks can be positioned freely. Two modes are available to define the execution order and to prevent that the CFC network is undefined due to free positioning. • Auto Data Flow Mode • Explicit Execution Order Mode

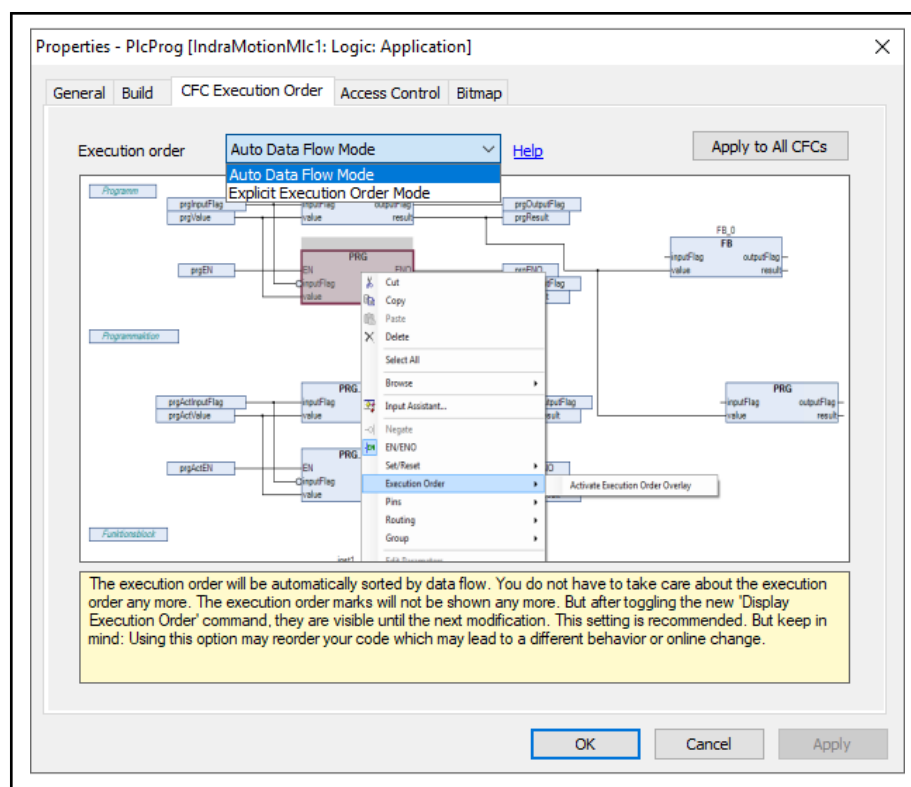


Fig. 18-73: Tab 'CFC Execution Order'

Elements of the "CFC execution order" tab

Modes to define the execution order in CFC networks	Auto data flow mode In this mode, the execution order is determined automatically by data flow, or in case of ambiguity, by network topology. The POU's and the outputs are numbered internally. The networks are executed from top to bottom and left to right. Advantage: The automatically defined execution order is optimized by time and by cycle. You do not need any information about the internally managed execution order during the development process. The following commands are provided afterwards in the CFC ► Execution Order menu: • Display Execution Order • Set Start of Feedback The elements in the CFC editor are displayed without markers and without numbering. It is not possible to change the execution order manually. For networks with feedback, you can also set a starting point. Explicit execution order mode In this mode, you can define the execution order explicitly. To do this, the elements are displayed in the CFC editor with markers and numbering, and menu commands are provided for defining the order. The following commands are provided afterwards in the CFC ► Execution Order menu: • send to front • Send to Back • Move Up • Move Down • Set Execution Order • Order by Data Flow • order by topology Note: Up to IndraWorks 14VRS, this behavior was the regular behavior of CFC POU's. Please note that it is the user's responsibility to adjust the execution order and assess the consequences and impacts. The execution order is permanently displayed.
Command "Apply to all CFCs"	This command changes the mode for all other CFC objects in the project to the mode selected in the list

Tab. 18-16:

- Also refer to
- [chapter "Command 'Send to front'" on page 650](#)
 - [chapter "Command 'Send to back'Symbol:' on page 651](#)
 - [chapter "Command 'One ahead'" on page 651](#)
 - [chapter "Command 'Order by data flow'" on page 652](#)
 - [chapter "Command 'Order by topology'" on page 652](#)

“Properties” - “SFC settings” dialog

Function: The dialog defines default settings for all POU's programmed in SFC and used in the project.

Call: Menu **View ► Properties**, context menu of a POU programmed in SFC in the Project Explorer below a control or in the "General module folder".

Variables (“SFC flags”)

List of all possible SFC flags	Active : The SFC flag is enabled and thus included while processing. Declaration : The SFC flag is automatically declared. If Declare is selected but Use is not selected, the variable is declared, but the flag has no function during processing. Note: When an SFC flag variable is declared manually, disable the declaration for this flag in the SFC settings. Otherwise, the automatically declared flag overwrites the manually declared flag. Note: Automatically declared variables are visible in the declaration section of the SFC editor only in online mode.
Use defaults	: The settings present in this dialog, overwrite the SFC settings defined in the individual POU's.

Also refer to

- [chapter "SFC flags" on page 241](#)

Compiling

Code generation

Calculate active transitions only	: IndraLogic only generates code for currently active transitions.

“Properties” - “Linking to file” dialog

Function: The dialog defines the link to an external file with the content of the Global Variable List (GVL). Export the GVL either to such a file or import it from such a file.

Call: Menu **View ► Properties** context menu of a POU of type GVL.

Export before compile	: Every time the project is compiled (Create ► Build or <Shift> + <F11>), IndraLogic saves a file with the extension <code>gvl</code> in the path specified in the “File name” field.
Import before compile	: An existing export file is automatically read in for each project compilation. Thus, a GVL exported from another project can be imported to establish the communication via network variables for example.

Also refer to

- [chapter 18.1.11 "“GVL” object - Global Variable List" on page 512](#)
- [chapter 10.3.1 "Configuring a network variable exchange" on page 160](#)

“Properties” - “Image Pool” dialog

Function: In this dialog, you can configure the selected image pool.

Call: **View ► Properties** of an **Image pool** object type; context menu of an **Image pool** object type

Download used images only	 : Instead of loading all images from the image pool, IndraLogic loads only the images that are actually used in the application on the control.
Internal	 : IndraLogic does not provide the image pool in the Tools view. You cannot drag these images to the visualization.

Symbol library settings

Mark library as symbol library	Marks the image pool as a symbol library for use in a visualization. The symbol library receives the key <code>VisuSymbolLibrary = TRUE</code> as file property in the library information. The <code>VisuElements</code> library is inserted automatically as a placeholder library in the "General module folder" of the library manager. Prerequisite: A library project is open. IndraLogic displays symbol libraries that are installed in the repository in the Project Settings (Visualization category, Symbol Libraries tab).
Text list for symbol translation	Select the text list from the drop-down list that contains the translated texts for the image pool.

Also refer to

- [chapter 20.1.2 "Displaying and labeling visualization elements" on page 759](#)
- [chapter 18.2.15 "Menu 'Text list'" on page 684](#)
- [chapter "'Project settings' dialog - 'Visualization'" on page 719](#)

"Properties" - "Options" dialog

Options (control)

Function: The dialog includes settings to monitor and log in for an object of type (example: "Safety_CPU_CFL01_1_F1").

The options available depend on the device description.

Call: Context menu of the device object or the menu **View ► Properties** if the device object is selected.

Monitoring interval (ms)	Monitoring interval (10 ms - 1000 ms)

Interactive login mode

This mode is used to impede an accidental login to another control.	
None	No interaction with the user upon login. Corresponds to the current response of the older versions
Enter ID	Upon login, IndraLogic requests and ID. The ID is stored on the control. No login without valid ID At next login, IndraLogic does not request the ID anymore if computer name, user name, device name and device address were not changed. The information is stored in the project options.
Key down	When logging in, a dialog is displayed requesting you to press a key on the control. The timeout for this action is defined in the device description.
Flashing (LED)	The connected control flashes upon login.

Symbol configuration

Variable access synchronous to IEC task	☐ : Default setting, consistent access not permitted
	☒ : Consistent access permitted The setting is only applied if all applications and boot applications are loaded again to the control. Note: If the option is enabled, the jitter can increase for all IEC applications on this device! The consistent access can disturb the real-time capability.

Also refer to

- [chapter 8.1 "Project Explorer and device editor" on page 69](#)
- [chapter 18.1 "Objects" on page 479](#)

18.3.7 “Project settings” dialog

Symbol: 

Function: This object contains the basic configuration of the project. They can be edited in the **Project settings** dialog.

Call: Double-click on the **Project settings** object in the Project Explorer, **General module folder**.

IndraLogic saves the project settings directly in the project. If you then transfer a project to another system for example, the **Project settings** object is also transferred with it, without a project archive being required.

The project settings apply to the complete project. They provide setting options for various categories such as **SFC** or **Compiler options / compiler warnings** or **StaticAnalysisLight** and **Visualization**.

“Project settings” - “SFC” dialog

Symbol: 

Function: This dialog is used for configuring the settings of SFC objects. The properties of each new SFC object automatically have the configured settings.

Call: Menu **Project ► Project settings**, category **SFC**

Prerequisite: A project is open.

“Variables” tab

Implicitly generated variables for checking and monitoring the processing in an SFC diagram.	
Active	☒ : The respective variable is used.
Declare	☒ : The variable is created automatically. Otherwise, you have to declare the variable explicitly if you intend to use it (Use is selected).
Apply to all	IndraLogic applies the changes made in this dialog to already existing SFC objects. IndraLogic selects the Use defaults checkbox in the properties of the SFC POUs.



Automatically declared variables are visible in the declaration section of the SFC editor only in online mode.

“Compile” tab

Code generation

Calculate active transitions only	 IndraLogic only generates code for currently active transitions.

SFC Library

This part of the dialog is only available for compiler versions < 3.4.1.0.	
Company Title Version	Defines the SFC library that IndraLogic uses by default.
Namespace	Enables unique references to libraries. Required when various versions of the library are available on the system. Please make sure that there are no discrepancies between the namespace defined in the library manager and the namespace defined for the individual object. The <code>SfcIec.library</code> data is used for the default settings that IndraLogic provides with the default profile.



Each SFC block stores the information via the library version that applied when you added the block. This can cause you to use multiple library versions within the same project. To prevent this, no specific versions of the `IecSfc.library` can be entered starting from the compiler version 3.4.1.0. The library version, which you use for all SFC blocks in the project, is defined with a placeholder.

IndraLogic resolves the placeholder depending on the compiler version in use. The allocation of the library version to the compiler version is defined in the library profile.

Also refer to

- [chapter "SFC flags" on page 241](#)

"Project settings" - "Compiler options" dialog

Symbol: 

Function: This dialog is used to configure the compiler options.

Call: Menu **General module folder** ► **Project settings**, category **Compiler option**.

Prerequisite: A project is open.

Compiler version

Fix Version	Defines the compiler version which IndraLogic uses when compiling and loading for compilation. Example: 3.5.14.0 for the version 3.5 SP14 The compiler version is only displayed. It cannot be edited.

Settings

Allow Unicode characters for identifiers	Cleared by default because using Unicode characters in identifier names is not permitted in the IEC standard. Can be required for some Asian languages.
Replace constants	☒ (Default): For each constant of scalar type, that is not for <code>STRING</code> , <code>ARRAY</code> or structures, IndraLogic loads the value directly. In online mode, IndraLogic marks the constants with a symbol that is prepended to the value in the declaration editor or monitoring window. In this case, access is not possible, for example using an <code>ADR</code> operator, forcing and writing. ☐ : Access to constants is possible, but it prolongs the computation time.
Enable logging in breakpoints	For breakpoints that are defined as execution points, you can create a message text in the Execution point settings dialog. IndraLogic outputs this text in the device log when the application halts at the execution point.

Compiler warnings

Maximum number of warnings	Refers to the warnings that IndraLogic prints to the messages view. Define the selection of displayed compiler warnings in the Project settings dialog in the Compiler warnings category.

Also refer to

- [chapter ""Project settings" - "Compiler warnings" dialog" on page 716](#)
- [chapter ""Log" tab" on page 495](#)
- [chapter 18.3.2 ""Breakpoint properties" dialog" on page 701](#)

"Project settings" - "Compiler warnings" dialog

Symbol: 

Function: This dialog is used for selecting the compiler warnings that IndraLogic displays in the messages view during a compilation.

Call: Menu **General module folder** ► **Project settings**, category **Compiler warnings**.

Prerequisite: A project is open.



You define the maximum number of listed warnings in the **Compiler options** dialog.

Also refer to

- [chapter ""Project settings" - "Compiler options" dialog" on page 715](#)
- [chapter ""Build" command" on page 611](#)

"Project settings" dialog - "Static analysis light"

Symbol: 

Function: The dialog activates the tests that the light version of the static analysis performs in CODESYS each time code is generated.

Call: In the Project Explorer **"General module folder" ► Project settings**, category **Static analysis light**.



You can exclude lines of code from the static code analysis by marking the code with the pragma `{analysis ...}` or the pragma `{attribute 'analysis' := '...')}`.

Additional compilation tests

SA033: Unused variables	Finds variables that are declared, but not used within the compiled program code. Note on GVL variables in the "General module" folder: If there are several applications in the project, IndraLogic only takes objects under the active application into consideration. If there is only one application, the objects in the "General module folder" are also taken into consideration.
SA0028: Overlapping memory areas	Detects the locations where two or more variables reserve the same storage space. For example, this occurs for the following declarations: <code>var1 AT %QB21: INT</code> <code>var2 AT %QD5: DWORD.</code> In this case, both variables use byte 21, which means that the memory range of the variables overlap.
SA0006: Write access from multiple tasks	Detects variables that are written by more than one task.
SA0004: Multiple write access to output	Detects outputs that are written to more than one location. Note: No error is reported when an output variable (<code>VAR_IN_OUT</code>) is written in different branches of <code>IF</code> and <code>CASE</code> statements. Note: Normally, this rule cannot be disabled via pragmas.
SA0027: Multiple use of identifiers	Detects multiple uses of a name/identifier for a variable or an object (POU) within the scope of a project <i>The following cases are detected:</i> • The name of an enumeration constant is the same as in another enumeration in the application or used in an included library. • The name of a variable is the same as an object in the application or an included library. • The name of a variable is the same as for an enumeration constant in and enumeration in the application or an included library. • The name of an object is the same as another object in the application. • The name of a variable is the same as the name of a method. • The name of an object is the same as the name of a superordinate object ("parent object").
SA0167: Temporary function block instances	Detects function block instances that are declared as temporary variables. This concerns instances that are declared in a method or in a function or as <code>VAR_TEMP</code> , and therefore are reinitialized in each processing cycle and for each block call.

Examples**SA0003: Empty statements**

```
;
(* Comment *);
iVar;
```

SA0006: Concurrent access

```
FUNCTION_BLOCK ADD_FB
g_iTemp1 := g_iTemp1 + INT#1;
```

```
PROGRAM PlcProg //controlled by MainTask
g_iTemp1 := g_iTemp1 + INT#2;
g_xTemp2 := g_iTemp1 > INT#10;
```

```

PROGRAM PlcProg_1 //controlled by SubTask
g_iTemp1 := g_iTemp1 - INT#3;
g_xTemp2 := g_iTemp1 < INT#-10;

```

SA0004: Multiple write access to output

```

VAR_GLOBAL
g_xVar AT %QX0.0 : BOOL ;
g_iTest AT %QW0 : INT ;
END_VAR

```

```

PROGRAM PlcProg
IF iCondition < INT#0 THEN
g_xVar := TRUE;
g_iTest := INT#12;
END_IF

```

```

CASE iCondition OF
INT#1:
g_xVar := FALSE;
INT#2:
g_iTest := INT#11;
ELSE
g_xVar := TRUE;
g_iTest := INT#9;
END_CASE

```

SA0006: Write access from multiple tasks

```

FUNCTION_BLOCK ADD_FB
g_iTemp1 := g_iTemp1 + INT#1;

PROGRAM PLC_PRG // Controlled by MainTask
g_iTemp1 := g_iTemp1 + INT#2;
g_xTemp2 := g_iTemp1 > INT#10;

PROGRAM PLC_PRG_1 //Controlled by SubTask
g_iTemp1 := g_iTemp1 - INT#3;
g_xTemp2 := g_iTemp1 < INT#-10;

```

SA0027: Multiple use of name

```

PROGRAM PlcProg
VAR
ton : INT; // error SA0027
END_VAR

```

SA0029: Different notation in implementation and declaration

In the Project Explorer, there is the `PLC_PRG` and the `fnC` (function) function block.

```

PROGRAM PLC_PRG
VAR
iVar:INT;
_123test_var_:INT;
END_VAR
iVar := iVar + 1; // notation different to that in
                  the declaration part -> SA0029
_123TEST_var_ := _123test_var_INT; // notation different to that
                                   in the declaration part -> SA0029
FnC(); // notation different to that in the devices tree -> SA0029
END_VAR

```

SA0167: Temporary function block instances

```

PROGRAM PLC_PRG
VAR
END_VAR

```

```
VAR_TEMP
  yafb: AFB;
END_VAR

FUNCTION Fun : INT
VAR_INPUT
END_VAR
VAR
  funafb: AFB;
END_VAR

METHOD METH: INT
VAR_INPUT
END_VAR
VAR
  methafb: AFB;
END_VAR
```

Also refer to

- chapter 9.11.2 "Analyzing code statically" on page 127

“Project settings” dialog - “Visualization”

Symbol: 

Function: The dialog is used to configure the project-wide settings for objects of type **Visualization**.

Call: General module folder ► Project settings, category Visualization

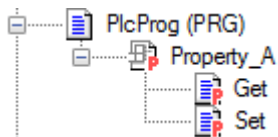
Prerequisite: A project is open.

“General” tab

Visualization directories

Text list files	Directory with text list available in the project to configure texts for different languages. IndraLogic uses this directory to export or text lists for example. Click on  and the Scan folder dialog is displayed. A directory in the file system can be selected using this dialog.
Image files	Directory with image files available in the project. Multiple folders are separated from each other by a semicolon. IndraLogic uses this directory to export or import image files for example. Click on  and the Scan folder dialog is displayed. A directory in the file system can be selected using this dialog.

Extended settings

Enable property handling in all element properties	Prerequisite: Visible is enabled. : A visualization element can also be configured with a  property in those properties, in which an IEC variable is selected. IndraLogic then generates additional code for the property handling while compiling a visualization. Prerequisite: Your IEC code contains at least one object of the type POU property. 

Also refer to

- [chapter "Object 'Property'" on page 537](#)

"Symbol libraries" tab

Symbol libraries	List of all installed system libraries, such as VisuSymbols
Assigned	 : Symbol library is selected in the project and IndraLogic provides it in the Tools view of a visualization.  : Symbol library is installed in the library repository, but IndraLogic does not provide it in the Tools view of a visualization.

Tab. 18-17: **Visualization symbol libraries**

Also refer to

- [chapter 20.3.6 "'Add visualization" dialog" on page 794](#)

18.3.8 "Properties"- "Text list"

Function: The dialog allows for setting the basic properties of the selected text list.

Call: In the Project Explorer via the context menu of an object of type "Text list"**Properties**

Download by visualization	 : The text list is downloaded with the visualization to the controller.
Internal	 : The text list can be used only in a library.

Also refer to

- [chapter 18.1.23 "'Text list" object" on page 553](#)

18.3.9 "Options" dialog

Function: The dialog is used to select IndraWorks/IndraLogic options to configure response and appearance of the user interface.

IndraWorks saves the current settings as default settings on the system.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**

"Options" dialog - "General Settings"

Function: This dialog is used to configure the general settings to work with IndraLogic.

Call: Menu: **Tools** ► **Options** ► **IndraLogic 2G** ► **General settings**.

Open object after adding:	When this option is selected, new objects that are inserted into the Project Explorer are opened immediately.
Show device type in the Project Explorer:	The name of the object is extended with its device type.
Activate PLC logger:	Using this button, the "Log" tab of the device dialog of every control can be activated.

Enable PLC shell:	The control shell is a text-based control monitor (terminal). Commands to query certain information from the control are entered into the input line and send to the control as string. The returned response string is shown in the result window of the browser. This functionality is used for diagnostic and debugging purposes. The commands supported by the target system can be extended by the respective control vendor. The PLC Shell Command Handler can be implemented either in a C-component in the runtime system or as function block in an IEC library or a project.
Enable PLC trace (IL trace) (only for support):	

“Options” - “CFC editor” dialog

Symbol: 

Function: this dialog is for the configuration of the settings for editing and printing in the CFC editor.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **CFC editor**

“General” tab

Enable automatic connection	 : If you drag a CFC element onto the work area of the editor and insert it, IndraLogic automatically connects together unconnected pins that "touch" one another. Make sure that you do not create unwanted connections when shifting elements!
Prepare values in the implementation part	 : In online mode, variable values can also be prepared for writing and forcing in the implementation section of the CFC function block. In addition, IndraLogic displays the values you have just prepared in the inline monitoring box of the variable in angle brackets.

“View” tab

Show grid points	 : Grid points, at which you can position the elements, are a validity range in the editor.
Show function block symbol	 : Existing function blocks that are linked with a bitmap are displayed by in the CFC editor as symbols. Prerequisite: You have either created the link for a function block or a function in the object properties or loaded it via a library.
Edit line colors	Opens the Edit line colors dialog to define the colors of the connecting lines, depending on the data type applied. The lines appear in offline and online mode in these colors, unless IndraLogic paints over them with the thick black and blue lines used to display the Boolean data flow. • Add type: Adds a data type to the list. • Delete type
Fonts	Display of the font and button for changing the font.

Also refer to

- [chapter ““Properties” - “Bitmap” dialog” on page 707](#)

“Print” tab

Setting the layout options	
Page scaling	Page or Poster
Scaling	Possible values: 20 % - 200 %

Also refer to

- [chapter "Programming in the CFC editor" on page 97](#)
- [chapter 17.1 "Programming languages and editors" on page 223](#)

“Options” – “Declaration editor” dialog

Symbol: 

Function: This dialog is for the configuration of the display settings for the declaration editor.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **Declaration editor**

Only textual	Textual view of the declaration editor
Tabular only	Tabular view of the declaration editor
Can be toggled between textual and tabular	The declaration editor offers two buttons for switching between the textual and tabular views: : Textual view : Tabular view The following option defines the view that appears by default when opening a programming object: • Always textual • Always tabular • Save last setting (per object) • Save last setting (global)

Also refer to

- [chapter 9.2.1 "Using the declaration editor" on page 85](#)

Dialog ‘Options’ - ‘Device editor’

Symbol: 

Function: This dialog includes settings for displaying the device editor.

Call: IndraWorks menu **Tools** ► **Options** ► **IndraLogic 2G** ► **Device editor**.

“View” tab

Display generic configuration editors	 : This tab with the list of device parameters is available in the device editors of parameterizable devices.
Create Cross References for IEC addresses (Clean necessary)	 : Creates the cross-references for unmapped I/Os.
Communication page	• Simple mode: The Communication tab is displayed as described in the corresponding section in the online help. • Classic mode: The "Communication" tab of the device editor opens as a dual view: The left side shows the current configured gateway channels in a tree structure. The right side shows the respective data and information.

Show implicit files for application download on the editor of a PLC	 The tab Synchronized files is available in the device editors. Synchronized files are downloaded to the PLC at the time of application download. These can be external files that were added to the application, or implicit files such as a source code archive.
Show "Access rights" tab	 The "Access Rights" tab is available in the device editors. Note: Depending on the device, the device description may overwrite this setting.

Also refer to:

- [chapter "Generic device editor" on page 490](#)
- [chapter ""Synchronized files" tab" on page 494](#)

"Options" - "FBD, LD and IL" dialog

Symbol: 

Function: This dialog is used for configuring the display options for the FBD/LD/IL editor.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **FBD, LD and IL**

"General" tab

View

Show network title	Displays the network title in the top left corner of the network.
Show network comment	Displays the network comment in the top left corner of the network. When the network title is also shown in IndraLogic, the comment is shown in the line below.
Show function block symbol	Displays the box symbol in the box element in the FBD and LD editor. The standard operators are provided with symbols as well.
Show operand comment	shows the comment that you indicated for a variable in the implementation part. The operand comment refers to the local occurrence of the variable only, as opposed to the "symbol comment". The comment is truncated automatically depending on available space. To limit the comment to a defined width, select the option "Fixed size for operand fields".
Show symbol comment	displays the comment above the variable name that you gave to a variable or a symbol during the declaration. You can also assign a local "operand comment" in addition to or instead of the symbol comment.
Show symbol address	If an address is assigned to a symbol (variable), then this address is displayed above the variable name.
Show network separators	A separator is displayed between the individual networks

Behavior

Placeholder for new operands	The operand field of pins for the new function block is left blank (instead of "???").
Empty operands for function block pins	Adds empty operands instead of ???.

Fonts

Click the input field to open the **Font** dialog.

Specified size for operand fields	 Edit operand size can be enabled.
Edit operand size	The Operand Sizes dialog opens for setting the number of characters and lines.

“FBD” tab

View

Networks with line breaks	 Display of the network with line breaks so that so that IndraLogic can show as many blocks as possible in the current width of the window.
Connect function blocks with straight lines	 The length of the lines between the elements are fixed and short.

Behavior

Network content	Drop-down list: Content of a new network.
Selection after insertion	Drop-down list: Element that IndraLogic selects after inserting a new network.

“LD” tab

View

Networks with line breaks	 Display of the network with line breaks so that so that IndraLogic can show as many blocks as possible in the current width of the window.

Behavior

Network content	Drop-down list: Content of a new network.
Selection after insertion	Drop-down list: Element that IndraLogic selects after inserting a new network.

“IL” tab

View

Enable IL	The IL implementation language is available in the development system.

Behavior

Network content	Drop-down list: Content of a new network.
Selection after insertion	Drop-down list: Element that IndraLogic selects after inserting a new network.

“Print” tab

Layout options

Fit method	Selection list for resizing.
Avoid cutting of elements	Elements that do not fit on the page are printed on the next page.
Mark connections on adjacent pages	Enabled for selection when Avoid cutting of elements is selected.

Also refer to

- [chapter 17.1 "Programming languages and editors" on page 223](#)

“Options” dialog - “IndraLogic 1x converter”



Symbol

Function: The dialog is used to define the image when converting an IndraLogic 1x project.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **IndraLogic 1x converter**

Devices

List and definition of the device images during a conversion. All device images of this list are also applied during the next conversion of an IndraLogic 1x project. Thus, the mapping definition does not have to be created again if the same device is used in several projects.	
Source device	Double-click on an entry or an empty field to open the dialog Select original device . IndraLogic lists all device that can be processed by the converter.
Target device	Double-click on an entry or an empty field to open the dialog Set target system library . • Select the target system from the list of installed devices: Confirm the selection with OK. • None: The source device is not available in the new project. This means as well that application-specific objects, e.g. the task configuration, are not applied.

Visualizations

Defines the handling of the representation variant “Target visualization” and “Web visualization” when opening an IndraLogic 1x project.	
Import target visualization Import web visualization	<i>Drop-down list</i> • Attempts to imitate the old behavior: Option is available if the currently device used supports this option. This option is used to ensure that the new target visualization settings or web visualization settings correspond as exactly as possible to the settings in the old project. • Prompt user once: Only upon the first import of an "old" visualization project, IndraLogic prompts to define the settings. IndraLogic then applies the settings automatically for all other visualization imports. • Prompt user: Every time an "old" visualization project is opened, IndraLogic prompts to define the settings • Device description settings: IndraLogic makes the settings as per the entries in the device description of the installed target device. • Attempts to imitate the old behavior: Option is only available if the currently device used supports this option. This option is used to ensure that the new representation variant settings for the target visualization or web visualization correspond as exactly as possible to the settings in the old project.



Target visualization does not support the access to `VAR_IN_OUT` variables, as it is not supported by the compiler. This response corresponds to the response of the IndraLogic 1x target visualizations.

Also refer to

- [chapter 2.4 "Opening IndraLogic 2G project" on page 10](#)

“Options” – “Libraries” dialog

Symbol: 

Function: This dialog is used to manage mappings and library references that IndraLogic uses while converting an old project. If you have not yet stored any mapping for a certain library, you must redefine the mapping each time when opening an old project in which this library is integrated.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **Libraries**

A mapping defines what a library reference looks like following the conversion of the project to the current format. There are three options:

- Retain the reference. This means that IndraLogic similarly converts the library into the current format (*.library) and installs it in the local library repository.
- Replace a reference with another reference. This means that one of the installed libraries replaces the library that was integrated until now.
- The reference is deleted. This means that the converted project no longer integrates the library.

Set target system library

IndraLogic applies all the listed mappings to the library references of an old project the next time it is converted.

Hence, repeat the mapping definition if the same library is integrated again in a project to be converted.

A new mapping can be entered into the last line.

Source library	Library path included in the project before conversion A double-click on an entry makes the field editable and the button for the input assistant is displayed.
Target library	Name and storage location of the library that is to be included in the project after conversion. A double-click on an entry opens the dialog Set target system library .

“Options” - “Refactoring” dialog

Symbol: 

Function: This dialog is used for defining operations in the project for which refactoring is automatically suggested. The refactoring functionality supports the user in improving the process.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G** ► **Refactoring**.

Suggest refactoring for the following operations

AutoDeclare When changing the variable name in a declaration by calling the auto declaration ([Shift]+[F2]), the enabled option "Apply changes by means of refactoring" is displayed. The "Refactoring" dialog opens and the variable can be changed in the project.	
During renaming of variables	 : When renaming variables, you are prompted whether IndraLogic should perform "Automatic refactoring".
For adding or removing variables or for changing the namespace	 : Delete the name from the auto declaration (dialog "Declare variable") and exit the dialog with OK . The "Refactoring" dialog opens and the variable can be removed from the project.

Unit conversion editor	
During renaming of unit conversions	 : When you change the name of a conversion in the unit conversion editor, you are prompted whether IndraLogic should perform "Automatic refactoring" when renaming.
Mapping editor	
During renaming of variables	 : When renaming a variable name in the device editor, I/O mapping tab, you are prompted whether IndraLogic should perform "Automatic refactoring" for the renaming.
Navigator	
During renaming of objects	 : When renaming the name of an object in the Project Explorer or in the "General module folder", you are prompted whether IndraLogic should perform "Automatic refactoring" for the renaming.
Tabular declaration editor	
During renaming of variables	 : When you change the name of a variable in the tabular declaration editor, you are prompted whether IndraLogic should perform "Automatic refactoring" when renaming.

Also refer to

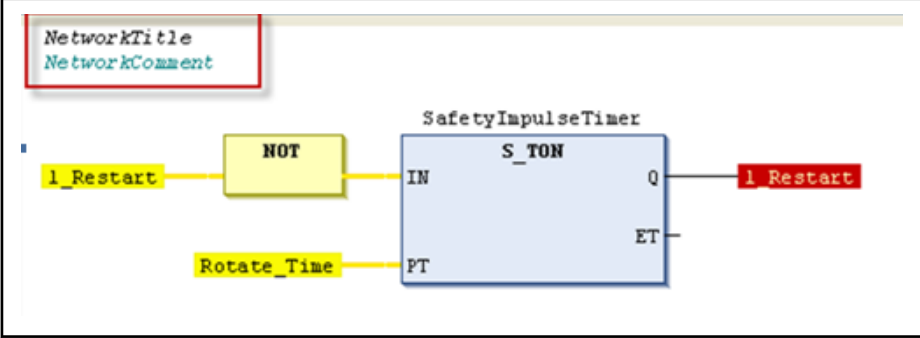
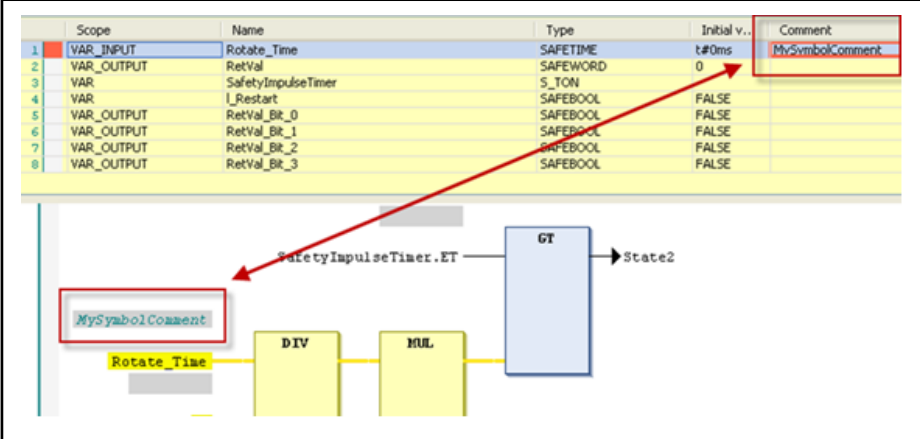
- [chapter 9.14 "Refactoring" on page 132](#)
- [chapter 9 "Programming applications" on page 83](#)
- [chapter "'Rename refactoring' command" on page 579](#)

"Options" dialog - "Safe FBD options"

Function: This dialog is used to extend the settings for editing and printing in the FBD editor for Safety applications.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **Safe FBD options**

Representation of connecting FBD elements	
Connect function blocks with straight lines	 : If this option is selected, the elements are arranged in a network so that the lines between them receive a fixed, short length and the width required for displaying the networks is reduced to the largest possible extent. Consequently, the function block boxes might be extended vertically in order to create enough space for the input and output elements.  : If this option is not selected, the elements maintain their standard size and the connection lines are modified to fit within the space required.
Optional code commenting	
In addition to the object comments, the program code can be commented on. No warnings are displayed in case of missing code comments. Optional code commenting can be selected in this dialog	
Show network title	 : Each network of the program code of the POU's of the safety application of a project can be assigned a title. The network title is edited in the first line of the FBD network

Show network comment	: Each network of the program code of the POU's of the safety application of a project can be assigned a comment. The network comment is edited in the second line of the FBD network. (Exception: If the option "Show network title" is not selected, the network comment is edited in the first line.)  <i>Fig. 18-74: Network title and network comment</i>
In POU's and GVL's	Comment on each user-defined variable declaration (optional)
Show symbol comment	: Variable comments entered into the declaration window of the FBD editor are displayed in the implementation window above the location of use of the variable.  <i>Fig. 18-75: Symbol comment, here from POU declaration</i>

"Options" - "SFC editor" dialog

Symbol: 

Function: Use this dialog for configuring the settings of the SFC editor.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **SFC editor**

Also refer to

- [chapter "Programming in Sequential Function Chart \(SFC\)" on page 102](#)
- [chapter 18.2.7 "Menu 'AS'" on page 642](#)
- [chapter "SFC \(sequential function chart\) editor" on page 236](#)

"Arrangement" tab

Elements

This defines the dimensions of the SFC elements: step, action, qualifier, property.

The values are specified in grid units.

1 grid unit = Font size currently set in the text editor options (Text area/Font).

The settings are always active immediately in all open SFC editor views.

Step height	Possible values: 1-100
Step width	Possible values: 2-100
Action width	Possible values: 2-100
Qualifier width	Possible values: 2-100
Properties width	Possible values: 2-100

Character set

The exemplary text shows the currently set font. Click on it to change the font.

Step actions

Standard insertion method	• "Copy reference": The reference to the action objects called by the step are copied when copying the step. The copied and the new step then call the same action. • "Duplicate implementation": The reference to the action objects called by the step are fixedly connected to this step. When copying the step element, new action objects are created for the new step, i.e. the implementation is duplicated. • "Always ask": When inserting a step action, it is always prompted whether its actions are to be duplicated when copying the step element or whether the reference is to be applied to an existing action. Note: If a step already includes an embedded action, newly inserted actions of this step are also embedded. Vice versa, the newly inserted actions are not embedded if the step already contains an unembedded action. In these cases, you are not prompted for the desired duplication mode anymore.

Embedded objects

Show action and transition objects in the navigator	 : Action and transition objects fixedly linked to a step in the SFC function block ("embedded") are shown in the tree view "Devices" or "POUs".

"View" tab

Visibility of properties

List of the element properties of the categories **General** and **Specific** and the definition of the display options.

Property	Define the element properties next to the element in the SFC diagram.
Value	 : Display of the property value.
With name	 : Display of the property value with name.

Online

Show step time	 : In online mode, shows the step time on the right of the steps.

“Options” dialog - “Smart coding “

Symbol: 

Function: This dialog is for configuring the settings for easier coding.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **Smart coding**

Declare unknown variables automatically (AutoDeclare)	 : The Auto Declare dialog opens when you type an undeclared identifier into a implementation language editor and then click away from the input line. In order for the AutoDeclare function to be available in the ST editor as well, the "Enable for ST editor" option also has to be selected.
Enable for ST editor	Prerequisite: The "Declare unknown variables automatically (AutoDeclare)" option is selected.  : The AutoDeclare function is also available in the ST editor.  : The AutoDeclare function is not available in the ST editor.
Show all instance variables in input assistant	 : The "List components" function also lets you select the local variables of a function block instance.  : With the "List components" function, only the input variables and output variables of a function block instance can be selected.
Show symbols of system libraries in input assistant	System libraries are inserted in the library manager automatically and displayed in light gray.  : Symbols, such as global variables, data types, and function blocks are provided in the input assistant.  : The symbols of the system libraries are not available in the input assistant.
List components after typing a dot (.)	 : Enables the "Listing components" function. That means: When you type a dot (.) at a location where IndraLogic expects an identifier, a drop-down list appears with possible code.
List components immediately after typing	Prerequisite: The List components after typing a dot (.) check box is selected.  : While you type code, a drop-down list appears with possible identifiers and operators
Insert with namespace	 : IndraLogic automatically adds the namespace before the identifier.
Convert keywords to uppercase automatically (AutoFormat)	 : IndraLogic displays all keywords in uppercase.
Automatically list selection in cross reference view	 : The cross reference list automatically shows the references of the variables/POUs/DUTs currently selected or in which the cursor is positioned.
Underline errors in the editor	 : Incorrect or unknown program code is underlined.

Highlight Symbols	 : All occurrences of a symbol where the cursor is positioned are highlighted in color within the editor. In this way, cross-references within the editor are quickly detected.
Max. Degree of Parallelism	Drop-down list for the number of parallel threads that can be used for precompile processing. IndraLogic detects the displayed number of threads from the number of CPU cores. This default number should be changed only in exceptional cases.

Also refer to

- [chapter 17.1 "Programming languages and editors" on page 223](#)
- [chapter 9.5 "Using input assistants" on page 107](#)
- [chapter 9.12.1 "Using the cross-reference list to find places of use" on page 129](#)

“Options” - “Syntax Highlighting” dialog

Symbol: 

Function: This dialog is used for configuring the color and font settings for the text elements of an editor (for example, operands and pragmas).

Call: Menu **Tools** ► **Options**, category **Syntax highlighting**

“Options” dialog - Text editor

Symbol



Function

this dialog contains settings for working in a text editor.

Call

Menu **Tools** ► **IndraLogic 2G** ► **Options**, Category **Text editor**

Tab 'Editing'

Number of undo steps	Maximum number of editing steps to which you can apply the command Edit ► Undo .
Structure	Defines the structuring of the code through indentations. If you select an indentation, you can expand or collapse the indentation section with the help of a plus or a minus sign respectively before the first line of the respective section. • Indentation: All lines that are indented in comparison with the preceding line are summarized by IndraLogic in an indentation unit. • Explicit: you explicitly mark the code section to be summarized in an indentation unit with comments: before the section there must be a comment containing 3 opening curly parentheses '{{{'; after the section there must be a comment containing 3 closing curly parentheses '}}}'. The comments can contain additional text. Example: <pre> 1 IF iVar1=1 2 ///comment {{{ 3 THEN 4 iVar2:=2; 5 ELSE iVar2:=10; 6 END_IF 7 ///}}}} 8 iVar1:=iVar1+1; </pre> <pre> 1 IF iVar1=1 2 ///comment {{{ [5 lines] 8 iVar1:=iVar1+1; </pre>
Word wrap	• Soft: The word wrap takes place at the edge of the editor window if 0 is entered for wrap point. • Hard: The word wrap takes place after the number of characters specified for the wrap point.
Tab size	Number of characters
Keep tabs	☒ : The empty space that you inserted with the <Tab> key is not resolved into spaces afterwards by IndraLogic.
Indentation size	If you have activated the option Automatically indentIntelligent or Intelligent with code completion , IndraLogic inserts the number of spaces at the beginning of the line.
Automatic indentation	• Do not automatically indent • Block: a new line automatically adopts the indentation of the preceding line. • Intelligent: Lines that follow a line containing a keyword (for example VAR) are automatically indented by the specified "indentation size". • Intelligent with code completion: Indentation as with the option Intelligent, but IndraLogic additionally inserts the concluding keyword (for example END_VAR).

Tab "Text area"

Highlight current line	The background of the current line is shown in color
Associated parentheses	If the cursor is positioned before or after a parenthesis within a code line, IndraLogic marks the associated closing or opening parenthesis with a frame in the set color.
End of line markers	IndraLogic marks the end of each editor line with a small dash in the set color behind the last character (including spaces) of the line.
Line break position:	If a soft or hard word wrap is activated, the defined word wrap point is indicated by a vertical line in the selected color.
Caret color	Color of the cursor character
Selection color	Color of the selected text area
Inactive	Color of a selection if the corresponding window is not active (focus is in another window).
Foreground, reduced line	Color of the header of a closed, indented section in the code
Background, reduced line	The header of a closed, indented section in the code is given a colored background.
Fonts	A click on the field opens the common dialog for the configuration of the font.

Tab "Margin area"

Settings for the left-hand border of the text editor window, which is separated from the input area by a vertical line:	
Number lines	Display of the line numbers in the declaration and implementation part, in each case starting with 1
Foreground color	Color of the line numbers
Background color	Border color of the border
Highlight current line	The line number of the current line is shown in color
Parenthesis	A parenthesis encompasses the lines between the keywords that open and close a construct, for example <code>IF</code> and <code>END_IF</code>. If this option is activated and the cursor is located before, after or in one of the keywords of a construct, the parenthesis area is indicated by a square parenthesis in the border. You can select the color of the Parenthesis from the selection list of this option.
Active border color	Color of the dividing line between the border and input area
Inactive border color	Color of the dividing line between the border and input area of the part of the window that is presently inactive
Mouse actions	You can assign one of the following actions to each of the specified mouse actions or mouse button combinations. IndraLogic executes the actions if you execute the mouse action on the plus or minus sign before the header of a parenthesized area: • Highlight structure: IndraLogic selects all lines of the parenthesized area. • Toggle structure: IndraLogic opens or closes the parenthesized area or, if there are nested parentheses, the first level of the parenthesized area. • Toggle structure completely: IndraLogic opens or closes all levels of a nested parenthesized area.

Tab "Monitoring"

Settings for the display of the monitoring fields	
Enable Inline monitoring	Display of the monitoring fields behind the variables in online mode
Monitoring foreground color	Display of the value in the monitoring field

Monitoring background color	Display of the background in the monitoring field
Sequence control, foreground color	Display of the value in the monitoring fields at the flow control positions
Sequence control, background color	Display of the background in the monitoring fields at the flow control positions
Number of displayed digits	Display of total length (1-20) in the monitoring field, consisting of integer places and decimal places. Example of variable value "1234.5678": • Display for 3 places (default): 1.23E+03 • Display for 5 places: 1234.6 • Display for 7 places: 1234.568
String length	Maximum length of string variable values in the monitoring field

Also refer to

- [chapter "Programming in Structured Text \(ST\)" on page 101](#)

“Options” dialog - Visualization

Function: General settings for working with visualizations in IndraLogic 2G are made in this dialog.

Call: Menu: **Tools ▶ Options ▶ IndraLogic 2G ▶ Visualization.**

General information

Display options (visualization editor in the development system)	
Define how the image file is to respond to changes in the element frame size.	
Fixed	The image keeps the original size even if the size of the element frame changes.
Isotropic	The image keeps its proportions; i.e. the height-width ratio remains even if the height and width of the element frame are modified separately.
Anisotropic	The image adjusts to the size of the element frame; i.e. height and width can be modified independently of each other.
Anti-aliased drawing	 Enable this option of the programming system “Antialiasing” (offline or online) when drawing visualizations in the visualization editor (anti-aliasing, reduction of unwanted effects resulting from the limited pixel grid), (image resolution, alias effect with pixel grid) or due to the staircase effect when generating a computer graphics (computer-generated graphics in 2D or 3D).

Editing options	
Linking with toggle variables and key variables	 Placeholder <Toggle variable/key variable> enabled in visualization element properties. When dragging an element provided with the property “color variables” and “color envelope”, IndraLogic configures this property with the placeholder <Toggle variable/key variable>. The following elements are affected: Button, frame, image, line, circle segment, polygon, rectangle, text field, scroll bar

Grid

Grid	
Visible	 : when this option is enabled, a grid based on the size defined below is shown in the visualization editor for orientation purposes.
Active	: If this option is enabled, visualization elements can only be positioned on the points in the grid "size" defined below, no matter if the grid is set as "visible" or only "active". When inserting or moving an element this means that the center of the element can only be positioned on one grid point. When changing an element, this means that the small rectangle on the element edge that is moved in order to change the size or form of the element can only be moved to one grid point.
Size	vertical and horizontal distance between the grid points in pixels.

File options

File options	
Text file for textual "List components"	In the input field located below, the name (including the path) of a file (of .csv format) can be entered. It can then be used as a source for the suggestions contained in the "List components" function during editing of the 'Texts/Text' field within the properties of a visualization element. This text file can come from the export of a global text list, for example (chapter "Import/export text lists" command" on page 686). It has a tabular structure which has to correspond to the structure of text lists which is described on the help page for text lists (chapter 18.1.23 "Text list" object" on page 553).
Visualization directories	
Text list files	Use the  button to open the dialog for defining an existing or a new directory for the language files that are to be used in the project for configuring text and language in the visualization. The files found in this directory are made available when a text list is to be imported, e. g. after it has been processed externally by a translator. Only one directory can be entered here. For text and language configuration, also refer to the description in chapter 18.1.23 "Text list" object" on page 553.
Image files	Use the  button to open the dialog for defining an existing or a new directory for the image files that are to be accessible in the project for visualizations. Enter several directories here in a list with the items separated by semicolons.

"Options" dialog - "Visualization style"

Function: The dialog is used to define the visualization style to be used in the visualizations in libraries and global visualizations ("General module folder") in projects.

Call: Menu **Tools** ► **Options** ► **IndraLogic 2G**, category **Visualization styles**



The current setting in the "Options" dialog is not applied in online mode, but those present in the visualization manager.

Style configuration for libraries and global visualizations	
Do not use any visualization style	If this option is selected, the visualizations in the libraries or in the "General module folder" respond as if no style was assigned.
Use following visualization style	If this option is selected, the visualizations in the libraries or in the "General module folder" respond as per the style defined. The selection list provides all currently installed styles.
Automatically derive visualization style	If this option is selected, the visualization in the libraries or in the "General module folder" try to use the style set in the visualization manager of the current project. If this is not possible, the style specified in the "workaround is no visualization style could be derived" is processed.



The style settings applicable to all visualizations of the current application have to be configured under "Visualization manager" - "Settings".

(chapter "Visualization manager" - "Settings" on page 996)

"Style selection"

Show all versions (for experts only)

- : The selection lists for styles include the selected style and all further styles installed in the visualization style repository (but only in the latest version). If a later version of the selected style is installed, it is also listed.
- : The drop-down lists for styles include all installed styles in all versions.

Also refer to

- chapter "Visualization manager" - "Settings" on page 996

Style for new visualization managers

This option allows to preselect the style to be used for new visualization applications.



Note that in case of a standard selection, the actual used style can differ. That depends on the functionality of the device used.

Also refer to

- chapter "Visualization manager" - "Settings" on page 996

18.3.10 "Trace configuration" dialog

"Advanced trace settings" dialog

Function: This dialog provides extended settings for recording data.

Call: Interface **Advanced** in the **Trace configuration** dialog, **Record settings** subdialog.

Prerequisite: The trace editor is open and active. The dialog "Trace configuration" is open and the top node of the trace record tree is selected so that the subdialog "Record settings" is available.



For the calculation of the values, you have to select a task in the "Trace Configuration" dialog.

The buffer size is defined as "number of samples". IndraLogic calculates the time intervals that corresponds to this number and displays the result in normal fonts on the right outside the table (for example, 1h1m1s1ms). The calculation is possible only with the help of the task configuration settings and when the task cycle time is known.		
Measurement in every nth cycle	Data recording in every nth task cycle Default setting: 1; then the application performs the data recording in each task cycle.	Scanning interval of data recording. Example: 100ms
Recommended size of the runtime buffer (samples)	Prerequisite: "Overwrite runtime buffer size" is disabled The maximum number of samples that IndraLogic calculates and recommends, which the application stores at runtime per trace variable. IndraLogic calculates the number in the task cycle time from the value in "Measure in every n-th cycle" and the value in Measure in every n-th cycle.	Maximum length of the time interval during which the application collects data on the runtime system. Example: 2s
Overwriting of recommended buffer size with	Maximum number of samples per trace variable that saves the application per trace variable in runtime mode. Example: 100 Value range: from 10  : The application uses the value entered here; not the value calculated by IndraLogic from Recommended size of the runtime buffer (samples) .	Maximum length of the time interval during which the application collects data on the runtime system. Example: 6s
Trace editor buffer size per variable (samples)	Number of values that can be stored per variable in the trace editor. Example: 10000	The maximum time period for the display in the trace editor results from the maximum number and the scanning interval of the data recording. You can scroll back a maximum of this time in the trace editor.

Also refer to

- [chapter 18.1.27 ""Trace" object" on page 568](#)
- [chapter 13.3.2 "Creating the trace configuration" on page 204](#)

'Trace configuration' dialog

Symbol: 

Function: The dialog contains the trace configuration for data recording.

Call

- IndraWorks menu **Trace ► Configuration**, context menu
- Link **Configuration** in the trace editor
- Link **Add variable** in trace editor

Prerequisite: The trace object editor is open and active.

Tree view 'Trace configuration'

The tree view lists the variables displayed in trace and allows for access to the variable settings.

Selected trace name	The subdialog "Record settings" is displayed on the right.
Selected trace variable	The subdialog "Variable settings" is displayed on the right.

Context menu commands

+ Add variable	Adds a new trace variable. The subdialog Variable settings opens on the right and it is partially configured. Select a variable in the input field of the variable setting to trace its value graph.
Assign to diagram	Lists the diagrams (in the submenu on the right) where the selected variable is not currently displayed. Select a diagram to display the variable there. The command is available when a variable is selected in the tree view. Tip: If the command is disabled, the variable is already displayed in all diagrams.
Activated	Enabled by default Disabled variables are grayed out. They are neither displayed nor recorded.

Tree view 'Presentation (Diagrams)'

The tree view lists the diagrams that are displayed in the trace editor and allows for access to their display mode.

Selected node time axis	The subdialog Display mode for the time axis is displayed on the right. Specify the time axis display. See below.
Selected diagram name	The settings for the coordinate system of the diagram and a preview are displayed on the right. See below.
Selected node Y-axis	The subdialog Display mode is displayed on the right. Specify the axis display. See below.
Show variables	
Selected trace variable	The subdialog Variable settings is displayed on the right. Configure the trace variable. Note: They are the same settings that can be accessed in the tree view Trace record.

Context menu commands

+ Add diagram	Adds a new diagram below and displays it in the tree view Presentation (Diagrams).
+ Add new variable	Adds a new trace variable. The subdialog Variable settings opens on the right and it is partially configured. Select a variable in the input field of the variable setting to trace its value graph. Specify its display. In addition, the variable is assigned to the selected diagram.
Add existing variable	Lists all trace variables (in the submenu on the right) where the selected diagram is not currently displayed. Select a variable in order to display it in the selected diagram. Tip: If the command is disabled, all trace variables are already displayed in the selected diagram.

Partial dialog "Recording settings"

Prerequisite: In the tree view trace record , the top node is selected.

Trigger enabled	 : Trigger resolution is activated. The trace data is buffered in runtime mode only when a trigger signal has been sent. Select how the trigger signal is sent in the settings Trigger variable , Trigger parameter , Trigger edge , Post-trigger in samples , and Trigger level .  : Continuous display of the current data sets
Trigger variable	Signal used as trigger. A complete instance path is required. A valid trigger signal is an IEC variable, a property, a reference, a pointer, an array element of the application, or an expression. Permitted types are all IEC-based types except <code>STRING</code> , <code>WSTRING</code> , and <code>ARRAY</code> . Enumerations are permitted when the basic type is not <code>STRING</code> , <code>WSTRING</code> or <code>ARRAY</code> . The content of a pointer is not a valid signal. When the runtime system uses the <code>CmpTraceMgr</code> component, a property that is linked to the 'monitoring' attribute can then be recorded as a variable.
Trigger parameter	System parameter used as trigger The Input Assistant dialog lists all valid system parameters in the Parameters category of the Categories tab.
	Selects the Trigger Variable or Trigger Parameter
Trigger edge	<i>Defined the edge detection for triggering:</i> • Positive – For Boolean trigger variables, triggering occurs when the values changes from <code>FALSE</code> to <code>TRUE</code>. – In case of an analog data type, triggering occurs as soon as the value defined in "Trigger level" is approximated from a lower value. • Negative – For Boolean trigger variables, triggering occurs when the values changes from <code>TRUE</code> to <code>FALSE</code>. – In case of an analog data type, triggering occurs as soon as the value defined in "Trigger level" is approximated from a higher value. • Both – For Boolean trigger variables, triggering occurs as soon as the values changes – For analog trigger variables, triggering occurs when the value as defined in Trigger level is reached.
Post trigger (samples)	Number of records per trace variable that are buffered after triggering. Default setting: 50. Value range: 0 to ($2^{32} - 1$)
Trigger level	Value causing the trigger when reached
Task	Task recording data

Record condition	In runtime mode, the application checks the recording condition. If the condition is fulfilled, the trace data is buffered. <i>Record condition for data recording with CmpTraceMgr runtime system component:</i> - As expressions that include only permitted operators and operands Permitted operators (can also be nested): (Logical) AND, NOT, OR, the relational operators <, <=, >, >=, =, <> Permitted operands: Variables valid for trace - As variables Permitted type: BOOL, bit access, property. The condition is met for TRUE or 1. The contents of a pointer are not permitted. <i>Record condition for a data recording with IEC code</i> - As an expression returning a Boolean value.
Comment	Comment, for example about the recording condition
Resolution	Measure for the time stamp that is recorded per data set ms: Time stamp in ms µs: Time stamp in µs microseconds for a task cycle time of 1 ms or less
Automatic restart	 : Persistently saves the trace configuration and the last contents of the RTS buffer to the target device.
Advanced	Opens the Advanced trace settings dialog

Also refer to

- [chapter "'Advanced trace settings' dialog" on page 736](#)

Subdialog 'Variable settings'

Requirement: A trace variable is selected in the tree view of Trace record or Display (Diagrams).

Variable	Valid variable; value recorded with full instance path <i>Valid:</i> - IEC variable - Property - Reference Book - Content of the pointer - Array element <i>Permitted data type</i> - IEC-based type except STRING, WSTRING, or ARRAY - Enumeration when the basic type is not STRING, WSTRING or ARRAY When the runtime system uses the CmpTraceMgr component, a property that is linked to the "monitoring" attribute can then be recorded as a variable.
Parameters	Parameters, whose data is recorded. Prerequisite: Runtime system with CmpTraceMgr component The Input Assistant dialog lists all valid system parameters in the Parameters category of the Categories tab.
	Enables toggling between variable and parameter
Color	Color of the variable in the trace diagram

Line type	<i>Display as line chart</i> Line: Values are linked to form a line Step: Values are linked to form stairs None: Values are not linked
Point type	<i>Display as scatter chart</i> Point: Value is displayed as point Cross: Value is displayed as a cross None: Value is not displayed
Enable lower limit	 : Warning when less than the lower limit
Lower limit	If the value of the trace variable falls below the limit, the variable is displayed in the warning color.
Color	Warning color when fallen below
Enable upper limit	 : Warning when exceeding the upper limit
Upper limit	If the trace variable value exceeds the upper limit, the variable is highlighted in a signal color.
Color	Warning color when exceeded

Subdialog 'Display mode'**Prerequisite:** In der tree view Display (Diagrams), an axis is selected.

Display mode	Scaling: "Auto": Automatically scaled time axis "Fixed Length": Time axis segment with constant length "Fixed": Time axis segment from minimum to maximum
Minimum	Start value of the time axis segment. Prerequisite: Display mode is fixed
Maximum	End value of the time axis segment. Prerequisite: Display mode is fixed.
Length	Constant segment length; the initial value is adjusted automatically.
Grid	 : Diagram with grid lines in the x-direction. Select the grid line color from the drop-down list of colors

Tick marks

Fix distance	 : Display of tick marks with distance and subdivisions
Distance	Distance of the tick marks
Divisions	Number of divisions between two tick marks
Fonts	Font used for the time axis
Link "Preview"	Displays the preview of the diagram

Diagram preview**Prerequisite:** In der tree view Display (Diagrams), a diagram is selected.

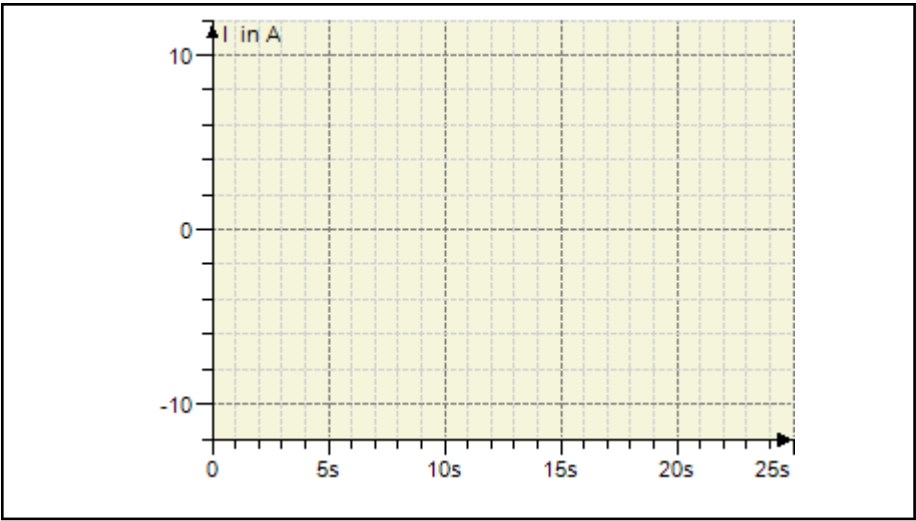


Fig. 18-76:

Background color	Background color of the diagram
Background color during selection	Background color of the selected diagram.

Link "Add variable"	Adds a new trace variable (in the tree view "Trace record").
Link "Remove variable"	Deletes the selected trace variable (in the tree view "Trace record").
Link "Add diagram"	Adds a new diagram (in the tree view "Display").
Link "Remove diagram"	Deletes the selected diagram (in the tree view "Display").
Link "Reset display settings"	Resets the display settings of either the selected diagram or y-axis to the default values.

Also refer to

- [chapter 13.3.2 "Creating the trace configuration" on page 204](#)
- [chapter 13.3 "Data recording with "Trace"" on page 200](#)
- [chapter "Attribute 'monitoring'" on page 394](#)

19 Libraries

Libraries are used to provide function blocks and functions for applications in IndraLogic. In addition to the descriptions in the online help, refer to the documentation in the library.

For information on the handling of libraries in the IndraLogic project, refer to: [chapter 16 "Creating and managing libraries" on page 219](#).

When creating libraries using IndraLogic, refer to the guidelines for the library developer!: [chapter 19.1 "Guidelines to create libraries" on page 743](#)

19.1 Guidelines to create libraries

Libraries have to be created according to specific rules to avoid compatibility problems.

Most important is:

- Define a reasonable library name (mandatory)
- Use templates to ensure consistency (optional)
- Use a known and unique project structure (if possible) (optional)
- Register a unique, global library namespace (mandatory)
- Complete the library info (mandatory)
- Reference the other libraries correctly (mandatory)
- Implement usable external and internal interfaces (mandatory)
- Implement a user-friendly error management (mandatory)
- Use suitable methods (protection) for the provision (mandatory)
- Follow the naming convention for a proper code (optional)
- In case of a further library development, ensure the interface compatibility to the previous versions.

Also refer to

- [chapter 16 "Creating and managing libraries" on page 219](#)

19.2 "Recipe management" library

If a recipe manager is added to an application, the `Recipe management` library is transferred to the library manager of the application.

The `RecipeManCommands` function block is included in this library. The function block manages the recipe management methods described in the following.

The return values of the methods are included in the Global Variable List `ReturnValues`.



When calling the recipe commands, IndraLogic accesses files internally. Depending on the device type, it can take some milliseconds.

Note that these calls are not executed from a task for which a watchdog or a real-time sensitive task was configured.

Furthermore, the **Save recipe changes files automatically** option also accesses files when changes are made to the recipes (recipe manager, tab "General").

Disable this option to explicitly control the saving via the application.

"CreateRecipe" method

This method generates a new recipe in the given recipe definition. It loads the current PLC values to the new recipe and saves them as recipe file under the default name.

The default name is <Recipe>.<RecipeDefinition>.<RecipeExtension>.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-1: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
CreateRecipe	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_ALREADY_EXIST - ERR_RECIPE_NOMEMORY - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-2: Return value

"CreateRecipeNoSave" method

This method generates a new recipe in the given recipe definition. It loads the current values to the new recipe.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-3: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
CreateRecipeNo-Save	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_ALREADY_EXIST - ERR_RECIPE_NOMEMORY - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-4: Return value

“DeleteRecipe” method This method deletes a recipe from the recipe definition.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-5: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
DeleteRecipe	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-6: Return value

“DeleteRecipeFile” method This method deletes the specified recipe file of a recipe. The recipe file has to be saved in the default name <Recipe>.<RecipeDefinition>.<RecipeExtension>.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-7: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
DeleteRecipeFile	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_RECIPE_FILE_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-8: Return value

“LoadAndWriteRecipe” method This method loads a recipe from the specified recipe definition. The recipe file has to be saved in the default name <Recipe>.<RecipeDefinition>.<RecipeExtension>. It loads the recipe to the PLC variables.



Entries in the recipe file without value assignment are neither loaded nor written ([chapter ""Load and write recipe" command" on page 682](#)).

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-9: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
LoadAndWriteRecipe	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_RECIPE_FILE_NOT_FOUND - ERR_RECIPE_MISMATCH - ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-10: Return value

Also refer to

- [chapter ""Load and write recipe" command" on page 682](#)

"LoadFromAndWriteRecipe" method

This method loads the specified recipe file to a recipe. It loads the recipe to the PLC variables.



Entries in the recipe file without value assignment are neither loaded nor written ([chapter ""Load and write recipe" command" on page 682](#)).

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe
FileName	STRING (255)		File name

Tab. 19-11: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
LoadFromAndWriteRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_RECIPE_FILE_NOT_FOUND • ERR_RECIPE_MISMATCH • ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-12: Return value

"LoadRecipe" method

This method loads a recipe from a recipe file. The recipe file has to be saved in the default name <Recipe>.<RecipeDefinition>.<RecipeExtension>.



Entries in the recipe file without value assignment are neither loaded nor written ([chapter "Load and write recipe" command](#) on [page 682](#)).

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-13: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
LoadRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_RECIPE_FILE_NOT_FOUND • ERR_RECIPE_MISMATCH • ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-14: Return value

"ReadAndSaveRecipe" method

This method loads the current PLC values to a recipe. It saves the recipe under the default name to a recipe file. The default name is <Recipe>.<RecipeDefinition>.<RecipeExtension>. The content of a possibly existing file is overwritten.

Libraries

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-15: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
ReadAndSaveRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_RECIPE_SAVE_ERR • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-16: Return value

“ReadAndSaveRecipeAs” method

This method loads the current PLC values to a recipe. It saves the recipe to a specified recipe file. The content of a possibly existing file is overwritten.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe
FileName	STRING (255)		File name

Tab. 19-17: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
ReadAndSaveRecipeAs	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_RECIPE_SAVE_ERR • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-18: Return value

“SaveRecipe” method

This method saves the recipe to a recipe file under the default name. The default value is
 <Recipe>.<RecipeDefinition>.<RecipeExtension>.
 The content of a possibly existing file is overwritten.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-19: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
SaveRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_RECIPE_SAVE_ERR • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-20: Return value

“ReadRecipe” method This method loads the current PLC values to a recipe.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-21: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
ReadRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-22: Return value

“WriteRecipe” method This method writes the recipe to the PLC variables.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Tab. 19-23: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
WriteRecipe	DWORD		<i>Return value, possible values</i> • ERR_RECIPE_DEFINITION_NOT_FOUND • ERR_RECIPE_NOT_FOUND • ERR_NO_RECIPE_MANAGER_SET • ERR_OK

Tab. 19-24: Return value

“ReloadRecipes” method This method loads the list of recipes from the file system. This refers to recipes with the default name <Recipe>.<RecipeDefinition>.<Recipe-Extension> located in the path defined in the recipe manager.

Libraries

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition

Tab. 19-25: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
ReloadRecipes	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-26: Return value

“GetRecipeCount” method This method returns the number of recipes of a recipe definition.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition

Tab. 19-27: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
GetRecipeCount	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-28: Return value

“GetRecipeNames” method This method returns the recipe names in a recipe definition.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
strings	POINTER TO ARRAY[] OF STRING		Pointer to the character strings saving the recipe names.
iSize	INT		Size of the character string array.
iStartIndex	INT		Start index

Tab. 19-29: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
GetRecipeNames	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-30: Return value

Example:

There are for example 50 recipes. To generate a table that displays 10 recipe names at the same time, define a STRING array:

```
strArr: ARRAY[0..9] OF STRING;
```

The recipe names for the specified range corresponding to `iStartIndex` are output.

```
iStartIndex := 0; // The names 0..9 are returned
iStartIndex := 20; // The names 20..29 are returned
```

The following applies in this example:

```
iSize := 10;
```

"GetRecipeValues" method

This method returns the values of a recipe.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName			Name of the recipe
strings	POINTER TO ARRAY[] OF STRINGS		Pointer to the character strings saving the recipe values
iSize	INT		Size of the character string array
iStartIndex	INT		Start index
iStringLength	INT		Lengths of the character string in the array

Tab. 19-31: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
GetRecipeValues	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-32: Return value

Example:

There are for example 50 recipes. To generate a table that displays 10 recipe values at the same time, define a `STRING` array:

```
strArr: ARRAY[0..9] OF STRING;
```

The recipe values for a specified range corresponding to `iStartIndex` are output.

```
iStartIndex := 0; // The values 0..9 are returned
iStartIndex := 20; // The values 20..29 are returned
```

The following applies in this example:

```
iStringLength := 80;
iSize := 10;
```

"GetRecipeVariableNames" method

This method returns the names of the recipe variables of a recipe.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName			Name of the recipe
strings	POINTER TO ARRAY[] OF STRINGS		Pointer to the STRING data types saving the names of recipe variables.
iSize	INT		Size of the STRING array.
iStartIndex	INT		Start index

Tab. 19-33: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
GetRecipeVariableNames	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-34: Return value

Example:

There are for example 50 recipes. To generate a table that displays 10 variable names of a recipe at the same time, define a STRING array:

```
strArr: ARRAY[0..9] OF STRING;
```

The variable names of the recipe for the specified range corresponding to `iStartIndex` are output.

```
iStartIndex := 0; // The variable names 0..9 are returned
iStartIndex := 20; // The variable names 20..29 are returned
```

The following applies in this example:

```
iSize := 10;
```

"SetRecipeValues" method

This method returns the recipe values of a recipe.

Name	Data type	Initial value	Description
RecipeDefinitionName	STRING		Name of the recipe definition
RecipeName	STRING		Name of the recipe

Name	Data type	Initial value	Description
strings	POINTER TO ARRAY[] OF STRINGS		Pointer to the STRING data types saving the recipe values.
iSize	INT		Size of the STRING array.
iStartIndex	INT		Start index

Tab. 19-35: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
SetRecipeValues	DWORD		<i>Return value, possible values</i> - ERR_RECIPE_DEFINITION_NOT_FOUND - ERR_RECIPE_NOT_FOUND - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-36: Return value

Example:

There are for example 50 recipes. To generate a table that sets 10 recipe values at the same time, define a STRING array:

```
strArr: ARRAY[0..9] OF STRING;
```

Define the recipe values for a specified range corresponding to iStartIndex.

```
iStartIndex := 0; // The values 0..9 are set
iStartIndex := 20; // The values 20..29 are set
```

The following applies in this example:

```
iStringLength := 80;
iSize := 10;
```

“GetLastError” method

This method returns the last error of the preceding operation.

Name	Data type	Initial value	Description
GetLastError	DWORD		<i>Return value, possible values</i> - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-37: Return value

“ResetLastError” method

This method returns the last error.

Name	Data type	Initial value	Description
ResetLastError	DWORD		<i>Return value, possible values</i> - ERR_NO_RECIPE_MANAGER_SET - ERR_OK

Tab. 19-38: Return value

"SetStoragePath" method This method sets the storage path for the recipe file. It overwrites the path specification in the **Storage** dialog of the recipe manager.

Name	Data type	Initial value	Description
stPath	STRING		File path, example: D:/recipefiles/

Tab. 19-39: Inputs (VAR_INPUT)

Name	Data type	Initial value	Description
SetStoragePath	BOOL		Return value TRUE (path was set), FALSE

Tab. 19-40: Return value

Return values The return values of the above described methods are included in the Global Variable List `ReturnValues`.

These are InOut constants of the data type UDINT.

Name	Initial value	Comment
ERR_OK	16#0	Operation was completed successfully.
ERR_FAILED	16#1	Operation failed.
ERR_PARAMETER	16#2	Incorrect parameter
ERR_NOTINITIALIZED	16#3	The DataServer object is not initialized. The DataServer is required when using the recipe manager together with the VisuDeviceHMI.
ERR_NOTIMPLEMENTED	16#C	The DataServer does not implement the interface <code>IDataServer4</code> required to use the recipe manager with the VisuDeviceHMI.
ERR_NO_OBJECT	16#10	The DataServer cannot write all variables of a recipe definition. Only the valid variables are written.
ERR_NOMEMORY	16#11	The DataServer is provided with insufficient memory.
ERR_RECIPE_FILE_NOT_FOUND	16#4000	The recipe file was not found.
ERR_RECIPE_MISMATCH	16#4001	The recipe file content do not match the current recipe. This error is only output if the storage type is "Textual" (refer to the dialog Recipe manager , tab Storage , Storage type) and if a variable name in the file conflicts with the variable name in the recipe definition. The recipe file is not loaded when this error occurs... Possible causes: A variable was removed from the recipe definition in the project.
ERR_RECIPE_SAVE_ERR	16#4002	Saving failed. Possible causes: The file could not be created or opened, as the disk is full. The configured file path does not exist (refer to the dialog Recipe manager , tab Storage , File path). The configured file extension is not permitted by the runtime system (refer to the dialog Recipe manager , tab Storage , File extension)
ERR_RECIPE_NOT_FOUND	16#4003	The recipe does not exist.

Name	Initial value	Comment
ERR_RECIPE_DEFINITION_NOT_FOUND	16#4004	The recipe definition does not exist.
ERR_RECIPE_ALREADY_EXIST	16#4005	The recipe already exists in the recipe definition. Create a new recipe with another name.
ERR_NO_RECIPE_MANAGER_SET	16#4006	The global recipe manager has not been created. Possible causes: The option Recipe management in the PLC of the current application is not set (in the dialog Recipe manager , tab General).
ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED	16#4007	The recipe definition includes more variables than the recipe file. In this case, the variable values of the recipe file are definitely written. This is just an information, not an actual error.
ERR_RECIPE_NOMEMORY	16#4008	The recipe definition does not have any free memory to create a new recipe. Possible causes: The option Save recipe changes automatically to recipe files of the current application is not set (in the dialog Recipe manager , tab General). In this case, only 50 recipes are possible per recipe definition. If the option Save recipe changes to recipe files automatically is set, the error cannot occur. If the disk is full, the error ERR_RECIPE_SAVE_ERR is output.
ERR_RECIPE_MANAGER_LOCKED_DURING_ONLINE_CHANGE	16#4009	The recipe manager was blocked during the online change. Possible causes: Some of the "RecipeMan" commands to be executed during an online change, were not executed.
ERR_SOURCE_EXHAUSTED	16#40A0	Used for UTF8 helper
ERR_TARGET_EXHAUSTED	16#40A1	Used for UTF8 helper
ERR_SOURCE_ILLEGAL	16#40A2	Used for UTF8 helper

20 IndraLogic 2G visualization

20.1 Visualization overview

Depending on the application, IndraWorks provides two visualization tools implemented with different levels of performance:

- WinStudio: This tool should only be used for user visualizations running on visualization devices (BTV/Vxx). The "Symbol configuration" object is used to transfer the data to be visualized ([chapter 10.2 "Symbol configuration" on page 156](#)).
- Visualization onboard: This tool described below should be used to support the commissioning of own applications.

Tools and editors

An IndraLogic 2G visualization is based on a modular visualization concept and provides the following tools/editors.

- [chapter 20.3 "Editor" on page 777](#): A user-friendly way to create visualizations/masks/panels in an IEC 61131-3 tool with a few clicks.
- [chapter 20.4 "Visualization elements" on page 795](#): Many ready-to-use elements are available.
- [chapter 20.9 "Visualization profile" on page 1016](#): Instead of the suggested standard profile, a visualization profile can be selected from a visualization library. Each visualization project containing at least one visualization has to be based on such a profile.
- [chapter 20.7 "Visualization libraries" on page 1015](#): Elements can be combined to provide them to other projects.
- The element display is defined using a visualization style to achieve a certain look and feel of the elements.
- [chapter 20.5 "Visualization manager with clients" on page 996](#): Here, different display variants are managed. The variants can easily be adjusted to different customer-specific applications.

IndraLogic 2G visualizations facilitate:

- Nesting of visualizations by references and/or switching between different visualizations: [chapter "Element 'Frame'" on page 866](#)
- Using interfaces for parameter transfer to instantiate complex visualizations: [chapter "Basic information on the interface editor" on page 782](#)
- Supporting multilingualism by text list editors: [chapter "Using text and language " on page 796](#)

General mechanism

[chapter "Diagnostic visualization" on page 999](#): This display variant only runs in the programming system (corresponds to the online visualization in IndraLogic 1x).

Data of external (remote) data servers can be used in visualizations. For this purpose, this data has to be applied and processed in the local application.

Also refer to

- [chapter 10.4.1 "Basics of the data source manager" on page 162](#)

Generating a visualization

A visualization object can be created in the Project Explorer below an application or in the "General Module Folder" by using the **Add...** command (also in context menu).

The window of a new visualization object shows a blank area at first.

Once the first visualization object is added to the project, all corresponding visualization libraries are automatically added by the library manager according to the currently active visualization profile.

A visualization library is always set up as special type of a **placeholder library**. As a consequence, the version of the used library is not resolved until the library is added to the project. Only then, the currently active profile specifies which version is actually needed.

Please note that this library type differs from the device-specific placeholder libraries. Here, placeholders are resolved in the device description.

Basic libraries inserted by default if a visualization object is inserted are: VisuElems, VisuElemMeter, VisuElemWinControls, VisuElemTrace, VisuInputs.

The user can insert more libraries.



Generally, **no** visualization library has to be added manually or has to be explicitly used in the project.

A visualization can be created as object in the "General Module Folder" or directly in an application.

As soon as the first visualization is added to an application, a visualization manager is also added ([chapter 20.5 "Visualization manager with clients" on page 996](#)).

A start visualization, i.e. a visualization that is opened first after a PLC login has to be inserted below the corresponding application object in the Project Navigator.

Each visualization has the properties as per its intended use:

Visualization, keypad/keyboard, dialog or display size ([chapter 20.6 "Dialog Properties" - "Visualization object" on page 1001](#)).

In this context, ensure that a visualization can be created and configured explicitly for use as user input dialog in other visualizations ([chapter "Input configuration" on page 800](#)).

A keypad or a keyboard are implicitly available by default. The use of a keyboard and dialog can be defined in the **Input configuration** properties of a visualization element.

If an error or an exception occurs in a visualization, the visualization is stopped without stopping the application.

An error message is displayed. The visualization can be started again via this error screen.

This exception is handled from visualization profile 3.5.7.0, compiler version 3.5.7.0 and a runtime system from version 3.5.7.0.

Select the command **Stop execution at handled exceptions** to analyze the reason for exceptions and to find the source of the error.

Also refer to

- [chapter ""Stop execution on handled exceptions" command" on page 700](#)

Online mode

The visualization works with an integrated runtime system. Thus, start and download messages are displayed when working with the editor.

Visualization, dialog, virtual keypad or virtual keyboard?

20.1.1 Displaying a preview of the library visualization

A library visualization is a function block in a library referencing a visualization.

IndraLogic uses the library manager to provide a description of each block. If the function block is a library visualization, a preview of the visualization is scaled isotropically and displayed on the **Graphical** tab.

Previewing a library visualization

1. Open the library manager.
2. Select a library containing a library visualization, for example `VisuDialogs`

The contents of the library are displayed below the tree view.

3. Select a library visualization from the tree view, for example `Keypad`.
The selected block is described to the right of the library tree view.

4. Click the **Graphical** tab.

The preview of the library visualization is displayed.

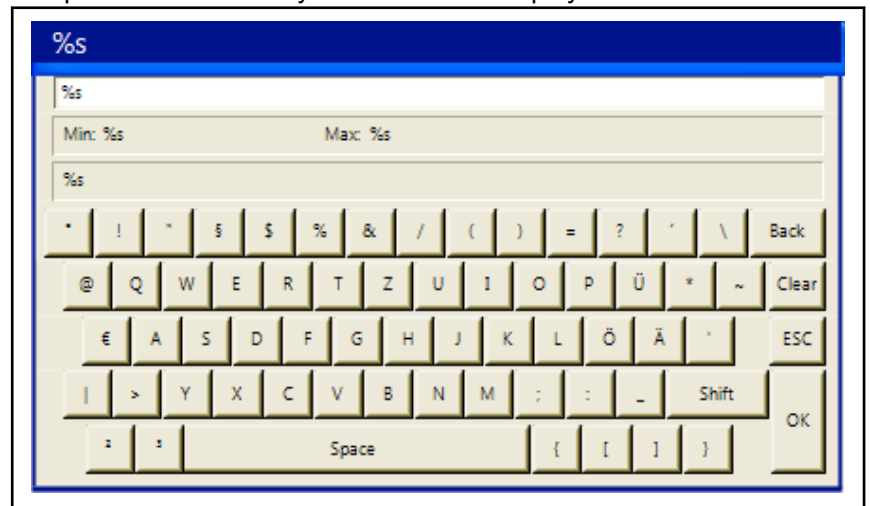


Fig. 20-1:

20.1.2 Displaying and labeling visualization elements

The view **Tools** provides all visualization elements available in the project. This comprises visualization profile elements, frame elements of library and display elements of symbol libraries. To identify and evaluate an element, IndraLogic provides a preview in the **Tools** view. Select an element and drag&drop it to the visualization editor. Furthermore, elements can be label and sorted via these labels.

Available elements

Origin of the elements

- current visualization profile
- images of image pools contained in the project or the library
- visualizations contained in the project or the library

An image available in the table is saved as reference on an image file in an image pool. However, IndraLogic ignores an image pool, when the option **internally** is enabled in its properties. An image can also be saved in a symbol library that can be accessed by the user.

An existing visualization that is displayed as element is saved in the project or in a library that can be accessed by the user. A library that cannot be accessed by the user is ignored and hidden by IndraLogic. When dragging and dropping an existing visualization to the visualization, it has the element type **Frame**.

Also refer to

- [chapter 9.9 "Using image pools" on page 121](#)
- [chapter ""Properties" - "Image Pool" dialog" on page 712](#)
- [chapter "Element 'Image'" on page 856](#)
- [chapter "Element 'Frame'" on page 866](#)

Identification

Each element is assigned to one or several identifications. The number of visible elements can be controlled using the identifications. IndraLogic provides each identification as interface in the **Tools** view. The button colors are either green or gray, depending on whether the identification is "selected" or "not selected". Only if an ID is selected and is green, IndraLogic displays the elements that are assigned to the ID in the table. By clicking on the button, the ID status can be changed.

IndraLogic automatically assigns IDs. However, IDs can also be created by the user and one or several elements can be assigned to the ID.

Use  for individual selection and  for multiple selection. Click on the ID button and keep the <Shift key> pressed, the selection mechanism is changed for one click.

IDs provided by IndraLogic

Each element of the visualization profile has an ID that categorizes.

Categorizing IDs

- **Basis**
- **General Control elements**
- **Measuring devices**
- **Lamps/switches/images**
- **Special Control elements**

An image of an image pool is assigned the image pool name as ID.

A library visualization is assigned the library name as ID.

Dragging elements to the visualization

*Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view is opened. The **Tools** view provides elements.
2. Drag and drop an element to the visualization.
The element is displayed in the visualization and has properties that can be configured by the user.

Generating IDs

*Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view opens.
2. In the **Tools** view, click on .
The **Configure IDs and entries** dialog is displayed.

3. In the dialog, click on the .
The dialog **Add ID** opens.
4. Enter a name in the **Name** input field, e.g. `tagA` and exit the dialog with **OK**.
The new user-defined ID `tagA` is inserted in **Configure IDs and entries** in the tree view below. It is assigned the symbol .
5. For the new ID, click on the **Activate** option and close the dialog with **OK**.
IndraLogic adds the button **tagA** to the **Tools** view.

Assigning an ID to an element

*Prerequisite: A project containing an object of type **Visualization** is opened. In the **Tools** view, the user-defined ID `tagA` is created.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view is opened.
2. Click on the **Tools** view on an element and activate the "Tools" context menu.
The context menu of the element contains command to assign an ID to an element.
3. Select the command **Add input to 'tagA' ID**.
4. Click on **tagA**.
All elements assigned to this ID are displayed. The assigned element is also displayed.

Removing the ID

*Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view opens.
2. In the **Tools** view, click on .
The **Configure IDs and entries** dialog is displayed.
3. Select a user-defined input in the tree view.
4. Click on  or enter `<DEL>`.
The ID is removed from the tree view.
5. Exit the dialog with **OK**.
IndraLogic removes the button in the **Tools** view.

Hiding the ID

*Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view is opened.
2. Focus one of the buttons in the visualization editor and enable their **context menu**.

The command **Hide ID** is displayed.

3. Click on the **Hide ID** command.
The ID is removed.

Show ID *Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view opens.
2. In the **Tools** view, click on .
The **Configure IDs and entries** dialog is displayed.
3. Under **Active**, click on .
Result: 
4. Exit the dialog with **OK**
The corresponding ID is visible as button.

Select ID in single selection *Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view opens.
2. In the **Tools** view, click on .
The buttons are selected individually.
3. Click on a gray button.
The button changes to green, all other buttons are grayed out.
In the table, the entries assigned to the green button are displayed.
4. Press <Shift> while clicking on a gray button.
The button is already green, in addition to the already selected button.
In the table, the assigned elements of the two green buttons are listed.
The buttons are still in single selection.

Selecting the ID in multiple selection *Prerequisite: A project containing an object of type **Visualization** is open.*

1. Double-click on the visualization.
The visualization editor and the **Tools** view opens.
2. In the **Tools** view, click on .
The button is in multiple selection.
3. Click on a gray button.
The button is green, all other already selected buttons remain green. In the table, the assigned inputs of all green buttons are listed.
4. Press <Shift> while clicking on a gray button.

The button changes to green, all other buttons are grayed out. In the table, the assigned elements of a green buttons are listed. The button field remains in multiple selection.

Also refer to

- [chapter 20.3.1 "Tools" view" on page 778](#)
- [chapter "Tools" command" on page 590](#)

20.1.3 Using the symbol library in the visualization

A symbol library provides images of image pools that can be referenced in the project and the application. When opening a visualization, IndraLogic provides the content of an installed and selected symbol library in the **Tools** view. Moreover, the user can easily access the images via the preview images.

A symbol library contains one or several objects of type **Image pool** and can contain objects of type **Text list**. A text list is assigned to exactly one image pool and contains the localized texts of the image IDs. A symbol library has the property `VisuSymbolLibrary = True`, IndraLogic reads out to identify a symbol library.

A symbol library can be provided as package.

Selecting the symbol library in the project for the first time

Prerequisite: A project not containing any visualization and no visualization manager is opened.

1. Select the application in the Project Explorer and select **Add** in the context menu.
2. Select **Visualization**.

The **Add visualization** dialog is displayed and lists the installed symbol libraries.

3. Activate the option **Active**, e.g. for `VisuSymbols` if you want to select one of the installed symbol libraries. Exit the dialog with **Add**.

The symbol libraries are selected in the project.

The libraries are in the "General module folder".

The **Tools** view contains the library symbols under **Symbols**.

4. Select the **General Module Folder ► Project settings**, category **Visualization, Symbol libraries** tab.

IndraLogic lists the installed symbol libraries and provides information whether they are activated for the project.

Disabling the symbol library in the project

Prerequisite: A project assigned to a symbol library is open.

1. Select the **General Module Folder ► Project settings**, category **Visualization, Symbol libraries** tab.

IndraLogic lists the installed symbol libraries and provides information whether they are selected for the project. The selected libraries have an activated option **Active**.

2. Activate the option in the symbol library you want to select.
3. Deactivate the option in the symbol library you want to deactivate.
4. Close the dialog with **OK**.

When opening a visualization, IndraLogic provides the content of the selected symbol library in the **Tools** view.

Updating the symbol library in the project

Prerequisite: A project assigned to a symbol library is open.

1. Select the **General Module Folder ► Project settings**, category **Visualization, Symbol libraries** tab.
IndraLogic displays the symbol libraries that are not up-to-date.
2. Select a newer version and exist the dialog with **OK**.
IndraLogic updates the symbol library.

Creating the symbol library

1. Select **File ► New project**.
The **New project** dialog is displayed.
2. Under **Categories**, select **Libraries**.
3. Select **Empty library**.
4. Enter a name, for example `SymbolLib_A`.
5. Generate an image pool, such as `ImagePool_A`.
6. Generate a text list that can be assigned to an image pool, e.g. `Text-List_A`.
7. Select the image pool object in the Project Explorer.
8. In the context menu, select **Properties**.
The **Properties - ImagePool_A** dialog is displayed.
9. Select the **Image pool** tab.
10. Activate the **Download only used images to control** option.
11. Click on the **Label library as symbol library** button.
The library is a system library. When loading a visualization, IndraLogic only downloads images to the control that are actually referenced in the visualization.
12. From the drop-down list of **Text lists for symbol compilation**, select the text list localizing IDs of the image pool, e.g. `ImagePool_A`.
13. Close the dialog with **OK**.
The library project is labeled as symbol library and contains an image pool with assigned text list.
14. Select **Tools ► Libraries ► Generate and install compiled libraries**.
The library project is saved and installed as symbol library.

Also refer to

- [chapter 16 "Creating and managing libraries" on page 219](#)

Generating image pool for symbol library

An image in the image pool of a symbol library has to be embedded in the project and has to **embed** the link type.

Prerequisite: A library project is open for a symbol library.

1. Select the **General Module Folder ► Add ► Image pool**.
2. Enter a name, such as `ImagePool_A`.
3. Double-click on the image pool.

The respective editor opens.

4. Double-click in a field under **File name** and click on .
The dialog **Select Image** is displayed.
5. In the **Image file** input field enter the file name of an image together with the path, e.g. D:\Images\Image_A.bmp.
6. Activate the **Embed in project** object and exit the dialog with **OK**.
The image is entered in the image pool list with the link type **embedded**.
7. Enter an ID under **ID**, e.g. Image_A.
An image is assigned to the image pool with the required properties.
8. Enter more images as described in the list.
The image pool has been created.

Creating a text list for the image pool

A text list, assigned to an image pool has to contains IDs that are contained in the image pool.

Prerequisite: A project with symbol library is open.

1. Select **Project ► Add ► Text list**.
2. Enter a name, for example TextList_A.
3. Select **Text list ► Add language** and enter the name of the language for the new column.
4. Enter an ID of the image pool in the text list under **ID**, e.g. Image_A.
5. Enter the translation in the language column, e.g. Bild_A.
The ID Image_A is localized.
6. If possible, localize all IDs of the image pool.
The text list has been generated.

Also refer to

- [chapter 20.1.2 "Displaying and labeling visualization elements" on page 759](#)
- [chapter 20.3.1 "'Tools' view" on page 778](#)
- [chapter "'Tools' command" on page 590](#)

20.2 Commands

The visualization commands are provided for the menu category **Visualization commands**, **Tools ► Adjust**. The commands facilitate editing of a visualization object in the visualization editor.

Most objects are part of the **Visualization** menu by default and thus also available in the context menu of the visualization editor. If requires, open the dialog **Tools ► Adjust** to view or change the menu configuration of the category **Visualization commands**.

20.2.1 “Add visual element” command

For this command (category **Visualization commands**), go to the **VI-Logic visualization** menu or to the context menu.

It is used to insert an element into the current visualization. It is analog to drag&drop in the **Tools** view ([chapter 20.3.1 “Tools” view on page 778](#)).

The command opens a submenu in which the desired element can be selected from the currently available elements. The element is then inserted into the upper left corner of the editor window.

20.2.2 Command “Interface editor”

Symbol: 

Default shortcuts: <Alt> + <F6>

This command (category **Visualization commands**) opens the interface editor to define frame parameters in the visualization used to be referenced in the “Frame” element of another visualization.

Also refer to:

- [chapter "Basic information on the interface editor" on page 782](#)
- [chapter "Element 'Frame'" on page 866](#)

20.2.3 Command “Keyboard usage”

This command (category **visualization command**) opens the “Keyboard usage” dialog for the current visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)).

The command is displayed in a tab view in the upper part of the visualization editor.

20.2.4 Command “Element list”

Symbol: 

This command (category **visualization commands**) opens the element list for the current visualization.

It is displayed in a tab view in the upper part of the visualization editor.

Also refer to:

- [chapter 20.3.5 "Tab “Element list”" on page 793](#)

20.2.5 Command “Activate keyboard usage”

Symbol: 

This command (category **Visualization commands**) is in the provided in menu bar for an integrated visualization. It activates or deactivates the keyboard usage in the online mode of a visualization.

When the keyboard usage is activated, input to elements and element selection can be performed using specific shortcuts. In this case, other commands given using the keyboard are not executed as long as the visualization editor is active and in online mode.

Also refer to:

- [chapter "Diagnostic visualization" on page 999](#)
- [chapter 20.11 "Keyboard operation in online mode" on page 1018](#)

20.2.6 Alignment

Command “Align left”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at the left edge of the most left element.

Command “Align top”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at the upper edge of the most top.

Command “Align right”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at the right edge of the most right element.

Command “Align bottom”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at the lower edge of the most bottom element.

Command “Center vertically”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at their common vertical center.

Command “Center horizontally”

Symbol: 

With this command (category **Visualization commands**), all selected visualization elements are aligned at their common horizontal center.

Command “Equalize horizontal distance”

Symbol: 

The command is active if three or more elements are selected.

Procedure

1. Select all elements to be positioned horizontally with the same spacing.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Equalize horizontal distance**.
The element are positioned so that the element to the far left and the element to the far right retain their position and the elements in between are aligned horizontally with the same distance.

Command “Increase horizontal distance”

Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned horizontally with more spacing.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Increase horizontal distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned horizontally with more distance between the elements. The distance is increased by 1 pixel.

Command “Decrease horizontal spacing”Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned horizontally with less spacing.
2. Execute the command **Decrease horizontal distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned horizontally with less distance between the elements. The distance is decreased by 1 pixel. The spacing reduces by one pixel.

Command “Remove horizontal distance”Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be horizontally positioned without any distance to each other.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Remove horizontal distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned horizontally without any distance between the elements.

Command “Equalize vertical distance”Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned vertically with the same spacing.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Equalize vertical distance**.

The elements are positioned so that the topmost element and the element at the bottom retain their position and the elements in between are aligned vertically with the same distance.

Command “Increase vertical distance”

Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned vertically with more spacing.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Increase vertical distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned vertically with more distance between the elements. The distance is increased by 1 pixel.

Command “Decrease vertical distance”

Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned vertically with less spacing.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Decrease vertical distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned vertically with less distance between the elements. The spacing reduces by one pixel.

Command “Remove vertical distance”

Symbol: 

The command is active if two or more elements are selected.

Procedure

1. Select all elements to be positioned to each other vertically without any distance.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Remove vertical distance**.
The elements are positioned so that the blue element retains its position and all other elements are aligned vertically without any distance between the elements.

Command “Equalize width”

Symbol: 

The command is active if more than one element except for a line or polygon element is selected.

Procedure

1. Select all elements that should be assigned the same width.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Equalize width**.
All elements are assigned the width of the element highlighted in blue.

Command “Equalize height”

Symbol: 

The command is active if more than one element except for a line or polygon element is selected.

Procedure

1. Select all elements that should be assigned the same height.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Equalize height**.
All elements are assigned the height of the element highlighted in blue.

Command “Equalize size”

Symbol: 

The command is active if more than one element except for a line or polygon element is selected.

Procedure

1. Select all elements to be aligned to the grid with regard to position and size.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Equalize size**.
Align all elements to the grid according to size and position.

Command “Adjust size to grid”

Symbol: 

The command is active if more than one element except for a line or polygon element is selected.

Procedure

1. Select all elements to be aligned to the grid with regard to position and size.
The first element is highlighted in blue. The other elements are highlighted in grey.
2. Execute the command **Size to grid**.
Align all elements to the grid according to size and position.

20.2.7 Order

“On level up” command

Symbol: 

Use this command (command **Visualization commands**) to position the selected element one level higher in the visualization, i.e. closer to the foreground of the visualization. Elements on lower levels are covered by those on higher levels.

Command “Place on top”

Symbol: 

Use this command (command **Visualization commands**) to position the selected element in the foreground of the visualization, i.e. on the top level.

Elements on lower levels are covered by those on higher levels.

Command “One level down”

Symbol: 

Use this command (command **Visualization commands**) to position the selected element one level lower in the visualization, i.e. closer to the background of the visualization. Elements on lower levels are covered by those on higher levels.

Command “Send to back”

Symbol: 

Use this command (command **Visualization commands**) to position the selected element in the background of the visualization, i.e. on the lowest level.

Elements on lower levels are covered by those on higher levels.

20.2.8 Command “Group”

Symbol: 

This command (category **Visualization commands**) is used to group the currently selected visualization elements and to display the group as a single selected object.

To select multiple items, press and hold the <<Shift>> key while clicking on the desired elements.

Alternatively, click in the editor window outside of an element and hold down the mouse key while drawing a rectangle around the desired elements.

To suspend the group, use the command “Ungroup”.

The following figure shows grouping (from left to right) and ungrouping (from right to left) for two rectangle elements:

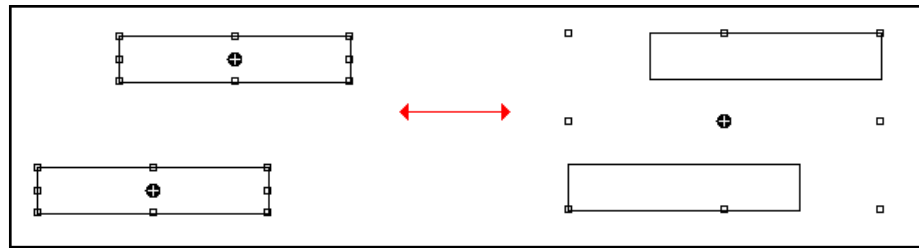


Fig. 20-2: Group and Ungroup

Also refer to:

- [chapter 20.2.9 ""Ungroup" command" on page 772](#)

20.2.9 "Ungroup" command

Symbol:

This command (category **Visualization commands**) is used to dissolve a group of visualization elements. The individual elements are displayed as individually selected.

Also refer to:

- [chapter 20.2.8 "Command "Group"" on page 771.](#)

20.2.10 "Frame selection" commands

This command (category **Visualization command**) is used to configure a frame ([chapter "Element 'Frame'" on page 866](#)).

A frame element is used to define a visualization in a subunit that can contain one or several visualizations. One of them is only active in online mode. It can be toggled between the available visualizations. Generally, the first visualization from the list is active ("[Switch frame visualization" on page 0](#)).

This allows switching back and forth among a variety of displays within a superordinate visualization, in contrast to IndraLogic 1.x, where users had jump from one visualization to another.

References are visualizations assigned to a frame. This means, they instantiate an original visualization and can add locally valid values to placeholders that were defined in the original visualization.

If a frame is selected in the editor, the visualization can be selected via the active command **Frame selection**. The dialog **Frame configuration** can be used.

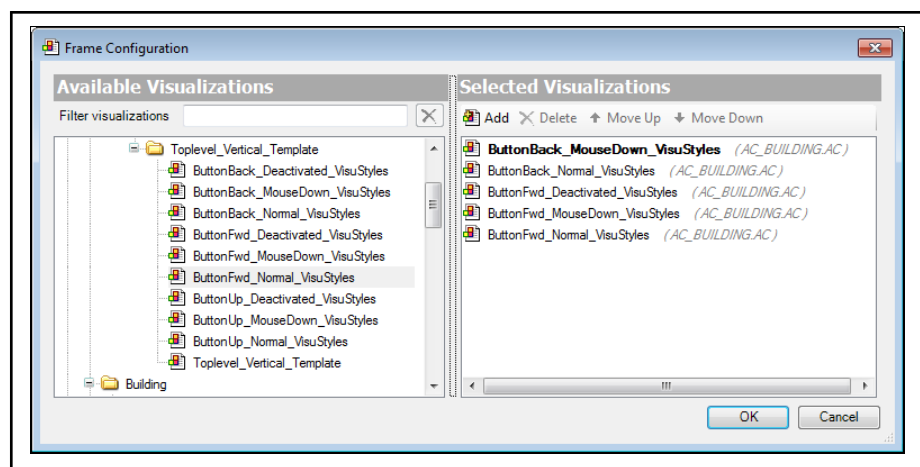


Fig. 20-3: 'Frame configuration' dialog

On the left side are the **Available visualizations** in the project. Select those that are to be referenced in the frame. By double-clicking on the command **Add**, add the visualizations to the list of **Selected visualizations** on the right side of the dialog. Selected visualizations can be removed from the list by double-clicking or by selecting the **Delete** command. Multiple selection of visualizations is also possible. Change the order in the list by clicking **Move up** and **Move down**.

It is recommended to assign only visualizations of the global pool to a frame. Otherwise, problems could occur if a device or an application object is re-named later and the path of the assigned visualization is no longer valid.

The order of the selected visualizations from top to bottom determines the automatically generated implicit index numbers of the visualizations. The top-most visualization is assigned 0 and the following visualizations are assigned 1, 2, etc. The index numbers are required to configure the toggle function of another element. The visualization is initially displayed with index "0".

Example

Use a button bar to specify the visualizations to be displayed in online mode.

1. Insert a frame element
2. All visualizations to be toggled are to be assigned to this frame using **Frame selection** in the **Frame configuration** dialog.
3. Add a control element for each visualization that allows switching to the visualization following user input. For example, an button bar with control elements is inserted.
4. Each control element is configured by configuring its input configuration:
In the `OnMouseClicked` properties, the follow-up action **Toggle frame visualization** is added and the visualization is assigned to a frame.

Start visualization

The "Start visualization", i.e. the visualization object to be displayed first after logging into the control with the application, has to be added below the respective application object in the Project Explorer.

If only one visualization is assigned to the application, it is automatically used as start visualization.

If several visualizations are assigned to the application, select the start visualization explicitly in the visualization manager. Afterwards, this option is automatically disabled in all other visualization objects.

The start visualizations below an application switches to online mode with the application.

Since visualizations in the "General module folder" do not have a direct reference to an application, these do not switch to the online mode.

If a change in visualization to a visualization from the folder "General module folder" is configured in a visualization below the application, it can thus be reached in online mode.

Also refer to:

- [chapter "Element 'Frame'" on page 866](#)
- ["Switch frame visualization" on page 0](#)
- [chapter "Input configuration" on page 800](#)

20.2.11 Command "Background"

Symbol: 

This command (category **Visualization commands**) opens the “Background” dialog. In this dialog, a color and/or an image can be selected for the visualization background. Select the desired option(s):

Bitmap: To define a background image, enter the path of an image file available in an image pool in the project here.

In addition, enter the name of the image pool and the ID of the image file, separated by a ".":

`<ImageCollection>.<ID>`

(e.g., "Images_1.drive_icon>", "Images_1.43").

Color: To define the background color of the visualization, select the desired color from the color selection list.

20.2.12 Command “Select all”

Using this command (category **Visualization commands**), all elements of the visualization currently open in the editor are selected.

Also refer to:

- [chapter 20.2.13 "Command “Deselect all”" on page 774](#)

20.2.13 Command “Deselect all”

Using this command (category **Visualization commands**), all elements of the visualization currently open in the editor are deselected.

Also refer to:

- [chapter 20.2.12 "Command “Select all”" on page 774.](#)

20.2.14 “Configure trace...” command

The command is available if a “Trace” element is selected in the visualization editor.

Use the command to open the dialog "Trace configuration, trace settings".

Also refer to:

- [chapter ""Trace” element" on page 972](#)
- [chapter ""Trace configuration, recording settings” dialog" on page 981](#)

20.2.15 “Insert elements for trace control” command

The command is available if a “Trace” element is selected in the visualization editor ([chapter ""Trace” element" on page 972](#)).

The command opens the [chapter 20.2.15 ""Insert elements for trace control” command" on page 774](#) dialog.

20.2.16 “Update frame parameters in <opened visualization>” command

The command is only available and visible in the adjusted menu if the visualization editor is open.

This command is only required in case of compile errors, affecting frame parameters. Use the following dialog to update the frame parameters.

The parameterized visualization is part of a library. It was updated in the library and an automatic update of the project parameters did not take place yet.

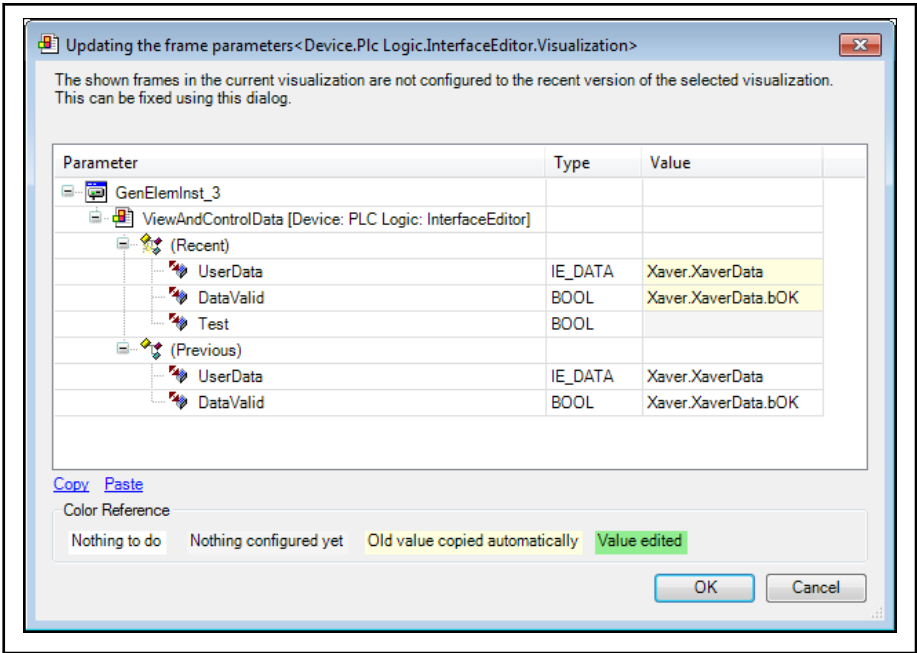


Fig. 20-4: Updating the frame parameters <visualization name>

The visualization to be reconfigured is displayed in brackets <...> in the dialog title.

Dialog use:

- chapter "Updating the frame parameters" on page 784

20.2.17 “Multiply visualization element” command

Symbol:

Function: The command opens the **Multiply visualization element** dialog to configure the process of multiplying the elements.

Call: **Visualization** menu, context menu

Prerequisite: The visualization is active and a template element is selected.

“Multiply visualization element” dialog

IndraLogic inserts more elements, resembling the template element, during the process of multiplying elements. This dialog facilitates configuring the number and order as well as the replacement of the indexes.

“Basic settings” tab

Total number of elements

IndraLogic inserts new elements as table. The line number is set in horizontal , the column number is set in vertical . The product of both numbers equalizes the total number of elements to be inserted.	
Horizontal	Number of elements per line Default setting: Corresponds to the number of components in \$FIRSTDIM\$
Vertical	Number of elements per column Default setting: Corresponds to the number of components in \$SECONDDIM\$

Offset between the elements

IndraLogic aligns the visualization elements as table. If an **offset** is specified, a space is inserted between the elements.

- **0** : The element frames overlap by 1 pixel
- **1** : The elements come into contact with each other
- **<n>** : There is a visible distance of n-1 pixel between the elements

Horizontal	Line width of the elements in pixel
Vertical	Column distance of the elements in pixel

"Advanced settings" tab

First dimension

Start index	Start index for \$FIRSTDIM\$ Default setting: 1 indicates that the index for \$FIRSTDIM\$ starts with 1. Example: Array[1, <\$SECONDDIM\$>]
Increment	Increment by which the index is incremented Default setting: 1

Second dimension

Start index	Start index for \$SECONDDIM\$ Default setting: 1 indicates that the index for \$SECONDDIM\$ starts with 1. Example Array[<\$FIRSTDIM\$>, 1]
Increment	Increment by which the index is incremented Default setting: 1

Also refer to

- [chapter "Multiplying an element" on page 817](#)

20.3 Editor

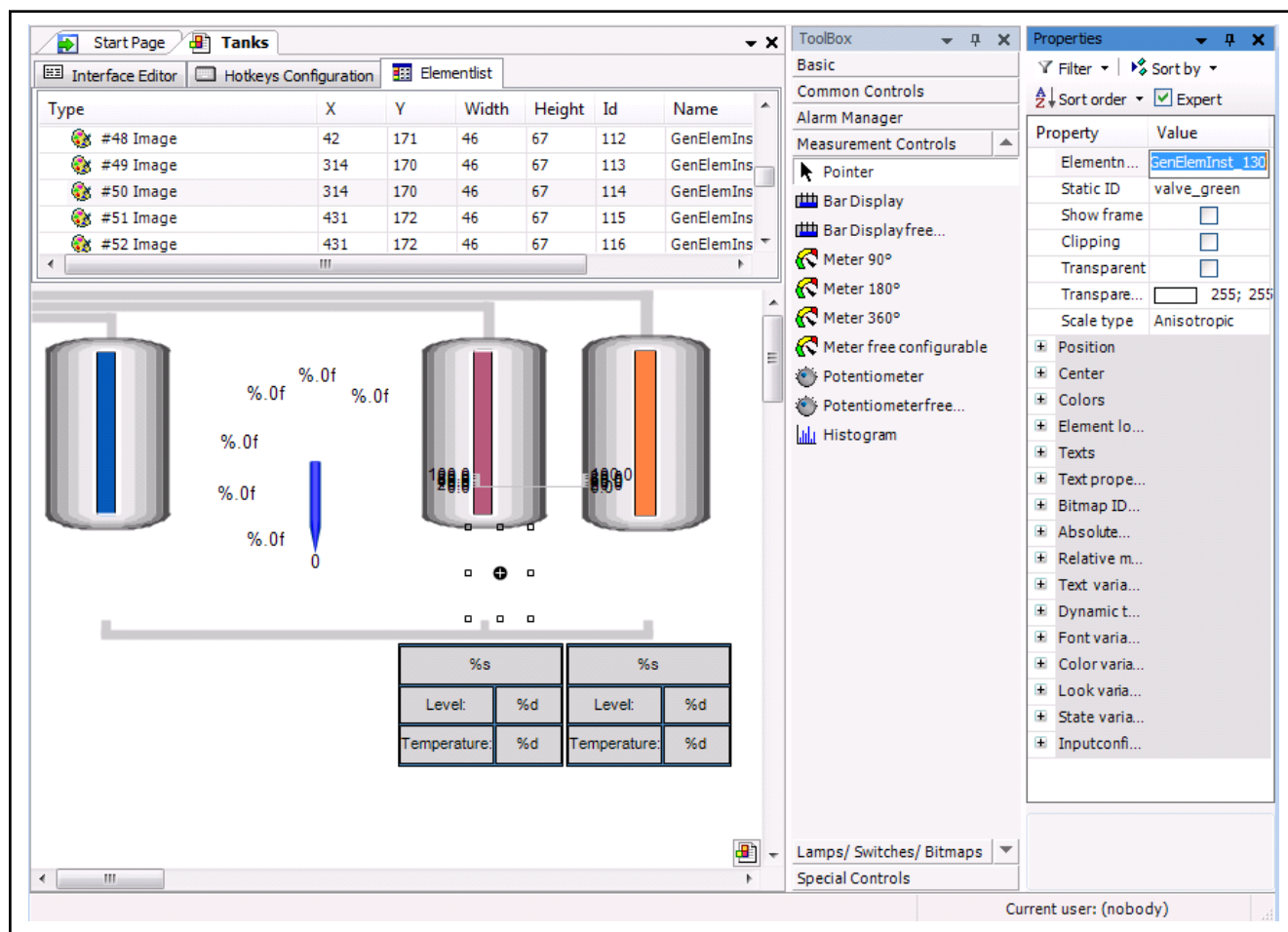


Fig. 20-5: Visualization editor

The visualization editor is available via the “Visualization editor” plug-in.

A visualization can be created by using the following editors and tools:

- the editor itself, labeled  <visualization name>[<object path>] containing the edited and programmed visualization.
- the **Tools** view containing currently available visualization elements.
- the **Properties** view containing all properties of the visualization element currently selected in the editor.
- the commands of the **Visualization** and the commands of the context menu in the editor
- the **Interface editor** to define placeholders if the visualization is to be inserted into another visualization.
- the **Keyboard configuration** to assign follow-up actions to keys or shortcuts. Please note that the device using the visualization supports the respective keys.
- the **Element list** providing an overview of all elements and their position of the current visualization and allows selection, multiple selection and deleting.

If **Interface editor**, **Keyboard configuration** or **Element list** is activated via a command (by default, in the **Visualization** menu), the visualization editor is divided into two parts and has corresponding tabs in the upper part.

IndraLogic 1x visualizations can be imported.

The visualization elements can be animated by directly using IndraLogic project variables or in the form of expressions, i.e. in combination with operators and constants. For example, this facilitates scaling of variables for use in the visualization.

Within a visualization, any expressions, also function calls, are permitted.

20.3.1 “Tools” view

Symbol: 

Function: The view provides elements in a table as preview that can be moved to the visualization editor using drag&drop. Only preview images with at least one ID which is selected and green is visible.

The selection can be changed by clicking on the ID buttons. IndraLogic automatically generated the IDs.

However, the user can also add IDs. Moreover, set the preview image size using the scrollbar or search for the element name using the full-text search.

Call: MenuView ► Tools

Prerequisite: A visualization is active.

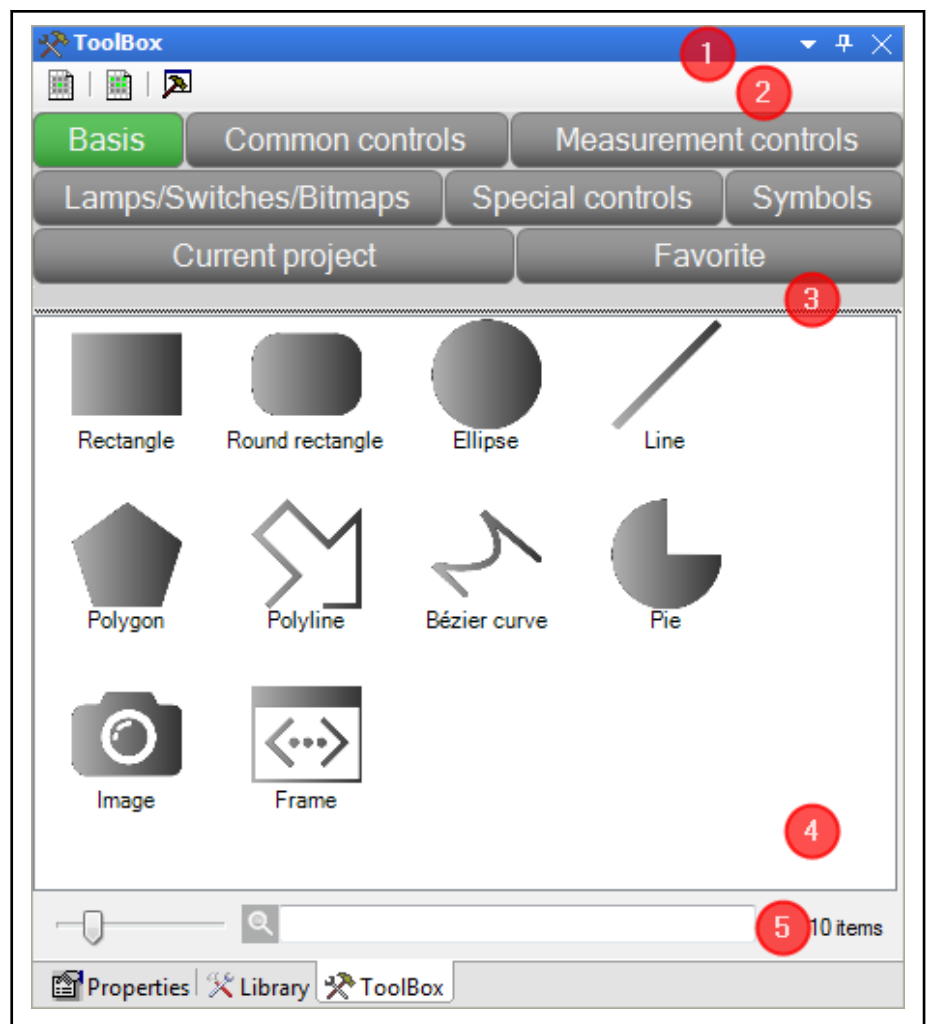


Fig. 20-6:

- (1): **Tools** view
- (2): Symbol bar for commands
- (3) : ID buttons
- (4) : Preview images of labeled elements in table
- (5) : Control elements

Toolbar

	"Single selection" selection type: In the ID buttons a maximum of one ID is selected and highlighted in green
	"Multiple selection" selection type: In the ID buttons, several IDs can be selected and highlighted in green.
	The Manage IDs and entries dialog is displayed.

ID buttons

IndraLogic shows an button for each automatically assigned or user-defined ID. Each ID is assigned to at least one entry. The entry can also be assigned to several IDs.

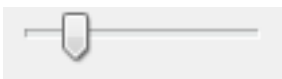
Button	The button is labeled with the ID name. - Gray button, for example . The ID is not selected and the assigned entries are not displayed. - Green button, for example . The ID is selected and the assigned entries are displayed.
Click on a gray button	Button turns green and the ID is selected. The assigned entries are displayed. The behavior of the other IDs varies, depending on the activated selection type.
Click on a green button	Prerequisite: Multiple selection is active Button turns gray and the ID is not selected. The assigned entries are not displayed anymore.
Click on a button during a pressed <Shift key>	The button switches. The selection behavior is also switched for this click and the behavior of other IDs varies, depending on the activated selection type. If Single selection is activated, the ID that was clicked on is additionally selected. If Multiple selection is activated, only the ID that was clicked on is selected.
Hide identification	Removes the ID from the tabular arrangement. Configure an ID as visible in the Configure IDs and entries dialog. Prerequisite: A button is focused and the context menu is activated.

Tabular view of preview images

Elements determined by IndraLogic from the project or the libraries are displayed with preview image and are labeled with a name, depending on the ID.

If the object property **Internal** was enabled for the image pool, IndraLogic does not display the image pool content in the **Tools** view.

Control elements

Scrollbar 	Scrollbar to change the preview image size
 with input field	For full text search. IndraLogic filters the entry names according to the string entered in the input field and displays the affected entries.
<Number> entries	Number of element entries selected via the ID and filter buttons

Also refer to

- [chapter 20.1.2 "Displaying and labeling visualization elements" on page 759](#)
- [chapter 20.3.7 "'Configure IDs and entries' dialog" on page 794](#)
- [chapter "'Tools' command" on page 590](#)

20.3.2 "Properties" view

Symbol: 

The plug-in "Visualization element" provides the command **View ► Element properties** used to open the **Properties** editor for a selected visualization element. If required, open the dialog **Tools ► Adjust** to view or change the menu configuration of the category **View**.

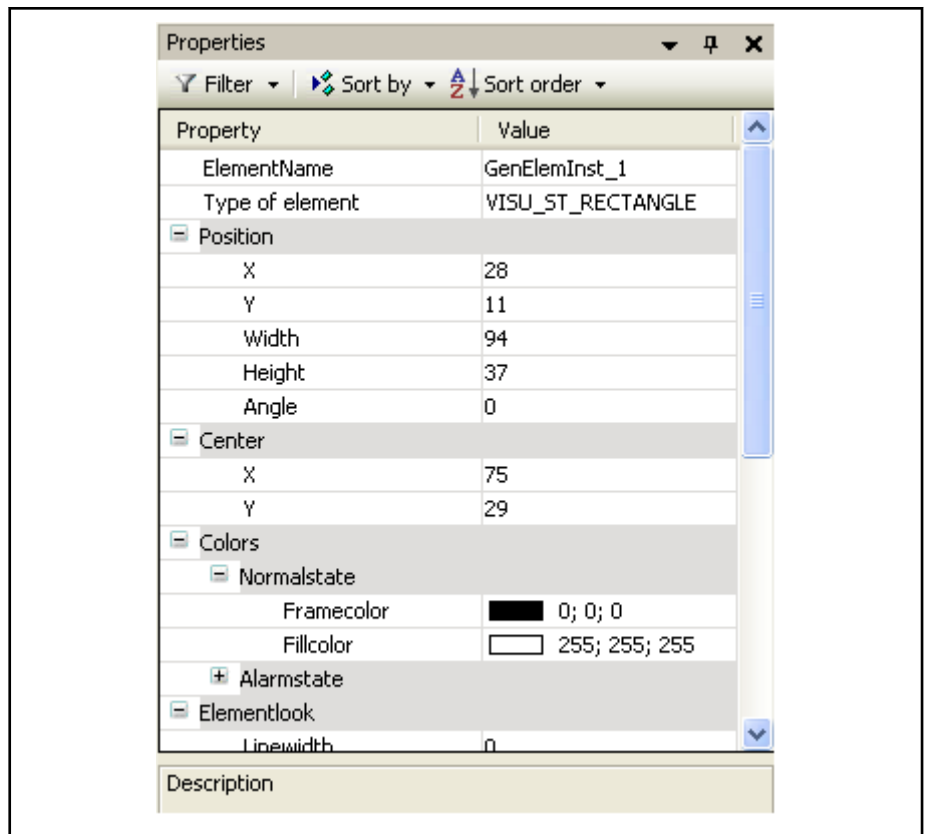


Fig. 20-7: Element properties editor

The **Properties** editor lists all properties of the selected element in a table and provides different user inputs to change the **value**. A description of the properties and their use is listed in the element description: [chapter 20.4 "Visualization elements" on page 795](#).

The menu bar provides commands and options:

- **Filter:** Filter settings selection to adjust the property list to the user requirements.
 - **Display properties of all categories:** all properties
 - **Simple:** only basic properties such as **texts**, **colors** and **input configuration**
 - **Standard:** the properties used the most (the selection is based on a property block as used in V2.2).
 - **Animation:** only properties used for animation
 - **Colors:** only properties referring to colors
 - **Color gradient:** only properties referring to the color gradient
 - **Texts:** only properties referring to text
 - **Inputs:** only properties referring to user input
- **Sort according to:**
 - **Sort according to type:** complete properties list in the original category order
 - **Sort according to name:** complete properties list in alphabetic category name order
- **Order:**

- **Sort ascending:** complete properties list in the original category order from bottom to top
- **Descending:** complete properties list in the original category order from top to bottom
- **Expert** checkbox: if this option is activated, all properties are displayed and **Show all categories** is selected.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)



If a style is used, the drop-down list of colors and fonts depends on the definitions in the respective style file.

In visualizations which are part of a device, there are device-independent limitations affecting the configuration options.

Examples: List of available elements, fonts, colors, image format, maximum number of visualization elements in a visualization, maximum number of visualization in a device.

20.3.3 "Interface editor" tab

It is used to define frame parameters in the visualization. The frame parameters are referenced in the frame element of another visualization.

Basic information on the interface editor

Declaring an interface

By default, this part of the  visualization editor can be disabled and enabled using the command "Interface editor"  in the **Visualization** menu ([chapter 20.2.2 "Command "Interface editor"" on page 766](#)). It is displayed in the upper section of the visualization editor as a separate tab.

As a visualization is treated as a function block, the interface editor is displayed as a regular declaration editor in the upper section of the visualization editing window. Input variables as well as frame parameters can be declared.

Declaring parameters using:

- VAR_INPUT
- VAR_INPUT with [chapter "Attribute 'parameterstringof'" on page 790](#)

- VAR_IN_OUT: if a data structure is transferred, define it as VAR_IN_OUT. Without pragma, the parameter values are copied if the visualization is open. Entries are written to the copied data structures and written back to the parameters when closing the visualization.
- VAR_IN_OUT with [chapter "VAR_IN_OUT_AS_POINTER" attribute" on page 787](#): It is permitted to transfer the objects to the visualizations as references. It is used for the diagnostic visualization, WebVisu and TargetVisu. It can be useful if information is sent to the visualization without copying that information. This applies as well when updating information from outside while the dialog is open.

Example:

Declaring the UserData interface

The ViewData visualization uses the interface "UserData" of the type "IE_Data".

The element GenElemInst_5 uses "UserData" and displays the variable value of UserData.iCounter, as it is programmed in the properties view of "GenElemInst_5".

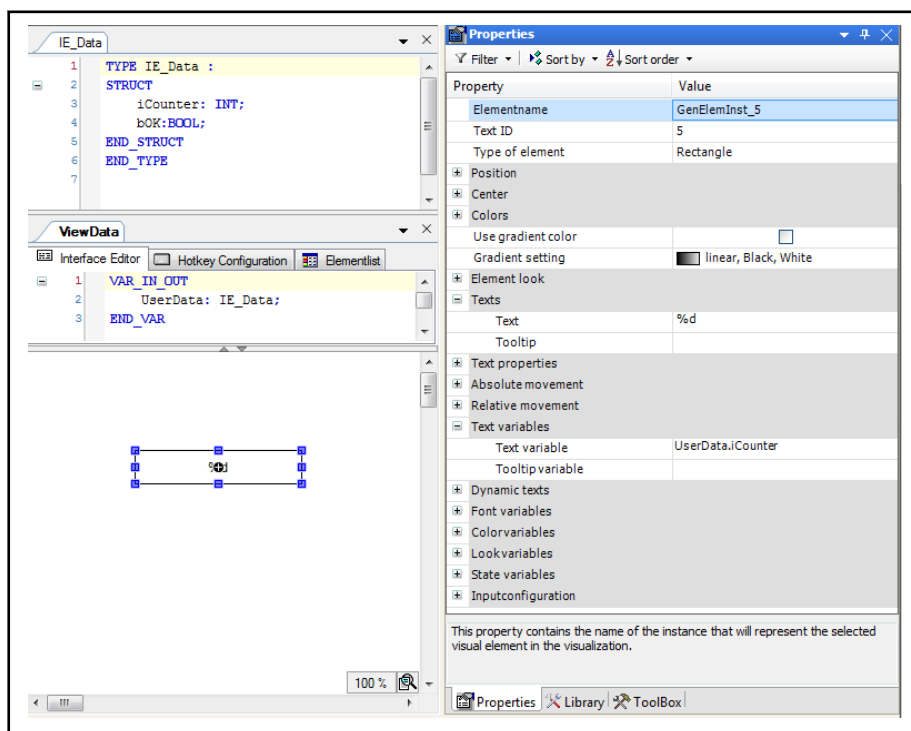


Fig. 20-8: Declaring the interface

Programming a frame

1. Add a frame element to the main visualization.
2. Set the reference to a visualization with interface that is not only local but also global or that is part of a library.
3. Assign an IEC variable/expression to each frame parameter.

If the visualization with interface is referenced in another visualization, these frame parameters can be programmed by assigning locally used expressions. Make these replacements in the **Properties** of the frame element. The inserted (instantiated) visualizations and its input variables are executed there and expressions that can be used in the local configuration of the instance can be assigned to the variables.

Example:

Programming a frame with the "ViewData" reference

The "Frame" element references the visualization ViewData (chapter "Element 'Frame'" on page 866).

In the properties view, this assignment is listed with all frame parameters. Click on the value field of such a variable and an IEC variable of the same type is assigned. Click on the value field of such a variable and an IEC variable of the same type is assigned. In this case, Hans.HansData is assigned to UserData.

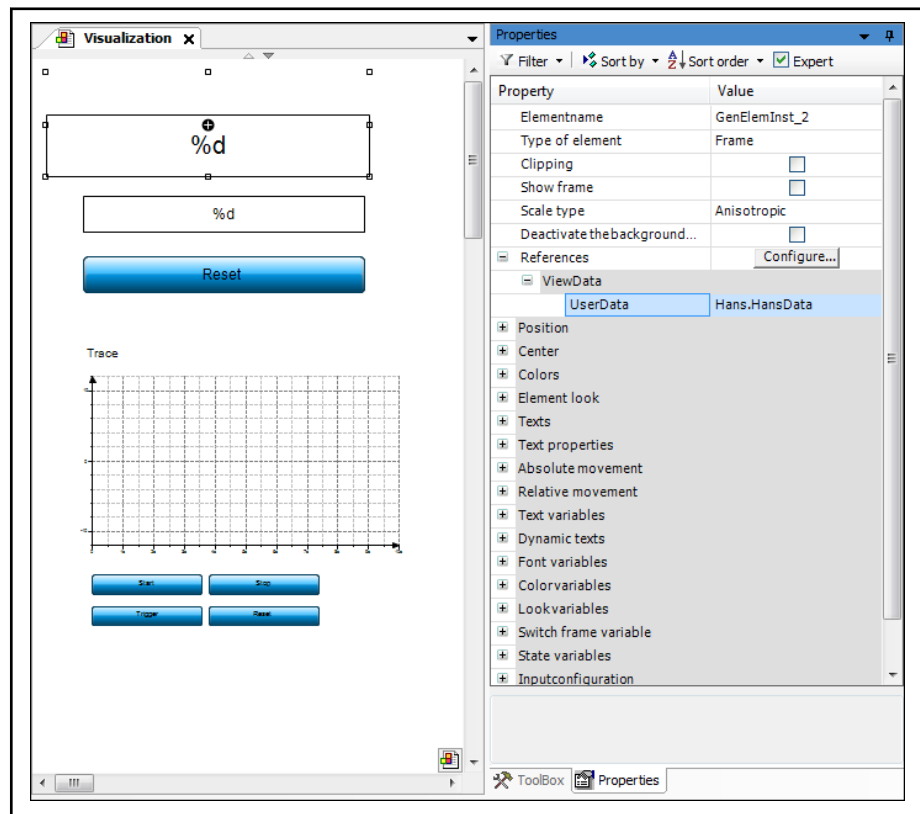


Fig. 20-9: Frame reference "ViewData"

Update frame reference

If the interface of a visualization which is referenced in the frame of another visualization is modified, the dialog "Updating the frame parameters" dialog is automatically opened when trying to save or compile the project or when trying to open an affected object (chapter "Updating the frame parameters" on page 784).

If required, configure the frame parameters again in this dialog.

Updating the frame parameters

If the interface of a visualization inserted into a frame is modified, this dialog is opened automatically when saving or compiling the project or when opening an affected object. In this dialog, the frame parameters can be adjusted.

In case that the visualization interface originating from a library has been changed, the project parameters also have to be updated.

Use the following command to open the Update dialog:

- chapter 20.2.16 ""Update frame parameters in <opened visualization>" command" on page 774

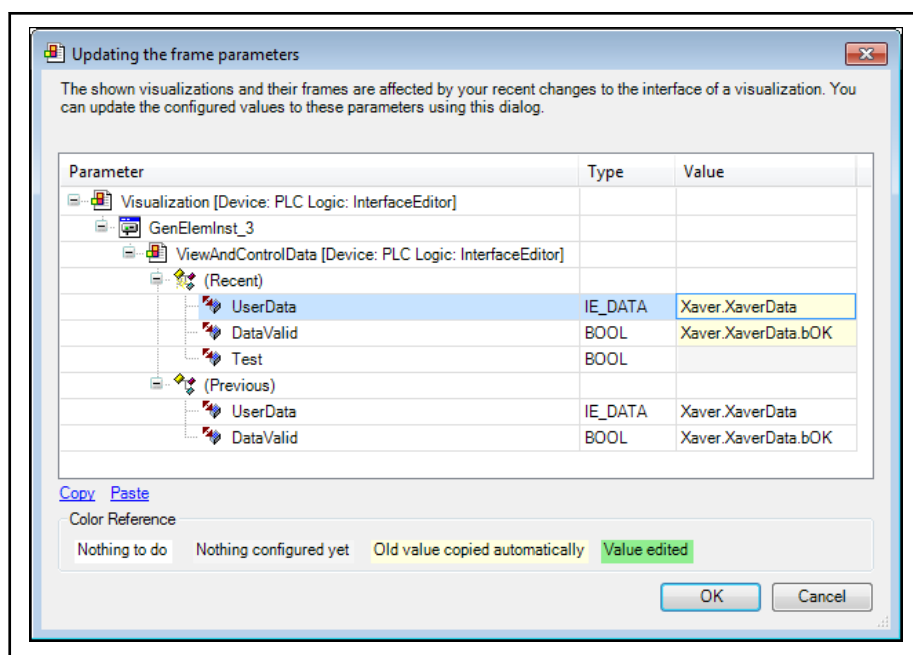


Fig. 20-10: Updating the frame parameters

Parameter: in this column, the changed parameters are displayed in tree view.

- The visualization is on top, then the changed frame.
- Subsequently, the referenced visualization and its interface.
- The currently valid parameters are listed under **(Current)**.
- The previously used declaration is contained under **(Previous)**.

Type: The respective parameter type is displayed in this column.

Value: The value field contains variables assigned to the parameter. Note the color of the field:

Color reference

- Beige background: This parameter is automatically applied from the previous configuration
- Gray background: No new value has been assigned to this new parameter
- Green background: This new parameter has already been assigned a value.
- White background: Nothing needs to be configured here.

Editing values under **(Current)**:

Click on the value field to select it. Click again on the field, press the <space bar> or start typing to open a line editor.

Assign another variable by entering its name or select a variable using the input assistance.

An existing value entry can be copied to another field. Select the entry to be copied, use <Copy> and select the field in which the entry is to be inserted and use <Paste>.

Confirm the dialog settings with <OK>.

The dialog closes and new parameter assignments are displayed in the **Properties** dialog, category **References** in the relevant frame element.



If you would like to use a higher-level variable within the referenced visualization block of the instance name, use the pragma attribute `chapter "Attribute 'parameterstringof'" on page 790` which provides a corresponding string.

Adding parameter DataValid

Expand the interface of the ViewAndControlData visualization by the **DataValid** variable.

Interface editor

```
VAR_IN_OUT
  UserData: IE_Data;
  DataValid: BOOL;
END_VAR
```

When now trying to compile and save the project or when trying to open Visualization, first the dialog to configure the new parameter is displayed.

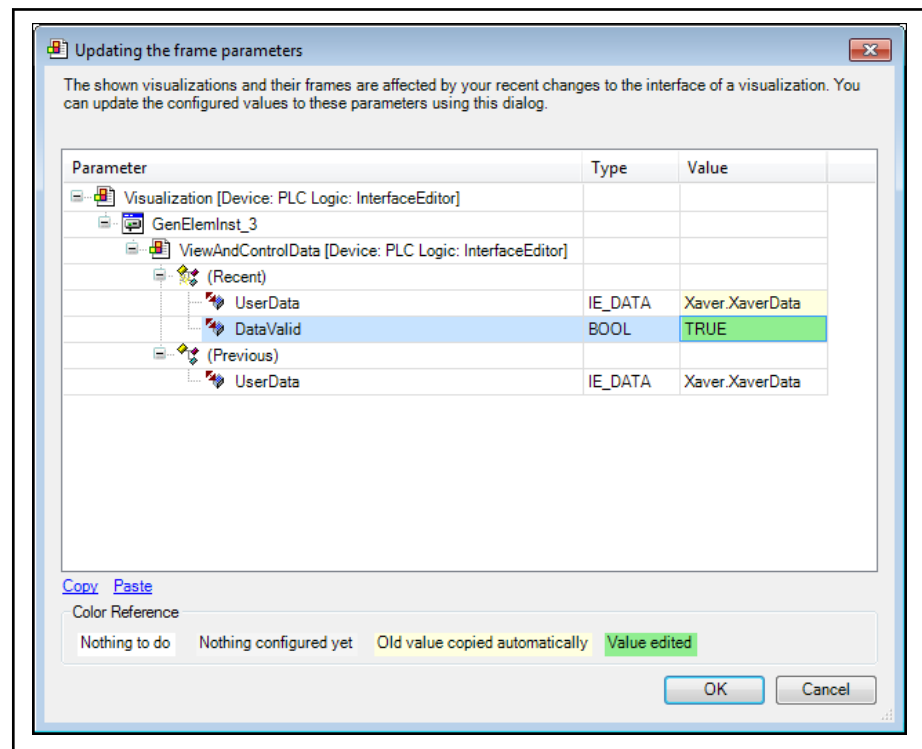


Fig. 20-11: Updating the frame parameters

Compare the current parameters to the previous parameters. Enter "TRUE" in the DataValid value field. Close the dialog with <OK>. The new parameters are contained in the **Properties** view, in the category **References** of GenElemInst_3.

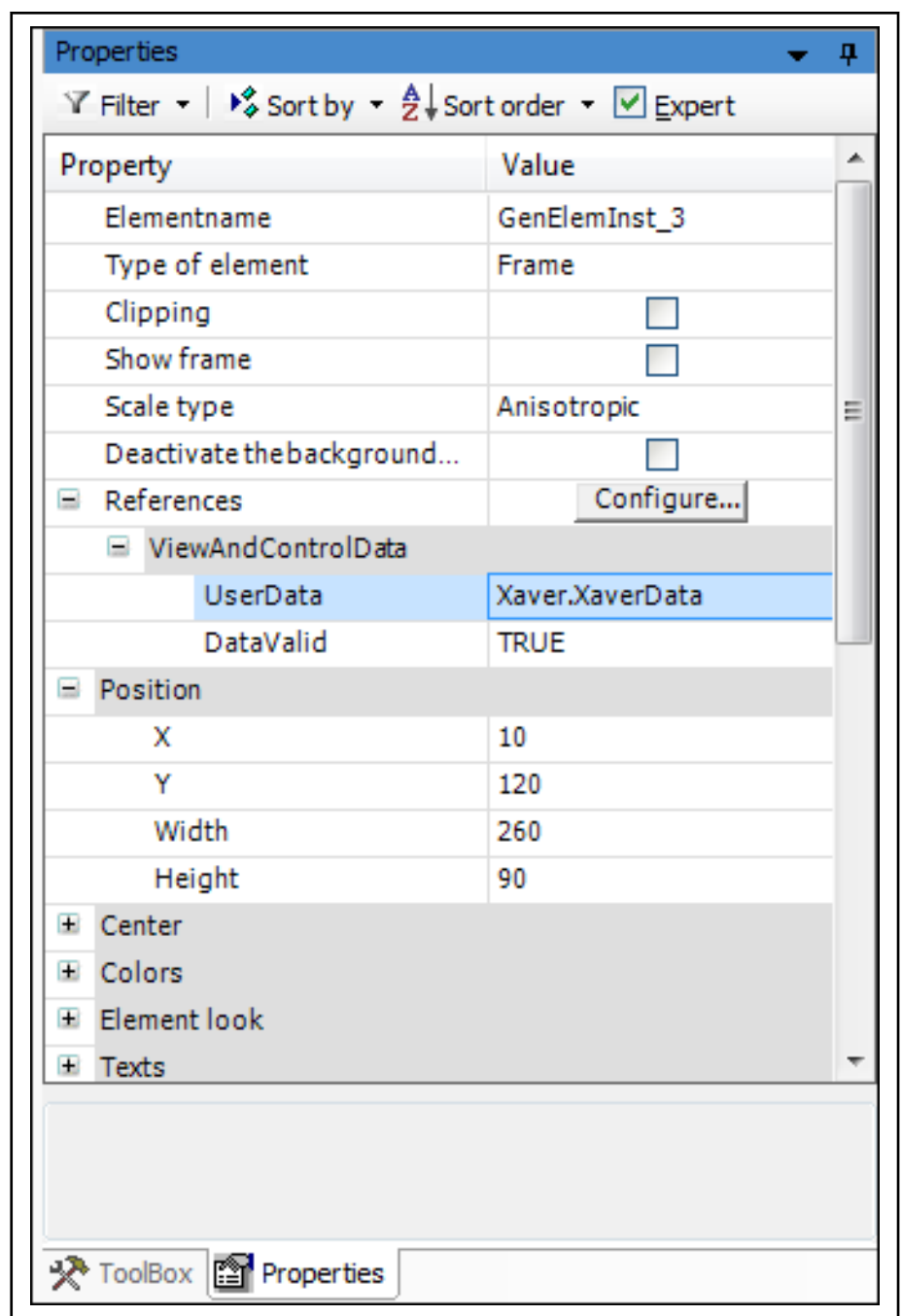


Fig. 20-12: Properties view after the frame parameters have been updated

Pragmas

Pragmas that can be used in the interface editor:

“VAR_IN_OUT_AS_POINTER” attribute

These pragmas facilitate using the reference to an object as frame parameter. This is possible in a visualization declared as dialog.

Syntax:

```
{attribute "VAR_IN_OUT_AS_POINTER"}
```

If references are used, proceed as follows:

1. Declare your user data object (DUT).
2. Declare a frame parameter as VAR_IN_OUT in the interface editor of a dialog as reference to the data object.
Provide the attribute "VAR_IN_OUT_AS_POINTER" to the frame parameter.
3. Program the user interface:
Use the dialog in a visualization or in the input configuration of a visualization element. In both cases, referenced data can be accessed.

Using an interface with pragma "VAR_IN_OUT_AS_POINTER"

```
FUNCTION_BLOCK ControlFB
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    bOk : BOOL := TRUE;
    nCounter : INT;
    nValue : INT;
END_VAR
nCounter := nCounter + 1;
```

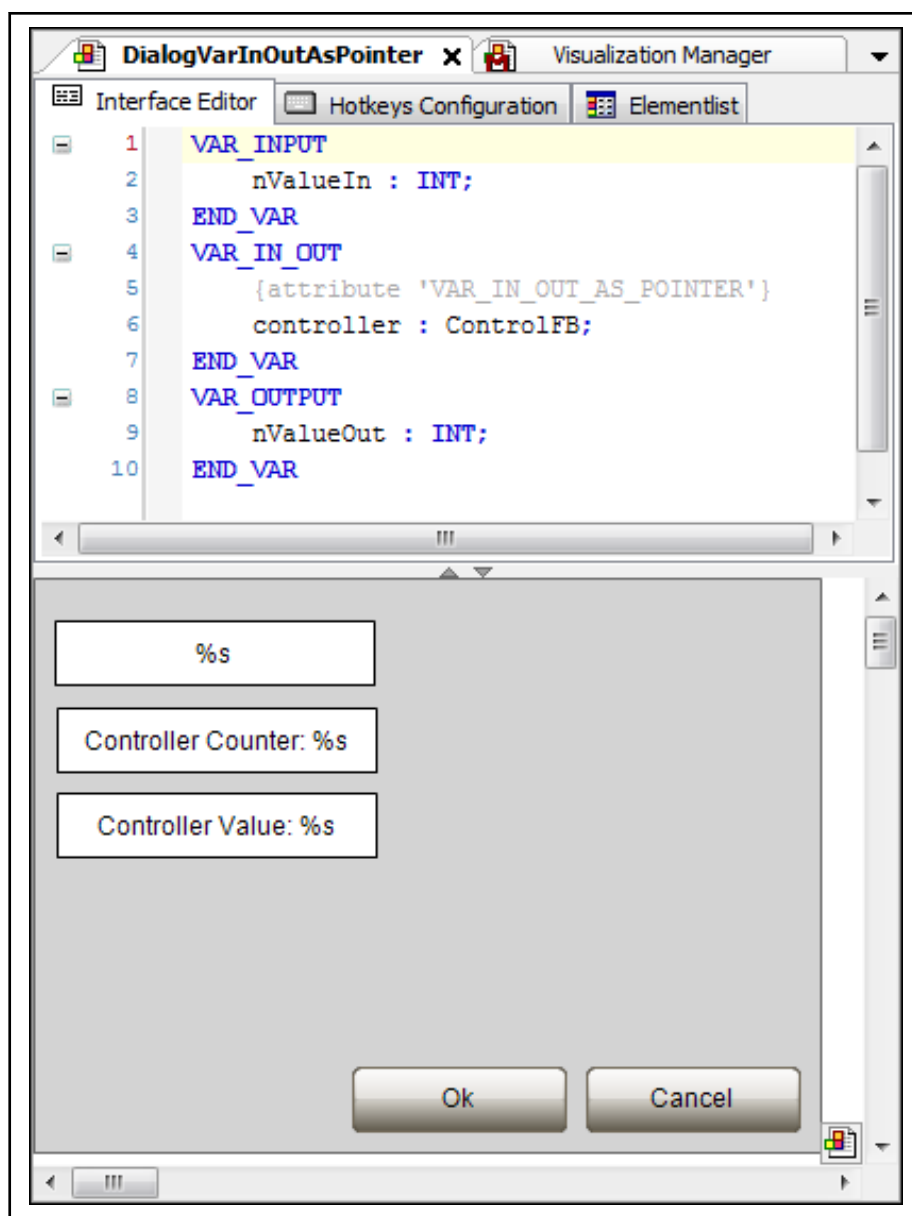


Fig. 20-13: Frame parameter declaration with VAR_IN_OUT_AS_POINTER

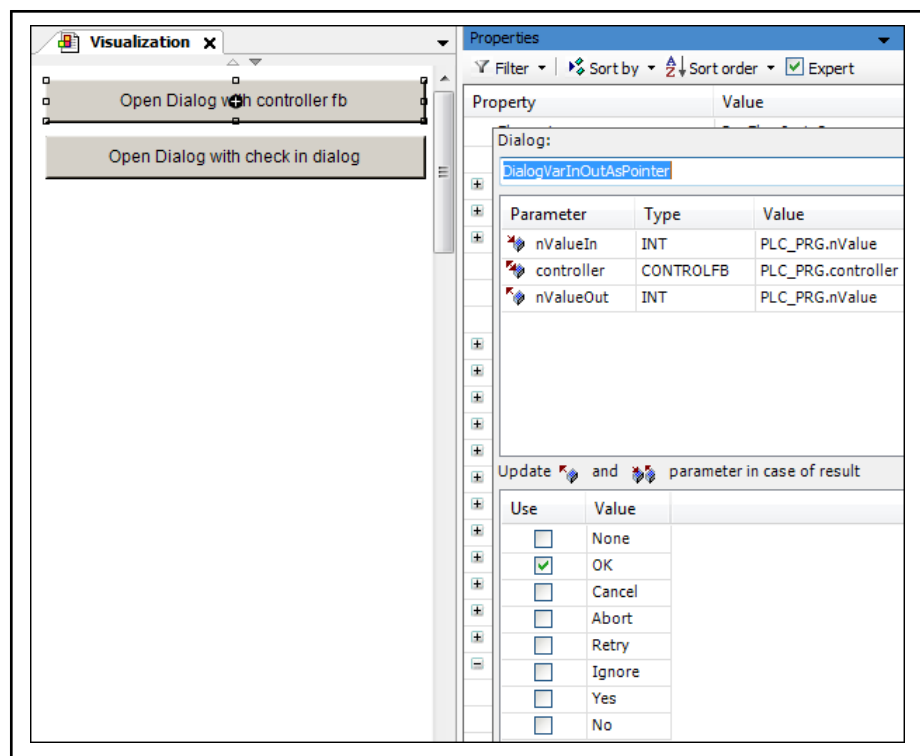


Fig. 20-14: User interface: Dialog opens

Attribute 'parameterstringof'

The attribute of the pragma {Attribute 'parameterstringof'} can be used to provide the instance name of a variable available to a visualization block.

Syntax:

```
{attribute 'parameterstringof' := '<variable>')}
```

Example:

In the main program, the instance myDUT of the user-defined structure DUT is created:

```
PROGRAM PlcProg
VAR
    myDUT: DUT;
END_VAR
```

This instance is the input of a visualization block "Vis" (in the input parameter "instance") which is referenced by a frame of another visualization "MainVisu":

Section from the properties of a visualization element of MainVisu

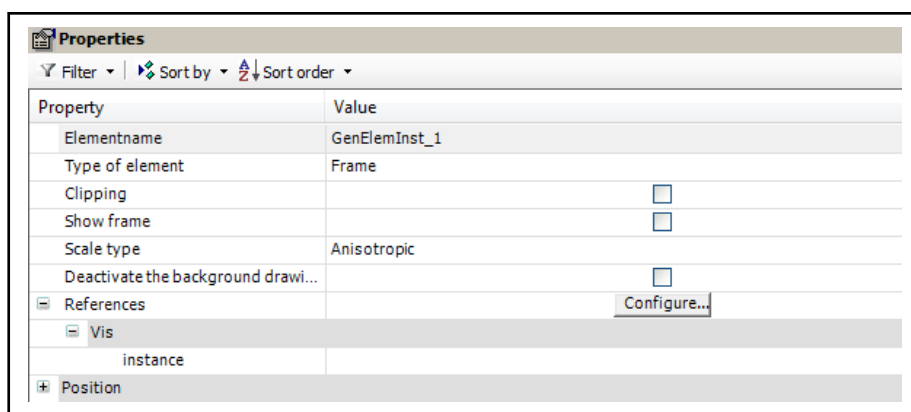


Fig. 20-15:

In the interface editor associated with “Vis” is the input/output variable `instance` and another input variable called `instanceStr`:

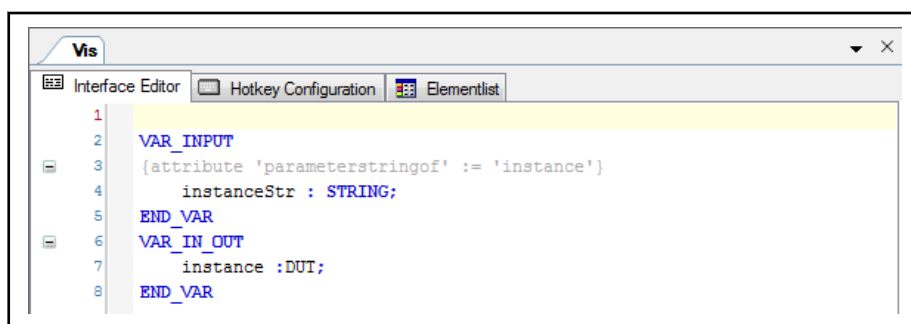


Fig. 20-16:

Although `instanceStr` is an input variable, it is not executed as an input in the reference (compare the figure above!).

This is caused since the variable `instanceStr` has the 'parameterstringof' attribute and is therefore automatically initialized with the name of the variables specified with the attribute.

In our example, `instance` is the associated variable, so string variable `instanceStr` is set to `PlcProg.myDUT` and can only be used within the visualization `Vis`, e.g. as a text variable for a placeholder `%s`.

20.3.4 Tab 'Keyboard configuration'

The keyboard configuration can be found via the “Keyboard operation” command in the upper part of the visualization editor.

Also refer to:

- [chapter 20.2.3 "Command “Keyboard usage”” on page 766](#)
- [chapter "Basic information on the interface editor" on page 782](#)
- [chapter 20.3.5 "Tab “Element list”” on page 793](#)

Basic information

In addition to the keyboard operation in online mode, special hotkeys can be defined to operate a visualization in online mode in this section of the visualization editor ([chapter 20.11 "Keyboard operation in online mode" on page 1018](#)). This means that an action can be assigned to a specific key or shortcut. This hotkey applies only to the respective visualization.

Define shortcuts to be used in all visualizations of the application in the visualization manager ([chapter ""Visualization manager” - “Default keyboard configuration”” on page 998](#)).



The **Device description** defines which keys are supported in a visualization that runs on the device!

Note the following call sequence to process key actions:

1. Event handler of the application, if enabled (optional); e.g. [chapter "Noticing events and input actions" on page 1011](#)
2. Keyboard configuration in the "Visualization manager" - "Standard keyboard configuration" that applies to all visualizations of the application ([chapter "Visualization manager" - "Default keyboard configuration" on page 998](#)).
3. Define the keyboard operation in online mode ([chapter 20.11 "Keyboard operation in online mode" on page 1018](#))
4. Special hotkey configurations of individual visualizations (managed in the hotkey configuration). The main visualizations are considered before those in the frames.

In this context, also note the **Key** property of a visualization element, which can be configured in the element properties under **Input configuration**. Such an element-specific hotkey is also managed in the **Hotkey configuration**. It can be edited in both locations and it is updated in the respective other editor.

Editor

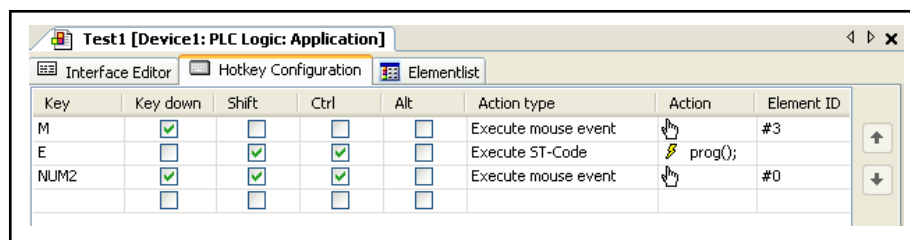


Fig. 20-17: Keyboard configuration editor

In each line of the table editor, a key or a hotkey can be linked with an action.

See the following columns:

Key: Name of the key. The key name can be input manually or using a selection list that opens by double-clicking the cell. The list contains all keys defined via the **device description**. In the case of a visualization that is not assigned to any device, all keys are supported by all devices.

Key down: If this option is selected, the action is executed as soon as the key is pressed. Otherwise, it is executed when the key is released. If the action is to be executed both when the key is pressed (KeyDown) and released (KeyUp), two corresponding definitions for the key have to be present in the table.

Shift, Ctrl, Alt: If this option is selected, the <Shift> or <Ctrl> or <Alt> key have to be pressed together with the key to execute the action.

Action type: It can be defined using a selection list. To open it, double-click on the cell. The action types correspond to those available in the **Properties** category **Input configuration** of a visualization manager.

Action: Exact configuration of the action to be executed. It depends on the action type and corresponds to the mouse action as it can be used in the **input configuration** of an element.

Element ID: ID of the visualization element to which the key is assigned to (using the "Hotkey" **Input configuration** ► **Keys**). Unique identification in the current visualization.

If a key is linked with several actions, these are executed in the same sequence as they appear; from top to bottom in the configuration table. This order can be changed by selecting a key definition (click in the table line) and using the **arrow keys** on the right of the table to move up or down.

20.3.5 Tab "Element list"

This part of the visualization editor provides a list of visualization elements of the currently open visualization.

The element list can be opened via the "Element list" command (by default, in menu **Visualization**) and is available as separate tab apart from other editors such as the interface editor or the editor for keyboard configuration ([chapter 20.2.4 "Command "Element list" on page 766](#)).

Type	X	Y	Width	Height	Id	Name	Access rights
#0 Button	50	60	150	30	0	Login	
#1 Button	50	110	150	30	1	Logout	
#2 Button	50	170	150	30	3	Change PW	
#3 Button	50	230	150	30	5	Open User Config	
#4 Rectangle	378	64	150	30	6	GenElemInst_7	
#5 Rectangle	586	65	150	30	10	GenElemInst_11	
#6 Round rectangle	402	145	221	52	7	GenElemInst_8	
#7 Ellipse	413	245	150	30	8	GenElemInst_9	
#8 Button	50	350	150	30	9	GenElemInst_17	
#9 Button	50	390	150	30	13	GenElemInst_19	
#10 Rectangle	413	352	262	30	16	GenElemInst_21	

Fig. 20-18: Element list

The elements are listed from top to bottom according to their positions on the z-axis of the visualization. The element furthest away in the background in the first line. The following values are displayed but cannot be edited here:

Type: Element type and symbol as used in the view "Toolbox", as well as the element number that results from the insertion sequence first ([chapter 20.3.1 "Tools" view" on page 778](#)).

X, Y: Position of the upper left corner of the element (0,0 = upper left corner of the visualization area)

ID: Internally assigned element identification

Name: Element name as defined in **Properties** of the element (**Element name**)

Access rights: If the behavior of an element is limited for some user groups, this is indicated by the padlock symbol .

Element(s) can be selected by selecting the corresponding table line(s). The selection is always synchronized with that in the main window of the visualization editor. However, note that **subelements** of a group can only be selected in the element list.

The position of a selected element on the Z-axis, i.e. **on the background-foreground axis** can be changed using the following commands:

- [chapter "Command "Place on top" on page 771](#)
- [chapter "On level up" command" on page 771](#)
- [chapter "Command "Send to back" on page 771](#)
- [chapter "Command "One level down" on page 771](#)

The following commands can also be used:

- **Undo**

- Restore
- Delete

20.3.6 “Add visualization” dialog

Function: The dialog is used for creating a new object of type **Visualization**.

Call: Select application in the Project Explorer, select **Add**, select **Visualization** type

Name	Name of the visualization, for example <code>Visu_A</code>
Requirement for the following settings: They are displayed only when you add a Visualization object to the project for the first time.	
Symbol library	List of all installed symbol libraries
Assigned	 The symbol library is selected. When closing the dialog with Add , IndraLogic assigns the symbol library to the project and provides it in the Tools view. IndraLogic manages this setting in the Project settings.

Also refer to

- [chapter 20.1.2 "Displaying and labeling visualization elements" on page 759](#)
- [chapter ""Add" command" on page 605](#)

20.3.7 “Configure IDs and entries” dialog

Function: The dialog displays existing IDs in a tree.

Below an ID, elements are contained that are assigned to the ID. More IDs can be added.

Call: Click on the  symbol in view **Tools**

Also refer to

- [chapter 20.1.2 "Displaying and labeling visualization elements" on page 759](#)
- [chapter 20.3.1 ""Tools" view" on page 778](#)

tree view

Identification	Tree view of all IDs and their entries •  <Name>: IDs provided by IndraLogic •  <Name>: A user-created ID. Default setting: Favorite
	IndraLogic expands the tree The entries assigned to the ID become visible.
Active	 An ID interface is displayed in the Tools view. IndraLogic labels the button with <Name> .

Toolbar

+	The dialog Add ID opens.
- or <Del>	IndraLogic removes the ID from the tree view. When exiting the dialog with OK , IndraLogic also removes the interfaces in the Tools view. Prerequisite: An ID is selected

Tab. 20-1:

The dialog "Add ID" opens

Call: Click on **+** in the **Configure IDs and entries** dialog

Name	Name of ID, for example tagA
Description	ID description

20.4 Visualization elements

20.4.1 Configuring elements - General information

User input when editing an element

Different user inputs (mouse or menu commands) can be executed in the visualization editor, depending on the cursor position in the editor and the element status. If the cursor position allows user inputs, it changes its form to signal it to the user.

The element status can be "not selected" or "selected". A selected element is additionally visualized with small rectangles, representing its position and size. Used commands have an effect on this element. Commands have an effect on this element.

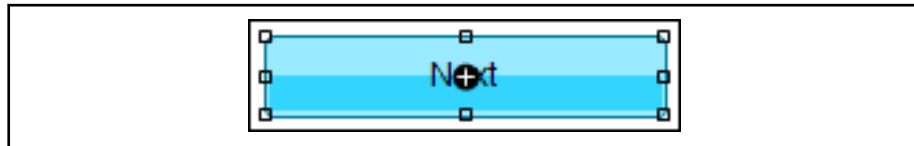


Fig. 20-19: Selected button

Commands

Depending on the element and the selection in the editor, different grayed out commands are available ([chapter 20.2 "Commands" on page 765](#)).

Editing a visualization element

Actions in the editor	Effect on properties
If an element is focused and if the cursor is a hand symbol, select the element by clicking on it. Position and size rectangles of the element are visible on the contour.	The properties of the currently selected element are displayed immediately and can be edited under Properties .
If an element is focuses and if the cursor becomes  , open the line editor by clicking on the element and enter a name/text. The line editor is closed using <Input>.	Text under category Texts contains the entered text.
Focusing on the center of the selected element changes the cursor shape in  and the center can be dragged to any X-/Y-position by keeping the mouse key pressed.	X and Y under the category Center are refreshed.

Actions in the editor	Effect on properties
Focusing on the selected element changes the cursor shape  in and the center can be dragged to any X-/Y-position by keeping the mouse key pressed.	X and Y under the category Position are refreshed.
If the element is selected and if the focus is on one of the small rectangles on the contour, the cursor shape changes and the rectangle can be dragged to a new size. If the size is changed using the mouse, the center is reset. If additionally, the <Shift> key is pressed, the width and height are proportionally changed. If additionally, the <Ctrl> key is pressed, the width and height are changed symmetrically to the center. If additionally, the <Shift> key and the <Ctrl> key are pressed, the width and height are changed symmetrically to the center.	Depending on the element type, its position and size is saved differently under category Position: • X, Y • X, Y, width, height • Category Points [0] .. [<n>]: X, Y

Multi-selection *Several items can be selected if*

- <Shift> is pressed while elements are selected or deselected using the mouse.
- a rectangle is drawn, enclosing several elements (drag-and-drop).
- [chapter 20.2.12 "Command "Select all" on page 774](#) is used.

Deselection is possible with [chapter 20.2.13 "Command "Deselect all" on page 774](#).

Optionally, elements can be selected in the "Element list" tab using <Shift> and <Ctrl> ([chapter 20.3.5 "Tab "Element list" on page 793](#)).



If several elements are selected, the changes made in **Properties** affects all selected elements.

Changing the selection using the keyboard

If an element is selected, the selection of the next element according to the order when inserting (Z-order) can be changed with <Tab>. If <Shift> + <Tab> are pressed, the selection is changed to a reversed order.

Grouping elements

Multiple selected elements can be group using the "Group" command ([chapter 20.2.8 "Command "Group" on page 771](#)).

The selected elements are contained in a group whose subelements can be selected via the "Element list" tab ([chapter 20.3.5 "Tab "Element list" on page 793](#)).

With "Ungrouping objects", the group is broken down again in its elements ([chapter 20.2.9 "Ungroup" command on page 772](#)).

Changing the order

To specify where the elements are to be positioned on the Z-axis of the visualization area, in the foreground, background or in between, use the commands under **VI-Logic visualization ► Order**.

Element alignment

Use the commands under **VI-Logic visualization ► Alignment** to align two or more elements.

Using text and language

The text can be assigned to a visualization element by a definition in the property editor or by direct input in the visualization editor. For the second op-

tion, an input field can be opened at the selected element using the <space bar>.

Also refer to

- [chapter "User input when editing an element" on page 795](#)

Text lists

Text lists are used to manage texts that can be displayed in the visualization. A text list is a POU that contains text strings that can be referenced uniquely within a project by combining a text list name with a list-internal text ID. Different language versions of a text (local country language) can be defined in a text list. Thus, each text is also identified by a language code. At least the default language has to be defined for every text.

Also refer to

- [chapter 18.2.15 "Menu 'Text list'" on page 684](#)



Directories that provide text lists for use in visualizations can be defined under **Tools** ► **Options** ► **IndraLogic 2G** ► **Visualization**, tab "File options".

Text types

Enter the text list name and the text ID (string format) in the text properties of a visualization element in order to define which text is displayed in the element in online mode.

Additionally, the text layout can be changed by formatting ("[Text formatting on page 797](#)").

As specifying a text ID is also possible using a variable, **dynamic switching among texts** becomes possible.

If the currently used text list defines more national language versions of a text, the language used in the visualization can be switched using a configured visualization element (**Language switching**).

Consequently, an input configuration is required that defines a mouse action for "Language switching". ([chapter "Input configuration" on page 800](#)).

This allows dynamic display and language selection for visualization texts.

There are two types of text in a visualization:

- **Static texts:**

A static text cannot be changed in online mode (only the national language is automatically adjusted to the version used in the visualization if the language versions are defined in the text list). Static text is configured in the **Texts** property and is managed in an automatically generated text list, "GlobalTextList".

- **Dynamic texts**

Dynamic text can be modified in online mode by the user input or by a variable of the application. The dynamic text is configured in the **Dynamic texts** property. The name of the dynamic text lists and the text ID for the desired text have to be specified.

Text formatting

In addition to entering the text in the element configuration, formatting specifications can also be added to format the text display in online mode. A formatting specification always begins with a "%" followed by a character that determines the type of formatting. Use formatting by itself or in combination with the actual text.

When inserting a %s in the text string, text is inserted at the position in online mode saved in the variable that is assigned under the **Text variables** property.

If the instance name of a variable transferred to a visualization block as an input parameter should be displayed, use the pragma attribute "parameter-stringof" ([chapter "Attribute 'parameterstringof'" on page 790](#)).



Note that any formatting specification in accordance with the standard C library function "sprintf" can be used if the specification matches the data type of the variable used.

Characters following "%"	Argument/output as
d, i	Decimal number
b	Binary number
o	Unsigned octal number (without a preceding zero)
x	Unsigned hexadecimal number (without a preceding "0x")
u	Unsigned decimal number
c	Single character
s	Character string: In online mode, this is replaced by the value of the variables that is specified in the Text variables ► Text variable properties field.
f	REAL value Syntax: % [<Alignment> <Minimum width> . <Accuracy>] f • <Alignment>: Add - + ; minus indicates a left alignment; plus indicates a right alignment (default). • <Minimum width>: Number of places before the comma • <Accuracy>: Number of places after the comma (default: 6)

Tab. 20-2: Examples of formatting specifications and its arguments:

Example:

Enter in property **Texts** ► **Text**: Fill level %2.5f mm

Enter in property **Text variables** ► **Text variable**: PlcProg.fvar1; variable of type REAL specifies the fill level value.

-> output in online mode:

Fill level 32.89999 mm



If a **percent sign %** is to be displayed together with one of the previously described formatting specifications, "%%" has to be entered.

Example:

Enter "Rate in %%: %s" to display the following in online mode:

"Rate in %: 12" (if the value of the text variable is currently "12").

The font used in online mode can also be used in the property editor. See properties **Text properties** and **Font variables**.

The horizontal and vertical text alignment is defined in the property editor of the elements.

Supporting the Unicode format

Unicode can also be enabled.

Also refer to

- [chapter 18.2.15 "Menu 'Text list'" on page 684](#)

System time output

A combination of "%t" and the following special characters in square brackets is replaced by the actual system time in online mode. The placeholders define the display format, see table.



Importing IndraLogic 1x projects:

The time format %t used in IndraLogic 1x is automatically converted into the new %t [] format when an old project is imported.

The following placeholders are not supported anymore: %U, %W, %z, %Z.

ddd	Name of weekday, abbreviated, e.g. "Wed"
dddd	Name of weekday, e.g. "Wednesday"
dddddd	Week day as number of 1 = Monday to 7 = Sunday
MMM	Name of month, abbreviated, e.g. "Feb"
MMMM	Name of month, e.g. "February"
d	Days in a month as number (1 - 31), e.g. "8"
dd	Days in a month as number (01 - 31), e.g. "08"
M	Months as number (1 - 12), e.g. "4"
MM	Months as number (01 - 12), e.g. "04"
yyy	Days in a year as number (001-366), e.g. "067"
y	Years without century (0-99), e.g. "9"
yy	Years without century (00-99), e.g. "09"
yyy	Year with century indication, e.g. "2009"
HH	Hour, 24 hour format (01-24), e.g. "16"
hh	Hour, 12 hour format (01-12), e.g. "4" for 4 p.m.
m	Minutes (0-59), without preceding zero, e.g. "6"
mm	Minutes (00-59), without preceding zero, e.g. "06"
s	Seconds (0-59) without leading zeros, e.g. "6"
ss	Seconds (00-59), without preceding zero, e.g. "06"
ms	Milliseconds (0-999), without preceding zero, e.g. "322"
t	Identifier for the display in 12 hour format: A (hours <12) or P (hours >12), e.g. "A" if it is 9 a.m.
tt	Identifier for the display in 12 hour format: AM (hours <12) or PM (hours >12), e.g. "AM" if it is 9 a.m.
' '	Character strings that contain one of the placeholders listed above have to be enclosed in single apostrophes. All other texts in the format string can remain without apostrophes: e.g. "update", since it contains a "d" and a "t"

Tab. 20-3: Valid placeholders

Example:

```
%t['Last update:' ddd MMM dd.MM.yy 'at' HH:mm:ss]
```

-> Display in online mode: Last update: Wed Aug 28.08.02 at 16:32:45

Output of a variable of data type TIME, LTIME, DATE, TOD or DT

In the visualization, variables of type **TIME**, **LTIME**, **DATE**, **TOD** or **DT** can also be output. Select a variable of the relevant type in the **Text variable** property. The type is output via the **Text** property. Enter %t or a placeholder in square brackets.

d or dd	days
HH	Hour, 24 hour format (01-23), e.g. "16"
hh	Hour, 12 hour format (01-11), e.g. "4" for 4 p.m.
m or mm	Minutes (0-59), without preceding zero, e.g. "6"
s or ss	Seconds (0-59) without leading zeros, e.g. "6"
ms	Milliseconds (0-999), without preceding zero, e.g. "22"
us	Milliseconds (0-999), without preceding zero, e.g. "22" only for LTIME
ns	Nanoseconds (0-999), without preceding zero, e.g. "22" only for LTIME
' '	Character strings that contain one of the placeholders listed above have to be enclosed in single apostrophes. All other texts in the format string can remain without apostrophes: e.g. "update", since it contains a "d" and a "t"

Tab. 20-4: Valid placeholders

Example:

Declaration in PlcProg: **LTIME1** : **LTIME** :=
LTIME#1000D23H44M12S34MS2US44NS

Property **Text variable**: PlcProg.LTIME1

Property **Text**: Time1: %t[dd.HH.m.s.ms.us.ns]

→ Display in online mode:

Time1: 1000.23.44.12.34.2.44

Input configuration

One or several subsequent actions described in the following can be defined in the **Input configuration** dialog for a certain event (e.g. OnMouseClicked) of the element. When clicking in the value field, the dialog opens the event in the **Property** window. The event name for which these subsequent actions are selected is visible in the gray line as title.

Close dialog

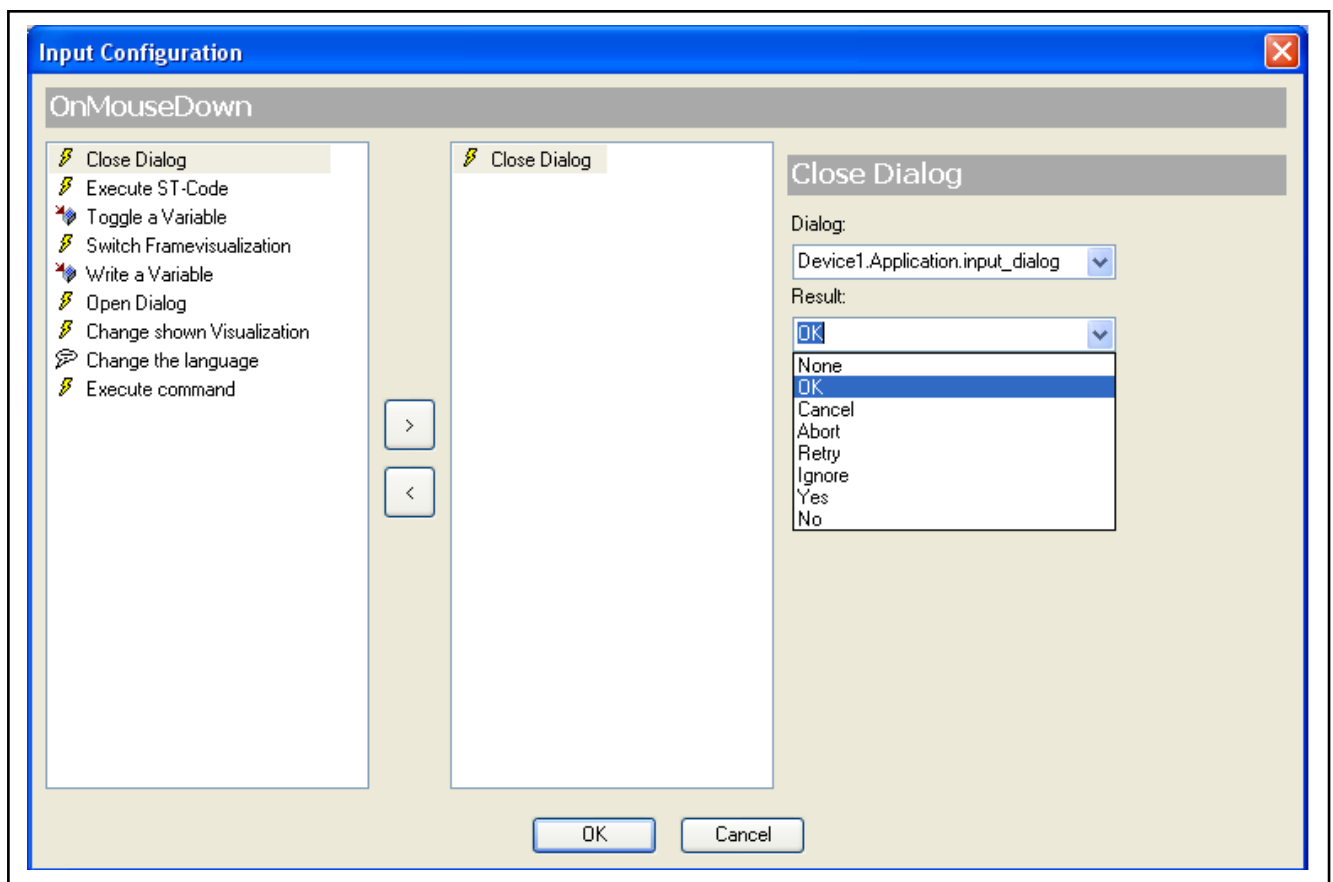


Fig. 20-20: "Input configuration" dialog, "Close dialog" selected

Here, it can be specified that the dialog is to be closed with the result entered when the event occurs. Select the desired dialog from the drop-down list, which provides all currently available [chapter "Input dialogs" on page 1002](#).

Dialog: Select a command from the drop-down list.

Result: The list contains the default options used in dialogs in which user reaction is required: OK, Cancel, Abort, Retry, Ignore, Yes, No.



Note that the result from the latest dialog closed can be called and that a corresponding reaction to it can be configured in any desired element in the same visualization. To do this, use the **On-DialogClosed** configuration option.

See [chapter "Input dialogs" on page 1002](#) for more information about the default and user-defined input dialogs in visualizations.

Open dialog

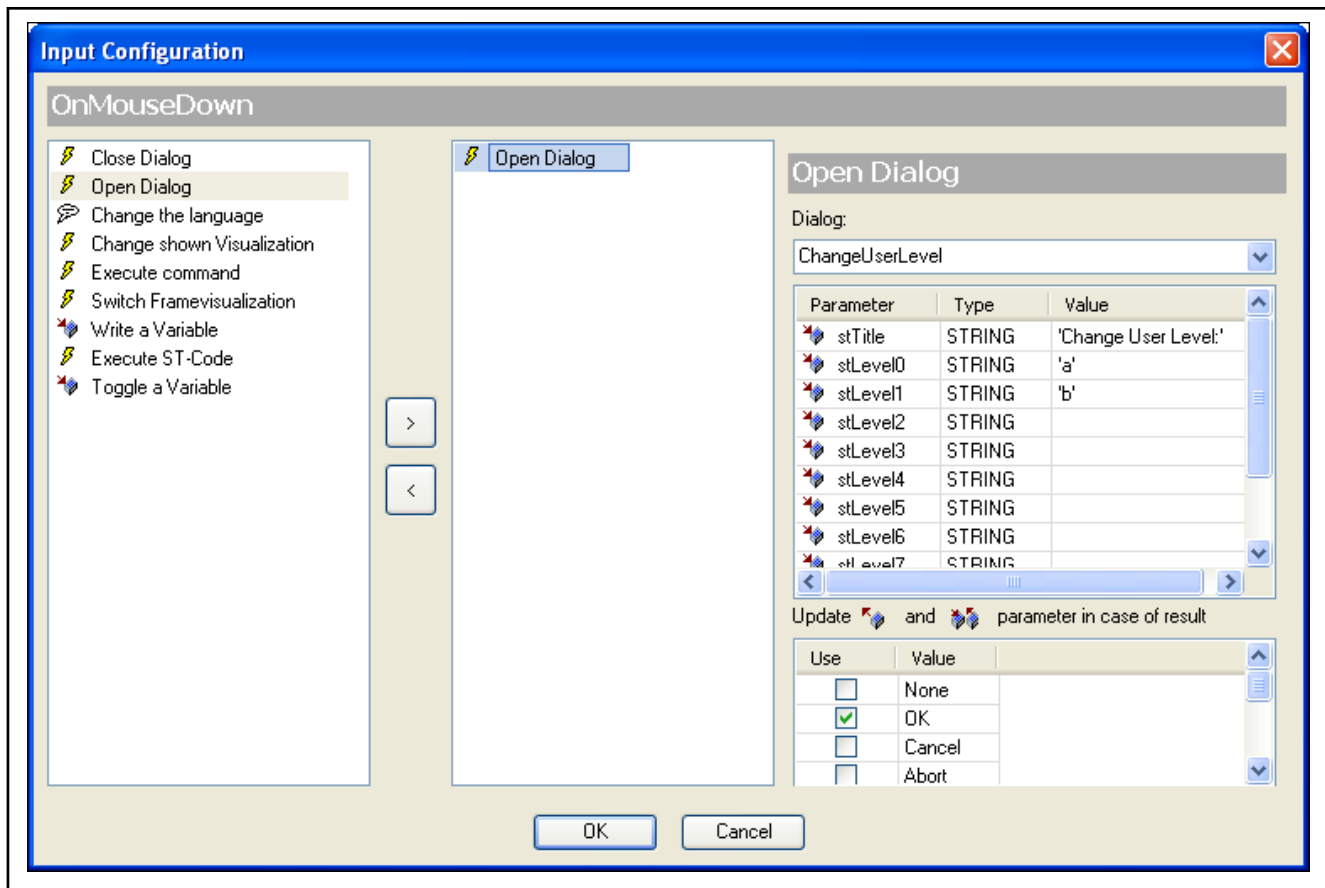


Fig. 20-21: "Input configuration" dialog, "Open dialog" selected

Here, define that a dialog opens as a result of the mouse action. This dialog is represented by another visualization (default or user-created). The drop-down list contains all visualizations in which the purpose "Dialog" is entered in their "Properties" dialog of a visualization object ([chapter 20.6 "Dialog Properties" - "Visualization object" on page 1001](#)). Thus, a customized dialog can be used as a user input mask in a visualization.

The standard objects "VisuDialogs.Login" and "VisuDialogs.FileOpenSave" are available if **VisuDialogs.library** is embedded in the project.

For the selected dialog visualization, the input (VAR_INPUT) and input and output parameters (VAR_IN_OUT) are displayed as defined in the basic settings of the interface editor (**Parameter, Type, Value**) ([chapter "Basic information on the interface editor" on page 782](#)).

The default values (=0) can be replaced by visualization-specific values which are always written to the dialog visualization when the dialog is opened.

Furthermore, it has to be specified upon which **Event** of the dialog visualization the output and input/output parameters have to be written: Tick the box in the **Use** column of the required event **value**. Possible values: None, OK, Cancel, Abort, Retry, Ignore, Yes, No.



For standard use of a dialog visualization, it is not required to define an **OnDialogClosed** action to close the dialog, in contrast to earlier visualization versions. However, the action will still be available for special workarounds.



Please note, that output and input/output parameters of a dialog visualization are not written before the dialog is closed! Until then, the values are stored in the stack. This means that they are not handled as references but as copies.

Language selection

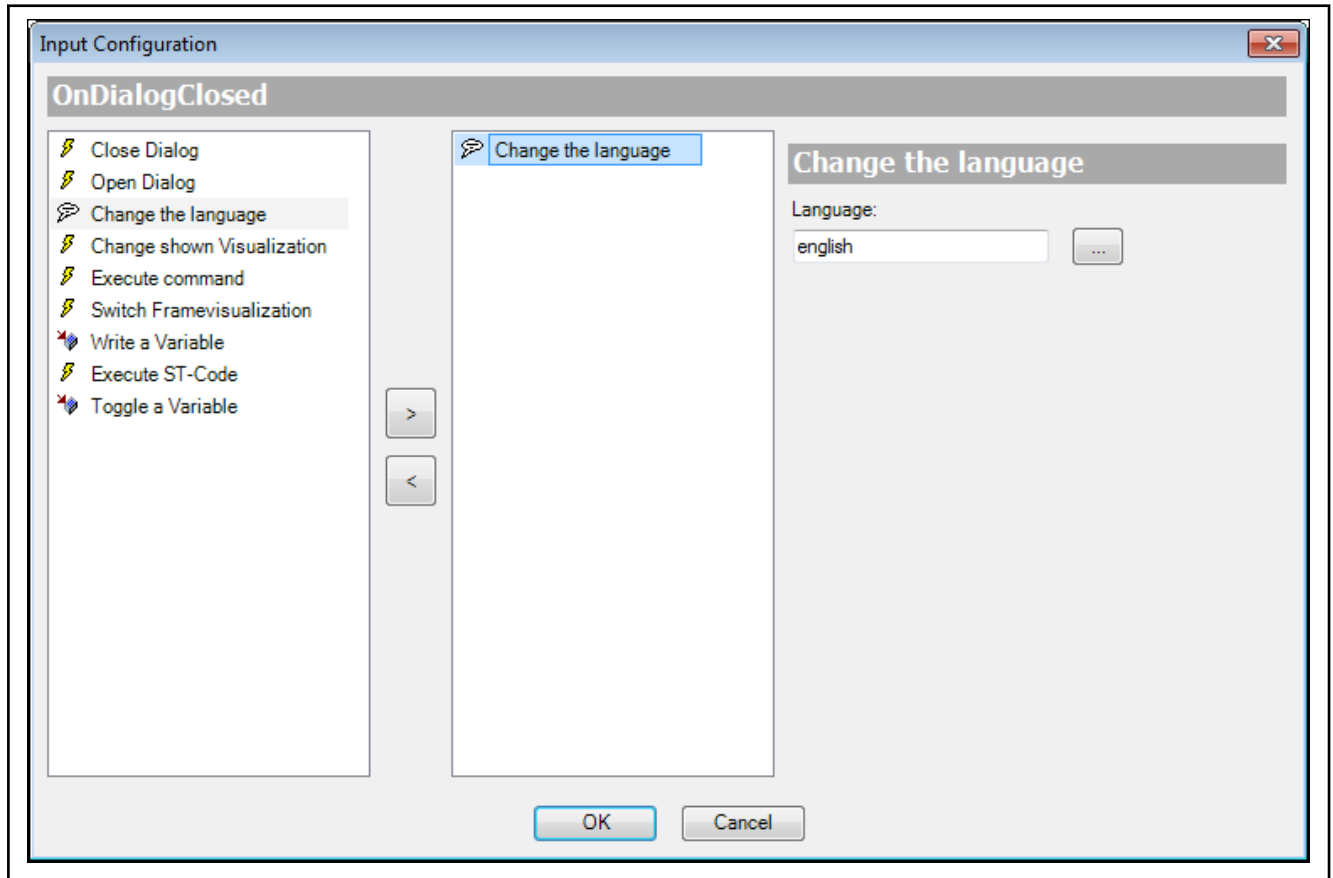


Fig. 20-22: "Input configuration", "Language switching" selected

In the Language field, a local country language can be entered that will be used in the display of visualization text in response to a mouse action. Use the language code from the respective text list(s). Optionally, click on the  key to select the language in the input assistance.

Switch visualization

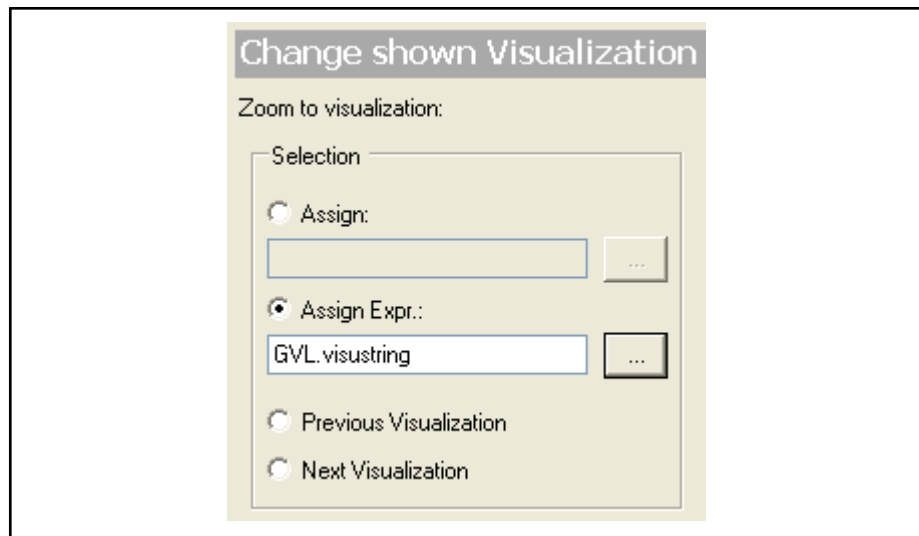


Fig. 20-23: "Input configuration", "Visualization change" is selected

Select one of the following options to specify which visualization is displayed as response to the mouse action in online mode (selection):

Direct assignment: The visualization can be input directly. The entire path must be entered: <Devicename>.<Applicationname>.<Visualizationname>. Use the input assistance opened via the  button. Example: "MyPlc.Appl1.Visu_xy"

Assign using an expression: Here, enter a variable of type STRING that is used by the application and that returns the name of the visualization. Example: "visu_stringvar: STRING := 'MyPlc.Appl1.Visu_xy' "

The sequence in which the visualizations were displayed via user inputs is saved internally. This information can be used with both of the following options:

Previous visualization: The previously shown visualization is displayed again. If no visualization was called previously, the current visualization is displayed.

Next Visualization: The visualization that follows the current one in the recorded sequence of changing visualizations is displayed. This is only possible if a visualization switch has already been made for this visualization using "Previous visualization".

Internal command Executing programs, printing and commands of the recipe manager.

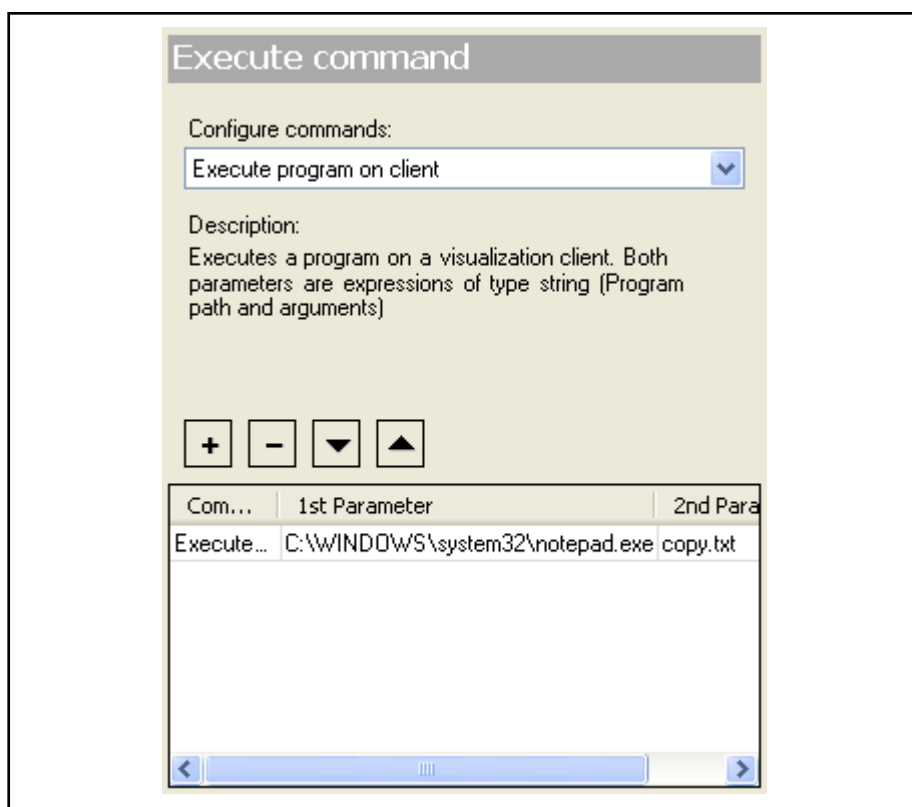


Fig. 20-24: "Input configuration", "Internal command" is selected

Here, define one or multiple commands to be executed as response to the mouse action.

Select a command from the **Configure command** list and press the  button to insert the command into the table in the lower section of the dialog. This table contains all commands selected for the current input configuration.

A brief description of the command selected in the list above is always shown in the center section of the dialog. Based on this description, complete the parameters for the command in the table columns **1st parameter** and **2nd parameter** below. (**Command** column displays the internal command name). See the descriptions of the individual commands below and on the help page.

Use  to remove the currently selected entry from the list. Later, when the user input is carried out in the visualization element, the configured commands are executed according to their arrangement in the table from top to bottom. To change the order, the entries can be moved using the  and  buttons.

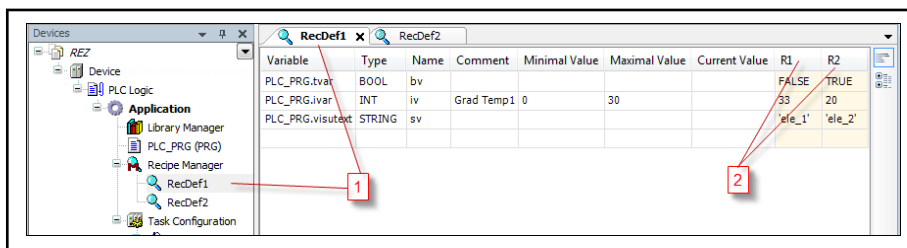
Execute program on the control Execute program on client	The specified program (*.exe) is executed on the control or the display variant. 1st parameter: Character string <code>STRING</code>, program file path (example: "C:\programs\notepad.exe") 2nd parameter: Character string of type <code>STRING</code>, arguments for the program to be executed, e.g. name of a file that the program is to open (example: "copyfile.txt")
Print	The default Print dialog opens, where settings for setting the print area and the printer parameters can be selected. The current visualization can be printed using these settings. This command is only supported within the target visualization under windows.
Navigate to URL (WebVisu)	Requirement: The visualization is executed as VisuDeviceWeb. Upon occurrence of the input event, the visualization navigates to the website of the specified URL. 1st parameter: Web address URL - As a variable type <code>STRING</code> to program the start page. For example: <code>PlcProg.stURL</code> - As a literal in single quotes. For example: <code>http://de.wikipedia.org</code> 2nd Parameter If no parameter is provided, the web page is displayed in a new window or a new tab. If 'replace' is specified, the VisuDeviceWeb is replaced by the web page.

Tab. 20-5: Commands

The following has to be noted for the recipe manager command:

The **recipe definition** is an object under the recipe manager in the device tree of the project. It consists of a list of variables and the existing recipes for these variables (i.e. variable values). The definition can be opened and edited in the editor window of the recipe manager.

The **recipe name** is the name of a recipe described in the recipe definition for the respective list of variables. The recipe names are entered as table headers.



- 1 Name of the recipe definition
2 Recipe name

Fig. 20-25: Recipe definition and recipe name

Create recipe	A new recipe is created in the indicated recipe definition. 1st parameter: Name of the recipe definition 2nd parameter: Name of the new recipe
Read recipe	The actual values of the variables in the indicated recipe definition are read by the control and written into the indicated recipe. During this procedure, the values are saved implicitly (in a file on the control) while they are displayed in the recipe definition table in the IndraLogic recipe manager. In other words: The recipe managed in IndraLogic is updated with the values from the control. 1st parameter: Name of the recipe definition 2nd parameter: Name of the recipe for which the values are read from the control and are to be saved as new recipe values.
Write recipe	The values of the indicated recipe - as saved in the recipe manager - are written to the corresponding variables one the control. 1st parameter: Name of the recipe definition 2nd parameter: Name of the recipe from this recipe definition whose values are to be written to the variables on the control.
Save recipe	The values of the indicated recipe are saved in a file with the extension *.txtrecipe. The file name has to be defined. To do this, the default dialog to save a file opens. Caution: The recipe files that are used implicitly for clipboard functions during reading and writing must not be overwritten, i.e. the name for the new file has to have a different name than "<Recipe name>.<Recipe definition name>.txtrecipe"! 1st parameter: Name of the recipe definition 2nd parameter: Name of the recipe to be saved in the specified file.
Load recipe	The recipe saved in a file (see above "Save recipe" above) can be loaded from this file. The default file selection dialog opens to select the file. The filter is automatically set to include the file extension "*.txtrecipe". After loading, the display of the respective recipe is updated in the IndraLogic recipe manager. 1st parameter: Name of the recipe definition 2nd parameter: Name of the recipe for which the values from the specified file are to be reloaded to the recipe manager.
Delete recipe	The specified recipe is deleted from the specified recipe definition. 1st parameter: Name of the recipe definition 2nd parameter: Name of the recipe

Switch frame visualization

Each input with the mouse that has been assigned an action in the input configuration dialog displays the property "Configured" in the property window.

Prerequisites: In a visualization in the project, there are frame elements that can be assigned different visualizations via the command "Frame selection" and the dialog **Configuration of frame visualizations** ([chapter 20.2.10 "Frame selection" commands](#) on page 772). These visualizations are assigned an index 0, 1, 2, etc. within the frame and by default, the first of the assigned visualizations (index "0") is displayed online in the frame.

Switch frame visualization: In this dialog, when the mouse action is executed on the current visualization element in a specific frame element, a specific visualization assigned to the frame element is displayed can be configured. In this way, with the current element, you can program a switch for displaying visualizations in a frame.

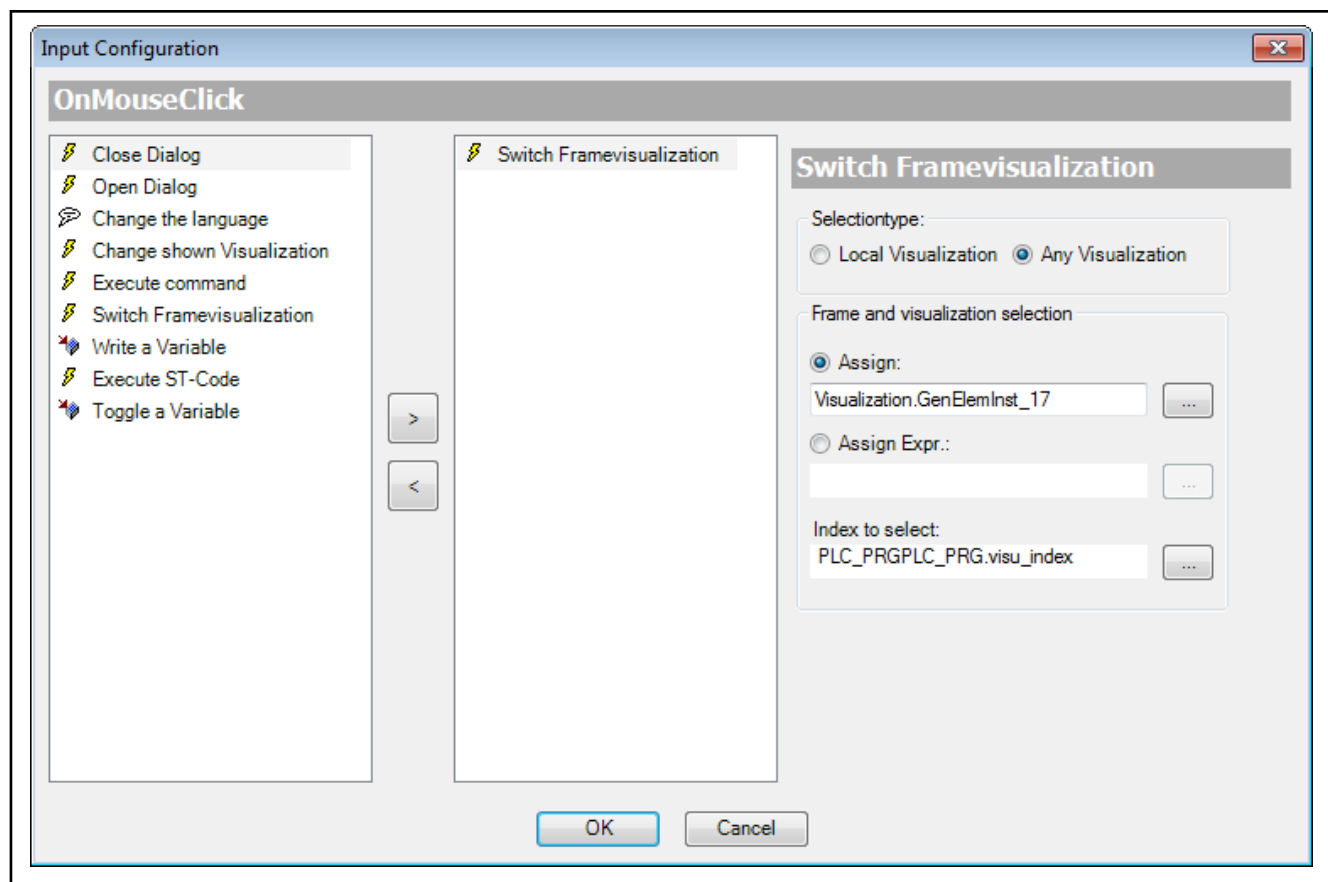


Fig. 20-26: "Input configuration", "Switch frame visualization" is selected

Selection type: Selection of the affected frame element...

- can be limited to the current visualization, which means that the configuration dialog will be simpler - but will not have dynamic configuration options, this means -> Option **Local visualization**.
- can be extended for all visualizations available in the project that allow a dynamic definition of the frame and the displayed visualization -> Option **Any visualization**.

Local visualization: Only frames in the current visualization can be addressed and are available in the "Frame selection" field (see below). This is a simple dialog for quick, direct configuration in the local visualization. However, it does not provide the option to enter the desired visualization using project variables. If this is necessary, select the "Any visualization" option.

The **Frame selection** field displays the local frame elements with the respective assigned visualizations indented beneath.

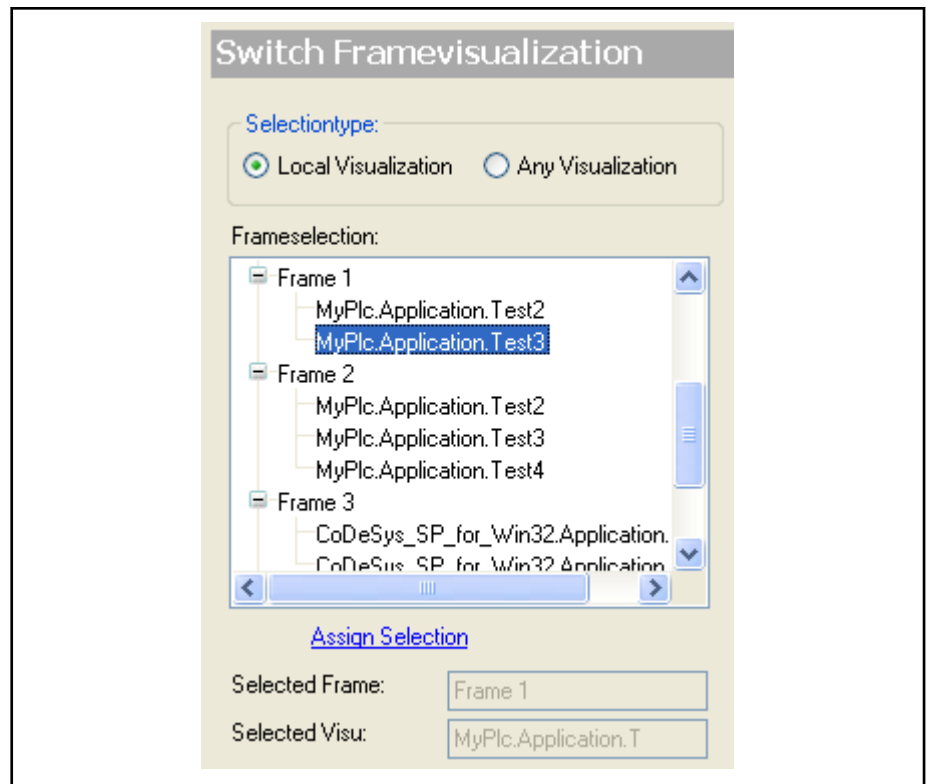


Fig. 20-27: Frame selection, local

Below the respective frame, select the desired visualization to be displayed in response to the mouse action. Click on Assign selection to save the setting. The current selection will then displayed in the "Frame Selection" and "Visualization Selection" fields.

Any visualization: All frames in the project and their assigned visualizations can be selected. In this case, frame and visualization can also be specified via project variables or dynamically. The "Frame and visualization selection" is executed via the following fields and options:

The desired frame element can be entered directly or by using an expression:

Direct assignment: If this option is enabled, the complete path of the respective frame element has to be entered directly. Via the button, the input assistance can be used. Syntax: <DeviceName>.<ApplicationName>.<VisualizationName>.<FrameElementName>. See the example in the following figure.

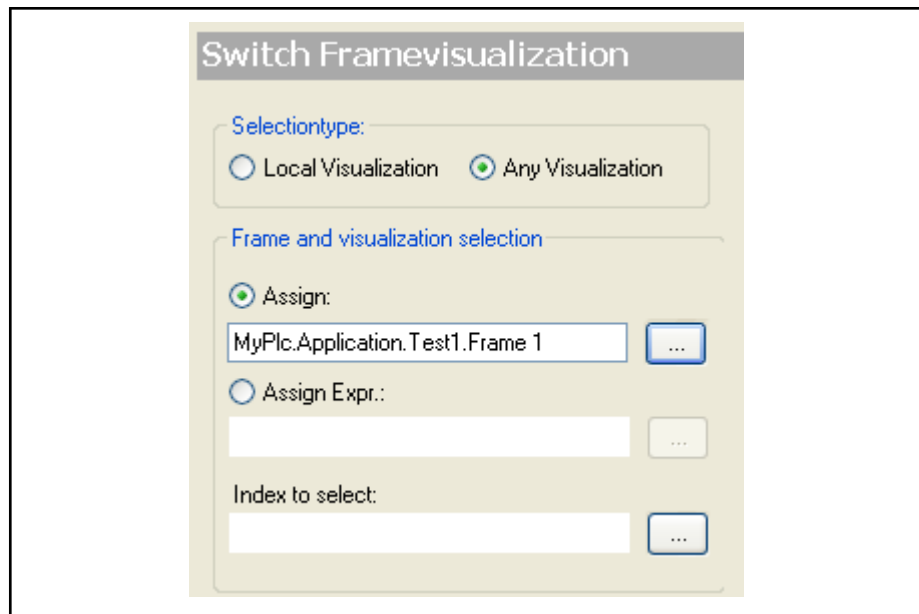


Fig. 20-28: Frame selection, direct

Assign using an expression: Alternatively, this option can be enabled to use a project variable of type STRING to enter the frame element. Via the  button, the input assistance can be opened. The variable has to return the entire path of the frame element. Syntax: <Device-Name>.<ApplicationName>.<VisualizationName>.<FrameElementName>.

Example: frame_var: STRING:='MyPlc.Application.Test1.Frame1';

The desired visualization to be displayed in the selected frame in response to the mouse action is entered using an index. This index numbers the visualizations with whole numbers, starting with 0. The numbered visualizations are those that were assigned to a frame in the "Configuring the frame visualizations" dialog. This way, the first visualization is always shown in this list by default.

Index to select: Enter the index of the desired visualization directly or via a project variable used in the application. The  button opens the input assistance to select a variable.

Write variable

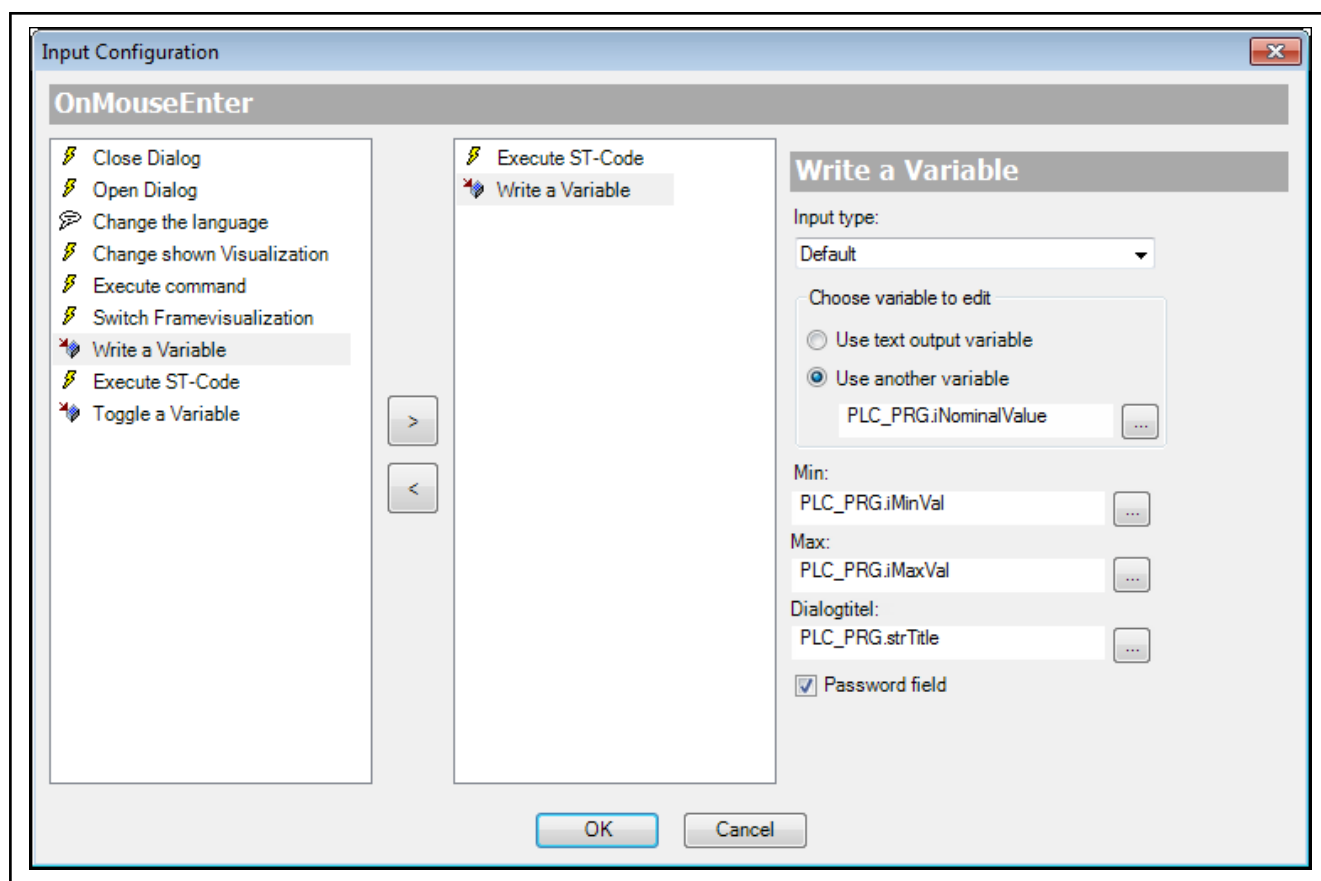


Fig. 20-29: "Input configuration", "write variable"

If there is an input configuration of the "Write variable" type for a visualization element, the element provides the option to enter a value when the corresponding mouse action is executed. The value can then be entered as a character string or using a numpad or keypad and after the entry is completed, it is written on the project variable specified in the "Input Configuration" dialog. The value is interpreted as text or numerical value based on the data type of the project variable.

Using the configuration dialog:

Chose one of the following input types from the selection list in the right section of the dialog:

- **Default:** The default input type is used. Specify the default in the visualization manager settings.
- **Text input:** A field for value/text input opens. To enter values, a keyboard has to be available
- **Text input with limits:** A field for value/text input is opened. Additionally, the input limits are displayed. These limits are highlighted in red in case they are fallen below or exceeded. To enter values, a keyboard has to be available
- **VisuDialogs.Keypad:** The mouse action opens a simulated keypad (keyboard). Click on the corresponding keypads to enter a character string.
- **VisuDialogs.Numpad:** The mouse action opens a simulated numpad (numerical keypad). Click on the respective keys to enter a numerical character string.

- In addition to the default input types, all visualizations that were defined as “Keypad/keyboard/input configuration dialog” (see [chapter 20.6 "Dialog "Properties" - "Visualization object" on page 1001](#)) in their properties are listed.

Select variable to be edited

- **Use text output variable:** The value is written to the variable that is specified as text variable in the visualization element.
- **Use another variable:** Project variable specification. Use  to open the input assistance.

Define an upper or lower limit for a value to be written to the specified variable. Either directly enter values in the **Min.** and **Max.** fields or use project variables.

Dialog title: Select a text or a text variable for the dialog title bar.

If you would like to display your input hidden by asterisks (*****), tick the checkbox in front of the **password field**.

Example:

See the example in the dialog figure above and assume that it is the configuration dialog for a rectangle element. If the mouse button is pressed on this rectangle in online mode, a keypad with the title provided by the `PlcProg.strTitle` variable is displayed.

By clicking on 1, 2 and 3 you can enter the value “123”, for example. This value is displayed in the upper section of the dialog along with the minimum and maximum values for the input that was entered with `PlcProg.iMinVal` and `PlcProg.iMaxVal`. As soon as the entry is confirmed by clicking on the OK key, “123” is written on the variable `PlcProg.iNominalValue`. If `iNominalValue` is defined as STRING variable, it is assigned the value '123', if `iNominalValue` is a numerical variable, it is assigned the value 123.

Execute ST code

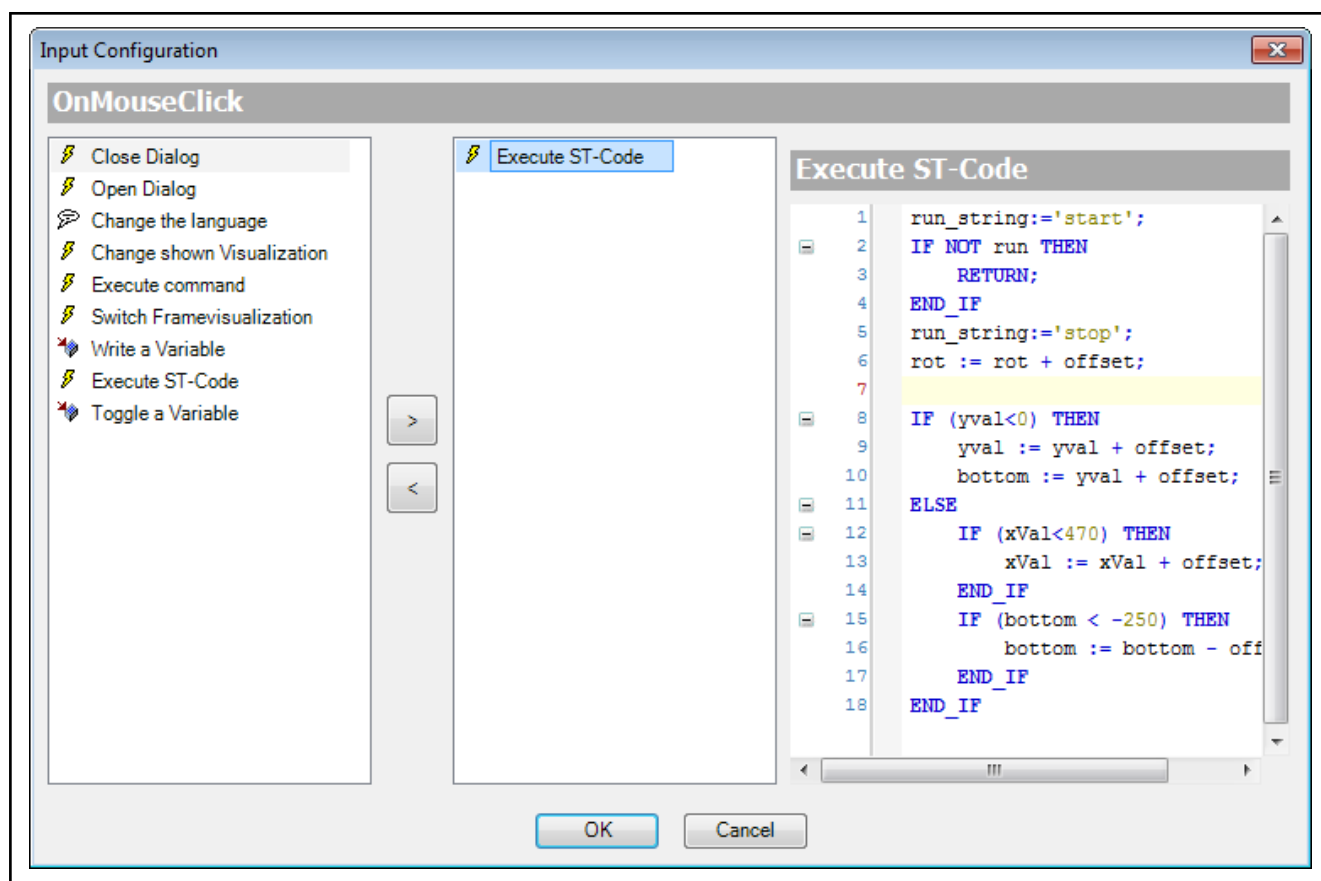


Fig. 20-30: "Input configuration", "Execute ST code" is selected

In this input field, enter code in structured text that is to be executed in response to the mouse action.

Toggle variable

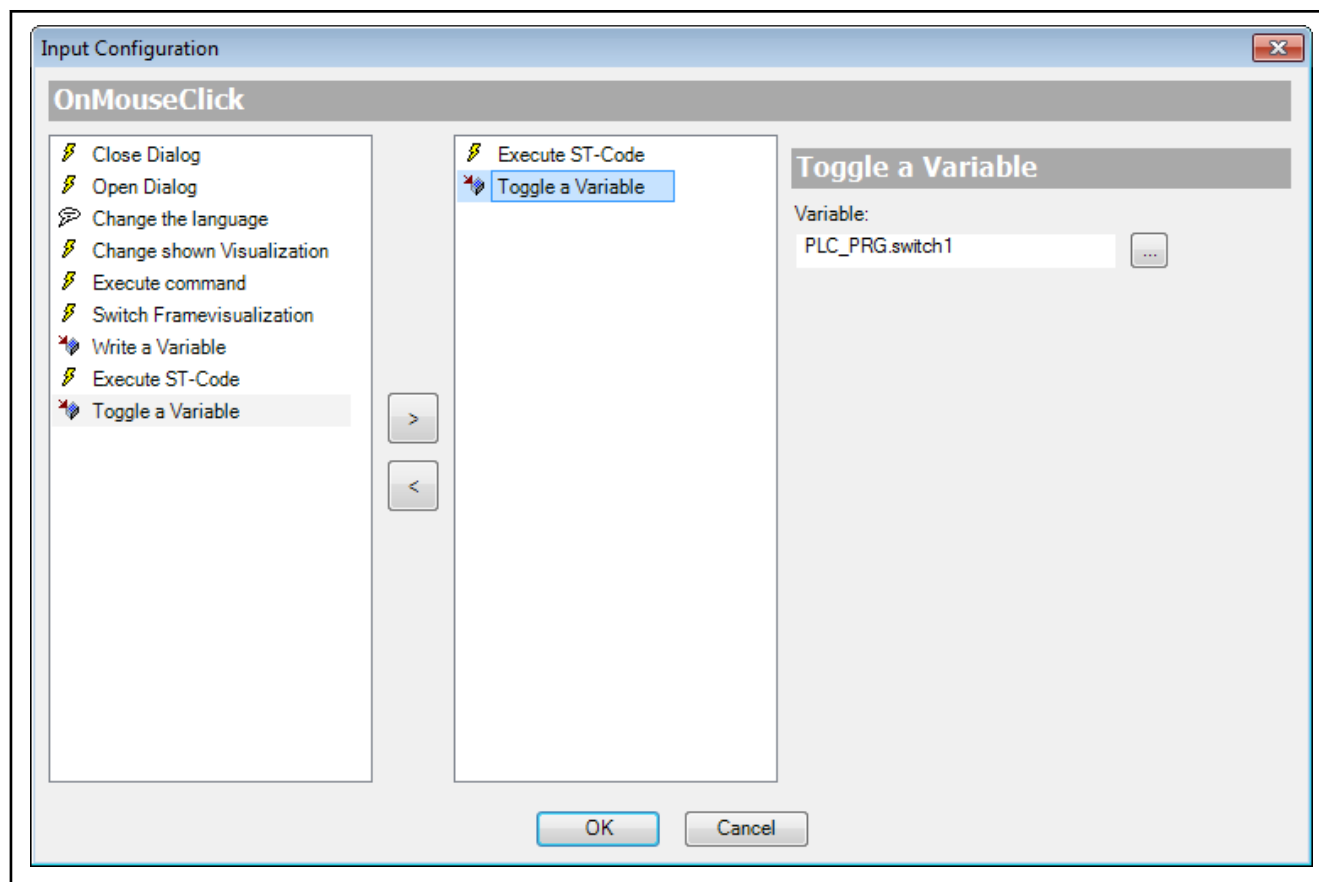


Fig. 20-31: "Input configuration", "Toggle a variable" is selected

Here, enter a Boolean variable that should switch between TRUE and FALSE at repeated mouse actions. Example: `PlcProg.switch1` Click on  to open the input assistance.

Gradient editor

Click on the **Gradient setting** value field in the **Properties** windows to open the **Gradient editor**.

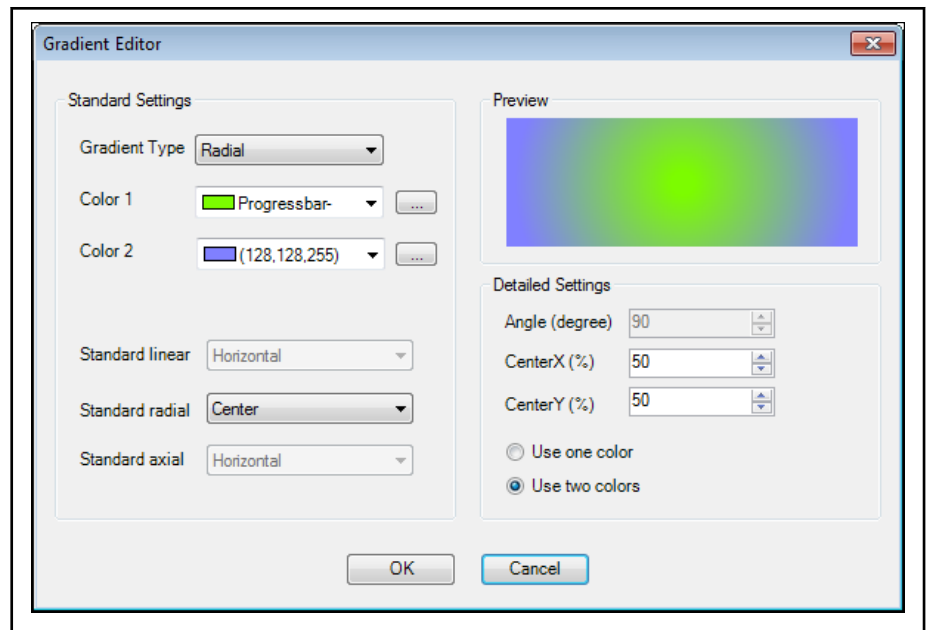


Fig. 20-32: Gradient editor

Standard settings**Gradient type:** Linear, radial, axial.**Color 1:** First color of the gradient. Select it from a combo box or a color dialog. Select  to open it.**Color 2:** Second color of the gradient. Select it from a combo box or a color dialog. Select  to open it.**Standard linear:** Predefined settings of the gradient type **Linear** (angle of the gradient): **Horizontal**, **Vertical**, **From top left**, **From top right**, **Horizontal color toggled**, **Vertical color toggled**, **From bottom left**, **From bottom right**.**Standard radial:** Predefined settings of the gradient type **Radial** (Position of the center): **Center**, **Upper left**, **Upper right**, **Lower left**, **Lower right**.**Standard axial:** Predefined settings of the gradient type **Axial** (angle of the gradient): **Horizontal**, **Vertical**, **From top right**, **From top left**.**Detailed settings****Angle (degree):** Only for the gradient types **Linear** and **Axial**.**Center X (%):** Only for the gradient type **Radial**. X-position of the center point (0 - 100%)**Center Y (%):** Only for the gradient type **Radial**. Y-position of the center point (0 - 100%)**Use one color:** Gradient type between **Color 1** and the same color with a different brightness. The **brightness** can be set between 0 (to) to 100 (white).**Use two colors:** Color gradient between the two selected colors **Color 1** and **Color 2**.**Using semitransparent colors**

You can provide a visualization element with a semitransparent color by setting the element property **Transparency** that is a subordinate property to a color or a font color.

A transparency value can also be defined in the visualization style.

The background color of a visualization is always 100% opaque.



Configuring semitransparent colors is possible as of compiler version 3.5.6.0.

The display variants behave as follows with respect to semitransparent visualization:

- The visualization integrated in IndraLogic always displays semitransparency

Also refer to

- [chapter "Using semitransparent colors" on page 815](#)
- [chapter 20.5.2 "Clients and client editors" on page 999](#)

Semitransparency depending on the element type

The configurable color for elements from the **Common Controls** category can be used only in partial areas, but also in several layers. The consequence is that the background does not show through despite semitransparent colors.

Semitransparent element type

- Everything in the **Basic** category, except
 - **Image**
 - **Frame**
- Everything in the **Common Controls** category, except
 - **Checkbox**
 - **Combo box**
 - **Progress bar**
 - **Radio button**
 - **Tab control element**
 - **Slider**
 - **SpinControl**
- Everything in the category **Date/time managing controls**

Opaque element type

- All of the **Measurement controls** category
- All of the **Lamps/switches/images** category
- All of the **Special controls** category

Activating semitransparency

Prerequisite: One of the display variants configured in the visualization manager can represent a semitransparent visualization.

Example: The PLC does not support semitransparency and the configured display variant is exclusively the integrated IndraLogic visualization.

IndraLogic is open with a project containing a visualization.

1. Double-click on the **Visualization manager** object

The editor  **Visualization Manager** is opened.

2. Click the **Settings** tab.
3. In **Additional settings**, select the **Activate semi-transparent drawing** checkbox.

The visualization editor displays transparent elements that have a transparency value set as the color setting in their properties.

Also refer to

- [chapter ""Visualization manager" - "Settings"" on page 996](#)

Assigning a value for transparency

Prerequisite: IndraLogic is open with a project containing a visualization.

1. Double-click on the visualization.
2. Click on **View ► Element properties**.
3. Select a visualization element.
4. Click the category **Colors** in the window **Properties**. Expand the color settings until the property **Transparent** is visible. Click into the field and set the value using the slider.

The integrated visualization displays the element as semitransparent. When running the application, the visualization is displayed as semi-transparent if the display variant is configured for it.

5. Click the category **Colors** in the window **Properties**. Expand the settings until the property **Transparent** is visible. Click into the field and enter a value between 0 and 255.

Multiplying an element

You can configure an visualization element to be a template suitable for multiplying. When multiplying, a layout develops that is either tabular, line by line, or column by column.

To configure an element as a template:

- Configure a unique element using an array variable with the index access placeholder. Specifically, this is an element-independent property, such as a text field that displays a different text for each multiplied element.
- Assign a fixed value to a property that is the same for all elements. For example, assign the frame color black if all multiplied elements should have a black frame.

For a placeholder array, you can use `$FIRSTDIM$` in the first dimension and `$SECONDDIM$` in the second dimension. In some cases, you can define an array provided with more dimensions than you indexed using placeholders.

For example: `Main.strArr2Dim[1..10, $FIRSTDIM$, $SECONDDIM$, 1..10]`

Furthermore, an element can be configured with any number of array which have, in the simplest case, the same indexing. In special cases, you must consider which dimension should be indexed by placeholders.

When you edit the element properties, you can insert placeholders via smart coding.

IndraLogic can detect the element as a template and execute the command **Multiply visu element**. When configuring the command, you can determine the amount and layout of elements to inserted. In addition, you can configure the replacement of placeholders.

IndraLogic inserts elements, which originate from the template element, in a tabular layout. The arrays configured in **Properties** receive concrete indices instead of placeholders.

Elements you can multiply:

- Rectangle
- Rounded rectangle
- Ellipse
- Line

- Polygon
- Polyline
- Bézier curve
- Image
- Frame
- Button
- Circle segment
- SpinControl
- Text field
- Checkbox
- Image switcher
- Lamp
- Lever button
- Energy button
- Pushbutton
- LED pushbutton
- Toggle switch
- Rotary switch

First steps *You opened a visualization in IndraLogic.*

1. Drag a rectangle to the visualization.
2. Configure the frame color red and the filling color gray.

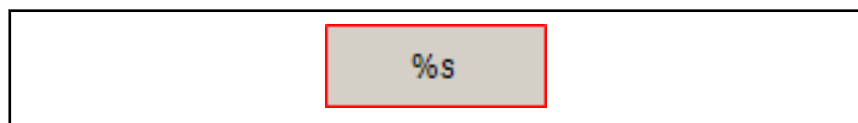


Fig. 20-33:

3. Declare arrays in the IEC code to configure the properties for the output text, tooltip text, and input configuration independent of the element.

```

PROGRAM PLC_PRG
VAR
    strArr2Dim : ARRAY [1..5, 2..10] OF STRING;
    strArr2DimIt : ARRAY [1..5, 2..10] OF STRING;
    boolArr2Dim : ARRAY [1..5, 2..10] OF BOOL;
END_VAR

```

Fig. 20-34:

4. Configure the template element with an array by entering the array name in **Text variables**. Replace the first dimension with the \$FIRSTDIM\$ and the second dimension with the \$SECONDDIM\$ placeholder. For example: PlcProg.strArr2Dim[\$FIRSTDIM\$, \$SECONDDIM\$].

Configure the property **Text** with %s.

The output text of the template element is configured with an array. At runtime, the visualization generates text depending on the index.

5. Configure the template element with an array by entering an array name in **Input configuration-OnMouseClicked**. For example: `PlcProg.boo-1Arr2Dim[$FIRSTDIM$, $SECONDDIM$]`

The input configuration of the rectangle is now configured. At runtime, the visualization will process each rectangle when the mouse button is clicked.

6. Click on **Visualization ► Multiply visu element**.

The **Multiply visu element** dialog opens.

7. Configure the multiply and click **OK** to close the dialog.

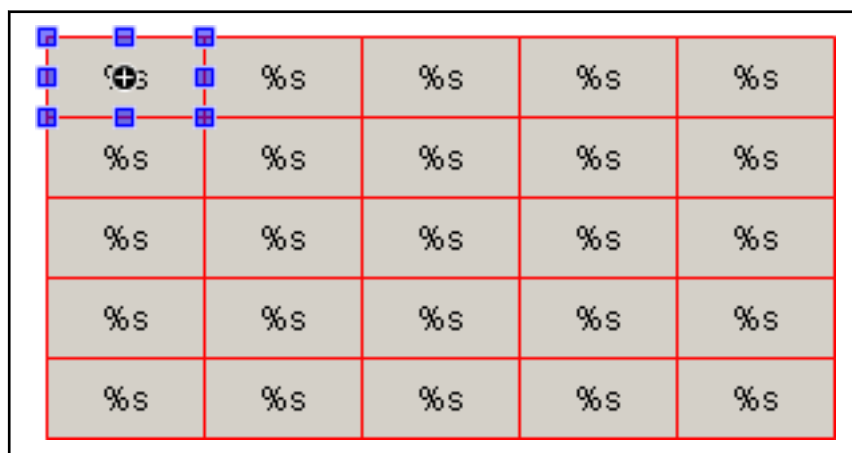


Fig. 20-35:

Also refer to

- [chapter 20.2.17 "Multiply visualization element" command](#) on page 775
- [chapter "Input configuration"](#) on page 800
- [chapter "Options" dialog - "Smart coding"](#) on page 730

Dragging an IEC variable to an element

You can select a variable from the place it was declared or used and then drag it to a visualization. If you drag the variable to an element, then configure an element property with the variable.



You drag a variable to an editor by first dragging the variable to the associated object in the device tree or project tree.

As an alternative, you can also first drag the variable to the editor tab if it is opened in the background. Then activates the editor in the main window.

Elements that support drag-and-drop operations

These elements are configured in a predetermined property with the variables.

Configure as output text in the property **Text variable** for the following elements:

- **Rectangle**
- **Line**
- **Polygon**
- **Circle segment**

- Button
- Image
- Text field

If the property **Text** is not configured, IndraLogic also inserts the formatted text %s.

*Configure as the property **Variable** for the following elements:*

- Image switcher
- Checkbox
- Combo box
- Progress bar
- Lamp
- Switch
- SpinControl
- Slider
- Radio button

*Configure as output text in the property **Value** for the following elements:*

- Bar graph
- Potentiometer
- Scroll bar
- Pointer instrument

*Configure as array in the property **Data array** for the following elements:*

- Table
- Histogram

Configure as trace variable in the property **Trace**. In this way, you can drag any number of variables to the element **Trace**.

Also refer to

- [chapter "Trace" element" on page 972](#)

Elements that do not support drag-and-drop operations

- Alarm banner
- Alarm table
- Date range picker
- Frame
- Group box
- Invisible input
- Label
- Legend
- Tab control element
- Text field
- Time picker
- Busy symbol Flower

First steps

1. Select a variable. For example: `strText`.
2. Drag the variable over a visualization element, for example a rectangle. The cursor is marked with a plus sign and the element is selected.

3. Release the left mouse button.

IndraLogic configures the element with these variables. In **text variable**, `PlcProg.strText` and `%s` is inserted in **Text**.

20.4.2 'Basis'

Element "Rectangle", "Rounded rectangle" and "Ellipse"

Rectangle, rounded rectangle und ellipse are of the same element type. They can only be converted by changing the **Element type** property.

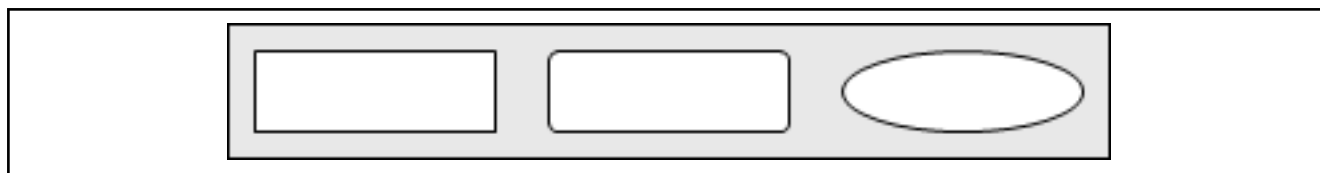


Fig. 20-36: Rectangle, rounded rectangle and ellipse

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

IndraLogic 2G visualization

Element type	• Rectangle • Rounded rectangle • Ellipse

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Normal state • Frame color • Fill color	Select a frame and fill color for normal state. In case that the variable is defined in Color variables ▶ Toggle color is FALSE, the element is in normal state.
Alarm state • Frame color • Fill color	Select a frame and fill color with which the element is displayed in alarm condition. This is the case if the variable in Color variables ▶ Toggle color is TRUE.

Use color gradient	By default, the checkbox is disabled. If it is enabled, the element is drawn with color gradient.
Gradient setting	Dialog opens. chapter "Gradient editor" on page 814

Appearance

Property	Value description
Line width	Weight of frame line in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is required, set Line type to <i>invisible</i> .
Fill type	Select one of the fill types used in Fill color • <i>Filled</i>: The fill color is visible • <i>Invisible</i>: The fill color is invisible
Line style	<i>Select one of the line types used in the contour:</i> • <i>Solid</i> • <i>Dashes</i> • <i>Points</i> • <i>Dash dot</i> • <i>Dash dot dot</i> • <i>Invisible</i>: Contour invisible.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties

These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • <i>Left</i> • <i>Centered</i> • <i>Right</i>
Orientation vertical	<i>Define the vertical text alignment:</i> • <i>Top</i> • <i>Centered</i> • <i>Down</i>

Property	Value description
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Absolute motion"

The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise In contrast to Rotation, the element actually rotates.

Element property 'Relative movement'

The element can be displayed in relation to its fixed position defined in **Position**. The top left and bottom right corner of the element are displaced by the values (pixel) specified in the integer variable in x- or y-direction. In contrast to absolute motion, a relative position is defined, i.e. the distance to the original position. The shape of the element can thus be modified.

See **Absolute motion** for displacing the entire element.

Movement top-left	
X	Integer variable whose value specifies the number of pixels by which the top left corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the top left corner is moved in y-direction.
Movement bottom-right	

X	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in y-direction.

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ▶ Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ▶ Tooltip has to contain %s.

Dynamic texts

These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. <code>'M1'</code> or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. <code>'T1'</code> or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables'

Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ▶ Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>

Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables"

The element property facilitates dynamic switching of the element colors using IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Color variable

`dwFillColor := 16#008FE03F;`

Coding: Red component: `8F`, green component: `E0`, blue component: `3F`

- Grey: `dwGrey := 16#00888888`
 - Red: `dwRed := 16#00FF0000`
 - Black: `dwBlack := 16#00000000`
-

Toggle color	<i>Control variable</i> - IEC variable of type BOOL, e.g. <code>prgA.bToggleColor</code> At runtime, the visualization dynamically controls the color display of the element TRUE: Element in Normal state, e.g. in gray with a black frame FALSE: Element in Alarm condition, e.g. in yellow with red frame. - or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Normal state	Color variables of element during smooth operation
Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColor</code>
Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColor</code>
Alarm state	Color variables of element during an alarm
Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColorAlarm</code>
Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColorAlarm</code>

Appearance variables

To be used in a **dynamic** definition of contour and element fill color appearance. Static definitions are specified in **Appearance**.

Property	Value description
Line width	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight .
Fill type	Input of a DWORD variables that defines the fill type. The color specified in Color variables can be displayed or ignored: - Variable value = 0: Filled - Variable value > 0 : Not visible, i.e. filling is visible This overwrites the fixed value specified in Appearance ► Fill type.
Line style	Entering a DWORD variable specifying the outer line contour. <i>The following values correspond to the following line types:</i> 0: Solid 1: Dash 2: Dot 3: Dash Dot 4: Dash Dot Dot 8: Hidden. The element is shown without an outer line.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration' Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- **OnDialogClosed**
- **OnMouseClicked**
- **OnMouseDown**
- **OnMouseEnter**
- **OnMouseLeave**
- **OnMouseMove**
- **OnMouseUp**

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

'Line' element

This element is a simple dash.

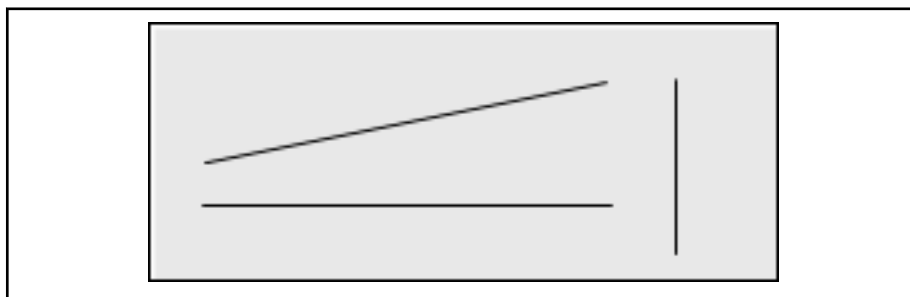


Fig. 20-37: Line

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.
[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Line

Element property "Position" One line is defined with two points: Points [0] to [1].

Points • [<n>] n of {0, 1} – X – Y	• [<n>] – X: Horizontal position in pixels. – Y: Vertical position in pixels.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Color	Select a color to fill the element in normal state. This is the case if the variable in Color variables ▶ Toggle color is FALSE. By editing the value, the element changes color.
Alarm color	Select a color to fill the element in alarm condition. This is the case if the variable in Color variables ▶ Toggle color is TRUE.

Look and feel

Property	Value description
Line width	Frame line weight in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is to be used, set line type to <i>invisible</i> .
Line style	<i>Select one of the line types used in the contour:</i> • Solid • Dashes • Points • Dash dot • Dash dot dot • Invisible: The contour is invisible.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Tooltip variable .

Text properties

These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Absolute motion"

The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise In contrast to Rotation, the element actually rotates.

Element property 'Relative movement'

The element can be displayed in relation to its fixed position defined in **Position**.

Motion of point [<n>] with n of {0, 1} • X • Y	• X: Enter a variable whose value specifies the number of pixels by which point[0]/point[1] is displaced in x-direction • Y: Enter a variable whose value specifies the number of pixels by which point[0]/point[1] is displaced in y-direction

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ▶ Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ▶ Tooltip has to contain %s.

Dynamic texts

These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. 'M1' or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. 'T1' or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables' Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ► Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>
Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code> . At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables" The property facilitates dynamic toggling of colors by means of IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Example of color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: 8F, green component: E0, blue component: 3F

- Grey: dwGrey := 16#00888888
- Red: dwRed := 16#00FF0000
- Black: dwBlack := 16#00000000

Toggle color	<i>Control variable</i> • as IEC variable of type BOOL, e.g. prgA.bToggleColor At runtime, the visualization dynamically controls the color display of the element – TRUE: Element is displayed in color, e.g. grayed out – FALSE: Element is displayed in alarm color, e.g. red • or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Color	IEC variable of type DWORD for the color, e.g. prgA.dwColor
Alarm color	IEC variable of type DWORD for the alarm color, e.g. prgA.dwAlarmColor

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Line weight variable" element property

Dynamic definition of the line element strength using the variable

Integer value	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight . Take into consideration that 0 codes the same as 1 and the line weight is set to 1 pixel.

Element properties "Line type variable"

Dynamic definition of the line type using the variable

Integer value	Variable input with integer data type. The value defines the line type. <i>The following variable values are supported:</i> • Solid • Dashes • Points • Dash dot • Dash dot dot • Invisible: Then the line is invisible. This overwrites the fixed value specified in Appearance ► Line type.

Element property 'Input configuration' Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- **OnDialogClosed**
- **OnMouseClicked**
- **OnMouseDown**
- **OnMouseEnter**
- **OnMouseLeave**
- **OnMouseMove**
- **OnMouseUp**

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

Element 'Polygon', 'Polyline', 'Bézier curve'

Polygon, polyline and Bézier have the same element type. They can only be converted by changing the **Element type** property.

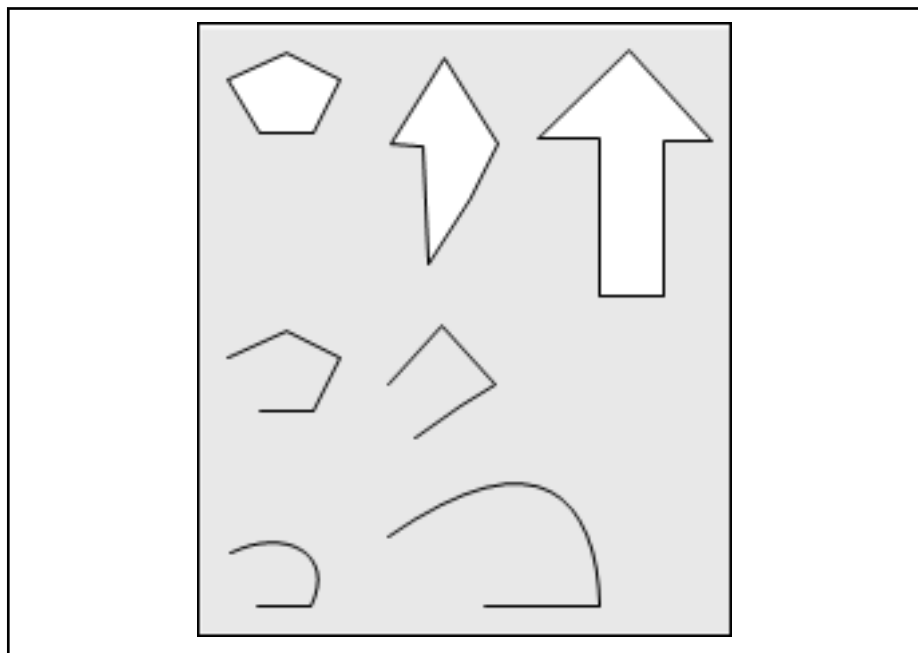


Fig. 20-38: Polygon, polyline, Bézier curve

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input:

- The element can be moved to the editor using drag&drop. The element is drawn with five points: Points [0] to [4]. Additional points can be inserted as follows: Select the element and set the focus on a corner point



(small square on the contour). The cursor form changes . Keep <Ctrl> pressed while clicking on the focuses point, another point is generated. Delete a point by keeping <Shift> + <Ctrl> pressed while clicking on the focuses point.

- Select the element in the tool area and subsequently click in the editor multiple times. The connection line from one point to the other is drawn. Close the element with a double-click left or a single click right. The polyline is also closed when selecting another element in the tool area.

Under **Properties**, changes are taken into consideration and **Position ► Points** are **refreshed**

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	<i>Select from:</i> • Polygon • Polyline • Bézier curve

Element property "Position"

Such an element can be defined with a user-defined number of points, determining the element section and element form. Default setting is five points: Points [0] to [4]. The point is determined by the x-/y-coordinate in pixels.

Points • [<n>] – X – Y	• [<n>]: – X: Horizontal position in pixels. – Y: Vertical position in pixels.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Normal state • Frame color • Fill color	Select a frame and fill color for normal state. In case that the variable is defined in Color variables ▶ Toggle color is FALSE, the element is in normal state.
Alarm state • Frame color • Fill color	Select a frame and fill color with which the element is displayed in alarm condition. This is the case if the variable in Color variables ▶ Toggle color is TRUE.

Use color gradient	By default, the checkbox is disabled. If it is enabled, the element is drawn with color gradient.
Gradient setting	Dialog opens. chapter "Gradient editor" on page 814

Appearance

Property	Value description
Line width	Weight of frame line in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is required, set Line type to <i>invisible</i> .
Fill type	Select one of the fill types used in Fill color • <i>Filled</i>: The fill color is visible • <i>Invisible</i>: The fill color is invisible
Line style	<i>Select one of the line types used in the contour:</i> • <i>Solid</i> • <i>Dashes</i> • <i>Points</i> • <i>Dash dot</i> • <i>Dash dot dot</i> • <i>Invisible</i>: Contour invisible.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties

These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • <i>Left</i> • <i>Centered</i> • <i>Right</i>
Orientation vertical	<i>Define the vertical text alignment:</i> • <i>Top</i> • <i>Centered</i> • <i>Down</i>

Property	Value description
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property “Absolute motion”

The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise In contrast to Rotation, the element actually rotates.

“Dynamic points” element property

Dynamic definition for points defining the element.

Arrays of points	Enter a variable pointing to an array of the structure <code>VisuElems.VisuStructPoint</code>. The <code>iX</code> and <code>iY</code> components of <code>VisuElems.VisuStructPoint</code> describe the x-/y-coordinates of a point. The array contains all points of the element and the structure can be filled dynamically by a project variable. The number of element points has to be explicitly declared in Number of points. Example: <code>pPoints : POINTER TO ARRAY[0..100] OF VisuElems.VisuStructPoint;</code>
Number of points	An integer variable must be entered here to define the number of element points. Example: <code>iCount : INT:=24;</code> The element requires 24 individual points. This specification is necessary as the definition of individual points is made with a pointer, which does not allow to control the number of points.

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Dynamic texts

These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. <code>'M1'</code> or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. <code>'T1'</code> or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables'

Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ► Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>

Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:  0; 128; 64

Property	Value description
Normal state • Frame color • Fill color	Select a frame and fill color for normal state. In case that the variable is defined in Color variables ► Toggle color is FALSE, the element is in normal state.
Alarm state • Frame color • Fill color	Select a frame and fill color with which the element is displayed in alarm condition. This is the case if the variable in Color variables ► Toggle color is TRUE.

Appearance variables To be used in a **dynamic** definition of contour and element fill color appearance. Static definitions are specified in **Appearance**.

Property	Value description
Line width	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight .
Fill type	Input of a DWORD variables that defines the fill type. The color specified in Color variables can be displayed or ignored: - Variable value = 0: Filled - Variable value > 0 : Not visible, i.e. filling is visible This overwrites the fixed value specified in Appearance ► Fill type .
Line style	Entering a DWORD variable specifying the outer line contour. <i>The following values correspond to the following line types:</i> 0: Solid 1: Dash 2: Dot 3: Dash Dot 4: Dash Dot Dot 8: Hidden. The element is shown without an outer line.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> • No action • MouseDown action: OnMouseDown action when the key is pressed. • MouseUp action: OnMouseUp action if the key is released. • MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

"Circle segment" element

The circle segment inserted in the visualization editor by default, is a full circle with the center point at the position at which the element is dropped.



Fig. 20-39: Circle segment

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input:

- Starting and end point of the circle segment can be changed by dragging and dropping the small squares on the circle line in the editor. The properties **Start** and **End** are refreshed simultaneously.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.

- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Circle segment

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Beginning	Define the start of the circle segment line.
End	Defines the end of the belt-synchronous working area.
Variable for start	Enter a variable if the start of the circle segment line is to be dynamically defined with the variable. Click on  to open the input assistance.
Variable for end	Enter a variable, if required use the input assistance, if the end of the circle segment line is to be dynamically defined with the variable.
Only show circle arc	If the option is enabled, the circle segment is displayed without radius line for start and end

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:  0; 128; 64

Property	Value description
Normal state - Frame color - Fill color	Select a frame and fill color for normal state. In case that the variable is defined in Color variables ► Toggle color is FALSE, the element is in normal state.
Alarm state - Frame color - Fill color	Select a frame and fill color with which the element is displayed in alarm condition. This is the case if the variable in Color variables ► Toggle color is TRUE.

Use color gradient	By default, the checkbox is disabled. If it is enabled, the element is drawn with color gradient.
Gradient setting	Dialog opens. chapter "Gradient editor" on page 814

Appearance

Property	Value description
Line width	Weight of frame line in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is required, set Line type to <i>invisible</i> .
Fill type	Select one of the fill types used in Fill color - Filled: The fill color is visible - Invisible: The fill color is invisible
Line style	<i>Select one of the line types used in the contour:</i> - Solid - Dashes - Points - Dash dot - Dash dot dot - Invisible: Contour invisible.



The **line weight** and the **line type** do not effect the systems based on Windows CE.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the

sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Absolute motion"

The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable. Absolute coordinate values are used here.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center  .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise. In contrast to Rotation, the element actually rotates.

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Dynamic texts

These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. <code>'M1'</code> or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. <code>'T1'</code> or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables'

Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties** ► **Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>
Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code> . At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables"

The element property facilitates dynamic switching of the element colors using IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: 8F, green component: E0, blue component: 3F

- Grey: `dwGrey := 16#00888888`
 - Red: `dwRed := 16#00FF0000`
 - Black: `dwBlack := 16#00000000`
-

Toggle color	<i>Control variable</i> - IEC variable of type BOOL, e.g. <code>prgA.bToggleColor</code> At runtime, the visualization dynamically controls the color display of the element TRUE: Element in Normal state, e.g. in gray with a black frame FALSE: Element in Alarm condition, e.g. in yellow with red frame. - or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Normal state	Color variables of element during smooth operation
Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColor</code>
Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColor</code>
Alarm state	Color variables of element during an alarm
Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColorAlarm</code>
Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColorAlarm</code>

Appearance variables

To be used in a **dynamic** definition of contour and element fill color appearance. Static definitions are specified in **Appearance**.

Property	Value description
Line width	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight .
Fill type	Input of a DWORD variables that defines the fill type. The color specified in Color variables can be displayed or ignored: - Variable value = 0: Filled - Variable value > 0 : Not visible, i.e. filling is visible This overwrites the fixed value specified in Appearance ► Fill type.
Line style	Entering a DWORD variable specifying the outer line contour. <i>The following values correspond to the following line types:</i> 0: Solid 1: Dash 2: Dot 3: Dash Dot 4: Dash Dot Dot 8: Hidden. The element is shown without an outer line.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration' Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- **OnDialogClosed**
- **OnMouseClicked**
- **OnMouseDown**
- **OnMouseEnter**
- **OnMouseLeave**
- **OnMouseMove**
- **OnMouseUp**

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

Image and its dialog

Element 'Image'

Use this element to insert an image in the visualization. An image is assigned to the image element by specifying the **Static ID** for an unambiguous access to extend the name of the image pool. Dynamic configuration is also possible by assigning a project variable, whose value is the current image ID, for dynamic switching of the images. Use **Bitmap ID**.



Directories providing images for use in visualizations are specified in **Tools ► Options ► IndraLogic 2G Visualization** category.

A **background image** for the entire visualization can be defined using the [chapter 20.2.11 "Command "Background"" on page 773](#) command.

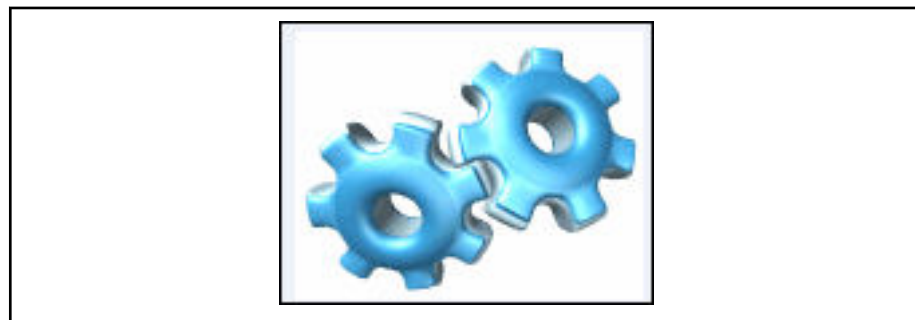


Fig. 20-40: "Ring gear" image

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.

- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Image

Static ID	Image file identifier for static definition. Enter the image file ID as it is defined in the corresponding image pool (STRING). To make the entry unique, the name of the pool should be added at the beginning (this is not necessary if the image file is managed in the image pool, since this pool is always searched first). For example: imagepool12.button_image. Click on  to open the dialog chapter "Dialog "Input assistant"" on page 865 and to list all available image pools with their images. See chapter 9.9 "Using image pools" on page 121 for more information about managing image files in image pools.
Draw frame	If this option is enabled, the image file is displayed with a frame.
Clipping	If this option is enabled together with Scaling type : If Unscaled is enabled, only the part of the visualization that fits in the frame is displayed, matching the frame.
Transparent	If the option is enabled, the color entered in the Transparent color property is displayed as transparent.
Transparency color	The  button opens the color selection dialog to select a color that is to be displayed as transparent if the Transparent option is enabled.

Scaling type	Define how the image file is to respond to changes in the element frame size. Isotropic: The image keeps its proportions; i.e. the height-width ratio remains even if the height and width of the element frame are modified separately. Please note the following if you want to retain the element alignment within a scaled frame element. For example, a lamp has been aligned in the center above a switch and the lamp is to remain in this horizontally centered position even if the frame size changes. To avoid undesired horizontal or vertical offsets, explicitly define the horizontal or vertical alignment with centered (see below). Anisotropic: The image adjusts to the size of the element frame; i.e. height and width can be modified independently of each other. Unscaled: The image keeps the original size even if the size of the element frame changes. Also take into consideration if the option Clipping is enabled. In case this setting has been selected, the element size is automatically adjusted to the image size upon each new ImageID assignment.
Orientation horizontal	Only available if the option Expert of the property editor is enabled and if the scaling type of the image is "isotropic". Is used to explicitly retain the horizontal alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: Isotropic setting - Left - Centered - Right
Orientation vertical	Only available if the option Expert of the property editor is enabled and if the scaling type of the image is isotropic. Is used to explicitly retain the vertical alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: Isotropic setting. - Top - Centered - Down

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:  0; 128; 64

Property	Value description
Color	Select a color to fill the element in normal state. This is the case if the variable in Color variables ▶ Toggle color is FALSE. By editing the value, the element changes color.
Alarm color	Select a color to fill the element in alarm condition. This is the case if the variable in Color variables ▶ Toggle color is TRUE.

Look and feel

Property	Value description
Line width	Frame line weight in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is to be used, set line type to <i>invisible</i> .
Line style	<i>Select one of the line types used in the contour:</i> • Solid • Dashes • Points • Dash dot • Dash dot dot • Invisible: The contour is invisible.

Element property "Texts" These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Tooltip variable .

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down

Property	Value description
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Image ID variable"

Image ID	Assign a project variable, containing the image ID. See chapter 9.9 "Using image pools" on page 121 for more information about managing image files in image pools.

Element property "Absolute motion"

The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise In contrast to Rotation, the element actually rotates.

Element property "Relative movement"

The element can be displayed in relation to its fixed position defined in **Position**. The top left and bottom right corner of the element are displaced by the values (pixel) specified in the integer variable in x- or y-direction. In contrast to absolute motion, a relative position is defined, i.e. the distance to the original position. The shape of the element can thus be modified.

See **Absolute motion** for displacing the entire element.

Movement top-left	
X	Integer variable whose value specifies the number of pixels by which the top left corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the top left corner is moved in y-direction.
Movement bottom-right	
X	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in y-direction.

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ▶ Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ▶ Tooltip has to contain %s.

Dynamic texts

These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. <code>'M1'</code> or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. <code>'T1'</code> or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables'

Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ▶ Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>

Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables"

The property facilitates dynamic toggling of colors by means of IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Example of color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: `8F`, green component: `E0`, blue component: `3F`

- Grey: `dwGrey := 16#00888888`
- Red: `dwRed := 16#00FF0000`
- Black: `dwBlack := 16#00000000`

Toggle color	<i>Control variable</i> - as IEC variable of type <code>BOOL</code>, e.g. <code>prgA.bToggleColor</code> At runtime, the visualization dynamically controls the color display of the element <code>TRUE</code>: Element is displayed in color, e.g. grayed out <code>FALSE</code>: Element is displayed in alarm color, e.g. red - or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Color	IEC variable of type <code>DWORD</code> for the color, e.g. <code>prgA.dwColor</code>
Alarm color	IEC variable of type <code>DWORD</code> for the alarm color, e.g. <code>prgA.dwAlarmColor</code>

Appearance variable

To be used in a dynamic definition of contour and element fill color appearance. Static definitions are specified in **Appearance**.

Property	Value description
Line width	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight .
Line style	Entering a <code>DWORD</code> variable specifying the outer line contour. <i>The following values correspond to the following line types:</i> 0: Solid 1: Dash 2: Dot 3: Dash Dot 4: Dash Dot Dot 8: Hidden. The element is shown without an outer line.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is <code>TRUE</code> , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is <code>TRUE</code>, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration' on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> • No action • MouseDown action: OnMouseDown action when the key is pressed. • MouseUp action: OnMouseUp action if the key is released. • MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.

Property	Value description
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

Dialog “Input assistant”

The input assistance for images of the image pool is opened automatically if a visualization element **Image** is inserted in the visualization for the first time. Alternatively, access the dialog by clicking in the **Static ID** value field in the property view of an image element and execute .

“Text search”

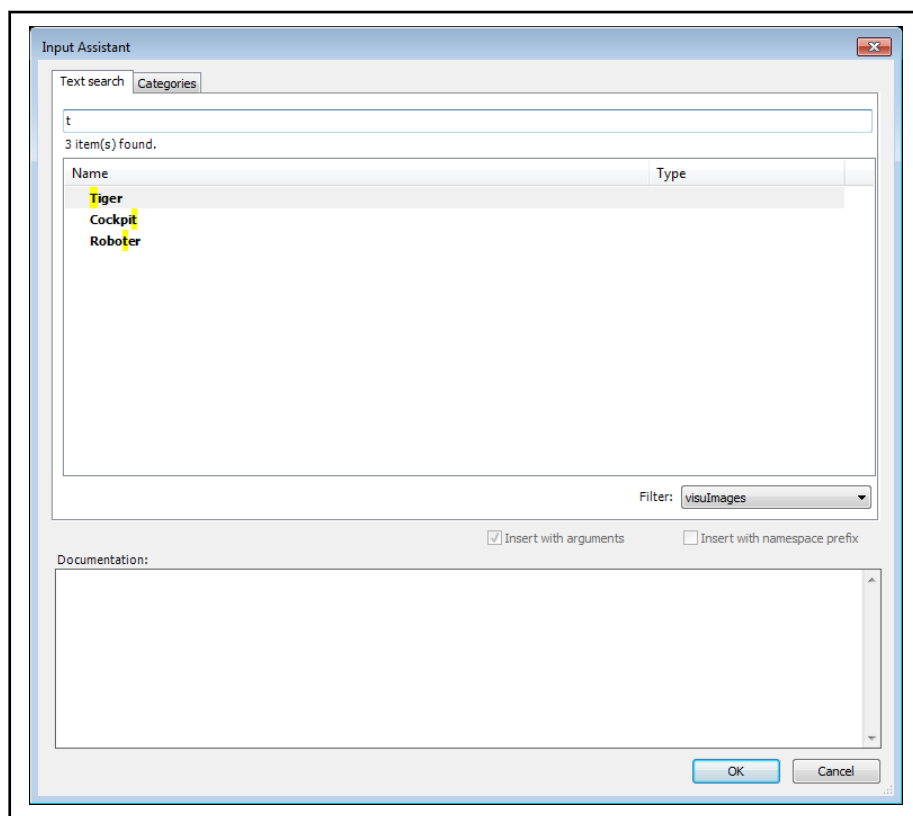


Fig. 20-41: Input assistance during image assignment

First set the **Filter** by selecting the image pool from the drop-down menu. Enter the name or part of the name of the image file in the upper line editor. All matching image file names are displayed. Select an image and close the dialog with <OK>. Or double-click on the image.

After the dialog was closed, the <Name image pool>.<Name image> image is referenced in the **Static ID** property.

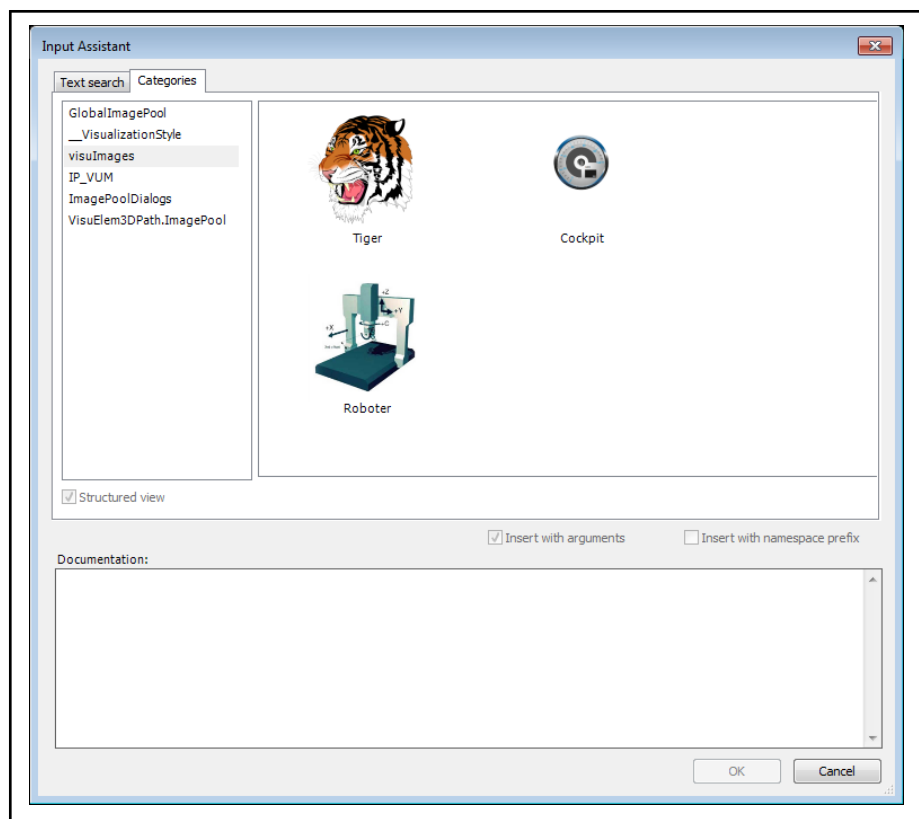
"Categories"

Fig. 20-42: Input assistance during image assignment

Categories: All project image pools are listed in the left column. Select an image pool and the images are displayed as preview. Select an image and close the dialog with <OK>. Or double-click on the image. After the dialog was closed, the <Name image pool>.<Name image> image is referenced in the **Static ID** property.

Element 'Frame'

A frame element is used

- as ["Reference to another visualization"](#) on page 866
- To ["provide an interface for placeholders"](#) on page 867
- To ["toggle between different visualizations"](#) on page 867

Content of this page:

["Element properties"](#) on page 868

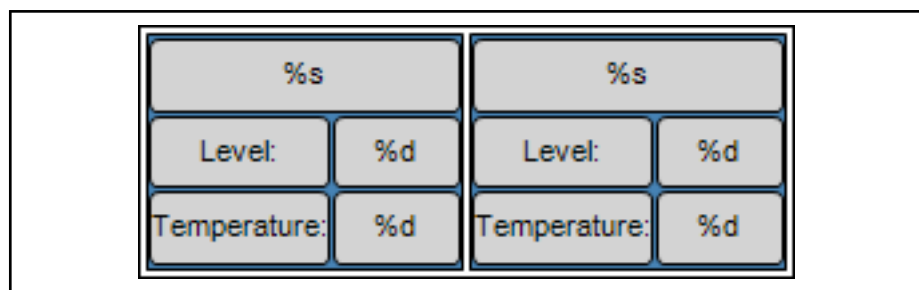


Fig. 20-43: Frame

Reference to another visualization

In principle, a visualization can be added to another visualization and thus referenced. Use a **Frame** that can include one or several visualization references. The position in the POU tree and the device tree in the currently

provide an interface for placeholders	opened visualization decide which visualization is currently available. See also chapter 20.2.10 "Frame selection" commands on page 772. A visualization can provide an interface, to be opened via the "Interface editor" command, in which the input variables can be defined as for a function block (chapter "Basic information on the interface editor" on page 782). These input parameters function as placeholders. In an instance (reference) of the visualization, a placeholder has to be replaced by values or expressions for the specific use in the local object. The replacement has to be carried out in the properties of the frame that contain the visualization instances: Referenced visualizations ► Referenced visualizations. Note that the input variables of a visualization instance assigned to a specific application have to be assigned to valid variables of this application.
toggle between different visualizations	A frame defines a part of the current visualization that can contain more visualizations that can be toggled in online mode. In these visualizations, inputs are configured to trigger a display change of these references in the frame. The chapter "Input configuration" on page 800 dialog provides the option Visualization switch. Take into consideration that mouse actions can be defined using the Input configuration property. To select the visualizations that are to be defined in a frame, the dialog Frame configuration can be opened by selecting the frame and by executing the chapter 20.2.10 "Frame selection" commands on page 772 command. All available visualizations are provided here. Select one and insert it in the frame. It is recommended to only use visualizations of the global POU pool to avoid problems in case a device is renamed. A "filled" frame also shows all visualizations in offline mode. <i>Example:</i> Toggle between visualizations A visualization contains three buttons "1", "2", "3" and a frame to which visualizations "Visu1", "Visu2" and "Visu3" are assigned. The buttons are configured (input, OnMouse actions, visualization switch) so that a user input to a button calls the respective visualization in the frame. In this way - in contrast to IndraLogic 1.x - within a visualization element, other elements can be displayed; in other words, a control panel can be visualized.
User input when editing an element	 This concept replaces the visualization and placeholder concept in IndraLogic 1.x. <i>When editing an elements in the visualization editor, the following user inputs are provided:</i> • visualization commands for at least one element: chapter 20.2 "Commands" on page 765 • different mouse operations, applicable at the element: chapter "User input when editing an element" on page 795
User input when editing element properties	<i>Element-specific command:</i> • chapter 20.2.10 "Frame selection" commands on page 772 <i>Properties define the characteristic features of a visualization element:</i> • Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization. • its order in the visualization. • its animation: the visualization element can be animated by directly using IndraLogic project variables.

- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Frame

Clipping	If this option is enabled together with " Scaling type : If <i>Unscaled</i> " is enabled, only the part of the visualization that fits in the frame is displayed.
Draw frame	If this option is enabled, the referenced visualization is displayed with a frame.
Scaling type	<i>Specify how the frame is to react to a change in the visualization size:</i> • Isotropic: The frame retains its proportions. Thus, the height/width ratio can be retained even if the height and width of the visualization have been modified independently. • Anisotropic: The frame depends on the visualization size so that height and width of the referenced visualization can be modified independently. • Unscaled: The original frame size is retained irrespective of the visualization size. If the option Clipping is also enabled only the matching part is displayed. • Unscaled and scrollbar: Use this option to display the referenced visualization without scaling. If it is larger than the frame window area, a scrollbar is added to the frame to be able to move the displayed visualization area. If you want to set the scrollbar position using a variable, use the properties Variable scroll position horizontal or Variable scroll position vertical, see ""Scrollbar settings" element property" on page 874.
Disabling the background drawing	To optimize the visualization performance, the non-animated elements of the frame element can be drawn as background bitmap. This can result in a different element order. Disable this function to avoid this behavior.

Referenced visualizations

Referenced visualizations	The command Configure opens the dialog chapter 20.2.10 ""Frame selection" commands" on page 772. Referenced element visualizations are listed.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:  0; 128; 64

Property	Value description
Color	Select a color to fill the element in normal state. This is the case if the variable in Color variables ▶ Toggle color is FALSE. By editing the value, the element changes color.
Alarm color	Select a color to fill the element in alarm condition. This is the case if the variable in Color variables ▶ Toggle color is TRUE.

Look and feel

Property	Value description
Line width	Frame line weight in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is to be used, set line type to <i>invisible</i> .
Line style	<i>Select one of the line types used in the contour:</i> • Solid • Dashes • Points • Dash dot • Dash dot dot • Invisible: The contour is invisible.

Element property "Texts" These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Absolute motion" The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable. Absolute coordinate values are used here.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always refers to the point of rotation. Center .

Element property 'Relative movement'

The element can be displayed in relation to its fixed position defined in **Position**. The top left and bottom right corner of the element are displaced by the values (pixel) specified in the integer variable in x- or y-direction. In contrast to absolute motion, a relative position is defined, i.e. the distance to the original position. The shape of the element can thus be modified.

See **Absolute motion** for displacing the entire element.

Movement top-left	
X	Integer variable whose value specifies the number of pixels by which the top left corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the top left corner is moved in y-direction.
Movement bottom-right	
X	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in y-direction.

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language" on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Dynamic texts These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. 'M1' or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. 'T1' or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables' Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties** ► **Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>
Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code> . At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables" The property facilitates dynamic toggling of colors by means of IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is re-

served for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always 16#00.

Example:

Example of color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: 8F, green component: E0, blue component: 3F

- Grey: dwGrey := 16#00888888
- Red: dwRed := 16#00FF0000
- Black: dwBlack := 16#00000000

Toggle color	<i>Control variable</i> • as IEC variable of type BOOL, e.g. prgA.bToggleColor At runtime, the visualization dynamically controls the color display of the element – TRUE: Element is displayed in color, e.g. grayed out – FALSE: Element is displayed in alarm color, e.g. red • or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Color	IEC variable of type DWORD for the color, e.g. prgA.dwColor
Alarm color	IEC variable of type DWORD for the alarm color, e.g. prgA.dwAlarmColor

Appearance variable

To be used in a dynamic definition of contour and element fill color appearance. Static definitions are specified in **Appearance**.

Property	Value description
Line width	Input of an integer variables that define the line weight for the element in pixel. This overwrites the fixed value specified in Appearance ► Line weight .
Line style	Entering a DWORD variable specifying the outer line contour. <i>The following values correspond to the following line types:</i> • 0: Solid • 1: Dash • 2: Dot • 3: Dash Dot • 4: Dash Dot Dot • 8: Hidden. The element is shown without an outer line.

"Scrollbar settings" element property

It is recommended to use the variable client specifically. If the variables change or if a scrollbar is moved using the mouse, the changes only affect the frame of the affected client in this case. Otherwise, all clients are updated.

Variable scroll position horizontal	Enter a variable containing the position of the horizontal scrollbar.
Variable scroll position vertical	Enter a variable containing the position of the vertical scrollbar.

Client for specific variable for the command position

IEC code

```
PROGRAM PlcProg
VAR
    iScrollHor : ARRAY[0..20] OF INT;
    iScrollVer : ARRAY[0..20] OF INT;
END_VAR
```

Variable scroll position horizontal	PlcProg.iScrollHor[CURRENTCLIENTID]
Variable scroll position vertical	PlcProg.iScrollVer[CURRENTCLIENTID]

Tab. 20-6: Specified properties for the frame element

Switching variable

This property facilitates switching visualizations within a frame.

Property	Value description
Variable	Specify an integer variable whose value contains the ID of the visualization to be displayed. The ID of a visualization is determined by its order in the list of the assigned visualizations in the dialog Configuration of frame visualizations . The first entry in this list results in the ID 0 for the integer value, the second value results in ID 1, etc.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys

Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration' on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

20.4.3 'General control elements'

Element "Text"

The **Text** element is used to add labels, headers and any type of text to a visualization.



Fig. 20-44: Text

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Text

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

State variables

Property	Value description
Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.

“Combo box, integer” element

This element facilitates selecting a value from a drop-down list. The index of the selected value is written to a variable. The entries can originate from a text list or image list in an image pool.

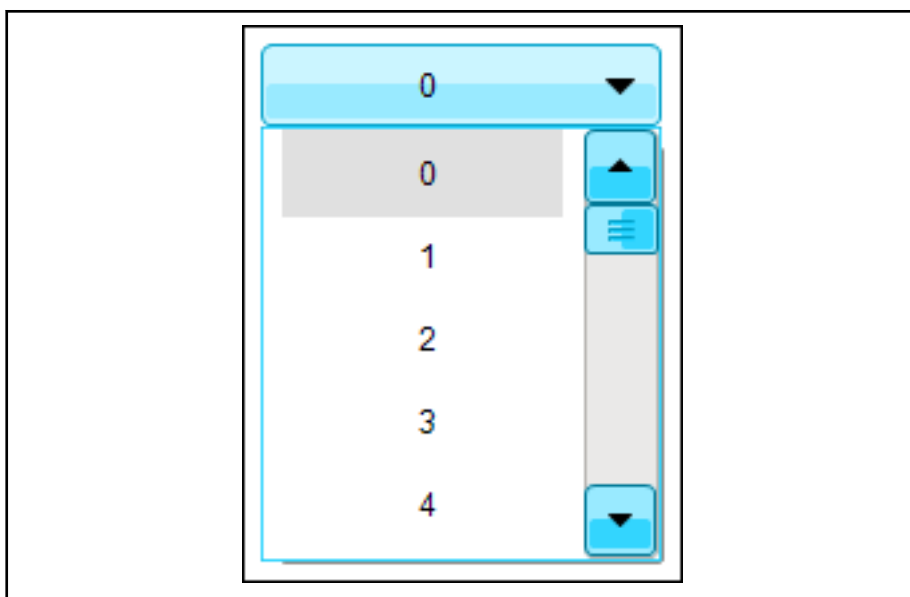


Fig. 20-45: Combo box, integer, style 8

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project. [chapter 9.5 "Using input assistants" on page 107.](#)

Element properties
Element name

All element properties and their descriptions are listed below.

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Combo box, integer

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Enumeration variable with the index value of the selected list element.
Text list	Assigned text list determining the drop-down menu of the combo box, for example 'TextList_A'. Note: If a text list is not defined, then you have to type a minimum value and maximum value in the Value range property. ▼ : The list box opens with project text lists
Image pool	Collection of images for the list box of the element (for example, ImagePool_A). ▼ : The list box opens with project image pools

Element property "List settings"

"List settings"	Displayed list that expands when a visualization user clicks into the element.
Line height	Default setting: From style
Number visible rows	Restricts the number of rows in the list box. Default setting: From style
Height of image	Default setting: From style Prerequisite: One is specified in the Image pool .
Width of image	Default setting: From style Prerequisite: One is specified in the Image pool .

Offset of image	Default setting: From style Prerequisite: One is specified in the Image pool .
Width of the scrollbar	Width of scrollbar that is displayed if more entries are available in the drop-down list than specified in Number of visible lines . Default setting: 20

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Element property 'Value range'

Value range	Minimum and maximum values for the list box
Value range limit	 : The visualization extends or limits the value range of the list box according to the text list, depending on the defined values in Minimum value and Maximum value . If you do not define anything, the list box corresponds directly to the text list. Prerequisite: Visible only if one is selected in the Text list property.
Minimum value	Index of the first entry in the list box (for example, 0). If you do not select a text list, then you must define a number here.
Maximum value	Index of the last entry in the list box (for example, 20). If you do not select a text list, then you must define a number here.
Filter missing text entries	 : The visualization shows the entries in the list box exactly as in the text list. Entries are removed if the Value range is limiting.  : The visualization shows the entries in the list box exactly as defined in Value range . The entries are supplemented if they are missing in the text list. Prerequisite: Visible only if one is selected in the Text list property.

Element property 'Text properties'

Text property	Representation of the text in the drop-down list
Usage of	Default style values Individual settings: Additional properties are visible, facilitating a user-defined representation in the drop-down list.

Individual text properties	Prerequisite: In Use of , User-defined setting has been selected.
Orientation horizontal	Horizontal text alignment
Orientation vertical	Vertical text alignment
Fonts	Formatting text

Font color	Text color
Transparent	Transparency level of text, 255 is opaque.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Combo box array" element

Combo box array allows the user to select a line with values of a drop-down list. The index of the selected line is written to a variable.

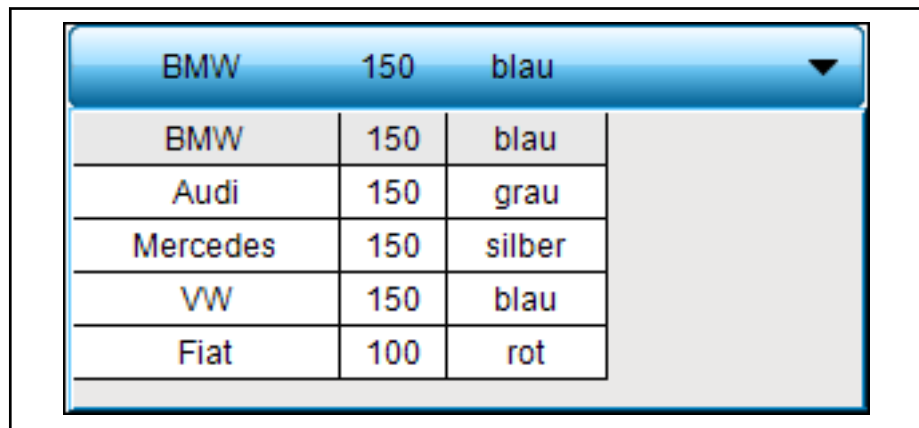


Fig. 20-46: Combo box, array, style 8

User input when editing an element

When editing an element in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**

- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Combo box ARRAY

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Variable containing the index of the selected line
Data array	Variable of type array with complete path which is to be displayed. The structure of the variables determines the number of rows and columns in the table.

Element property "Columns"

The "Combo Box, Array" element displays an array variable or structure variable in a tabular view. The index of array elements or components of a structure is displayed in a column or a line. Two-dimensional arrays or arrays of a structure are shown in multiple columns. The visualized variable is specified in the **Data array** property. When a variable is assigned there, you can specify the display of the table columns where the array elements are shown. For each column to which an index [<n>] is assigned, an individual configuration is possible.

Columns	Due to the variable structure defined under data array, the number of columns is determined and written via index <n>.
• [<n>]	StringTable: ARRAY [0..2, 0..4] OF STRING := ['BMW','Audi','Mercedes','VW','Fiat', '150','150','150','150','100','blue','gray','silver','blue','red']; three columns are formed [0], [1] and [2].
Max. array index	Optional numerical variables that can be specified via a maximum array index.

Line height	Height of the rows (in pixels).
Scrollbar size	Width of the vertical scrollbar in pixel.

Element property "Columns": Column [<n>]

Width	Width of the column (in pixels)
Image column	Using this option, images can be displayed in this column. If it is activated, images of the global image pool or of the user-defined image pools in this column of the table can be displayed. The cell values of a table specify the image ID in the image pool.
Image configuration - Fill mode - Transparency - Transparency color	Fill mode Fill cell: The image is adjusted to the cell size without taking the width-height-ratio into consideration. Centered: The image is centered by retaining the proportions within the cell. Even if the height or width is adjusted individually, the width-height-ratio is retained. If Transparency is enabled, the color selected under Transparency color is displayed as transparent. Transparency color: Select a color from the combo box or using the color selection dialog that can be opened via the  interface. The color is transparent if the option Transparency is enabled.
Text alignment in column	<i>Column alignment:</i> - Left - Centered - Right

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element "Tab control element"

A tab control element facilitates opening of several forms within a window. Tabs are used as navigation widget to toggle between the forms.

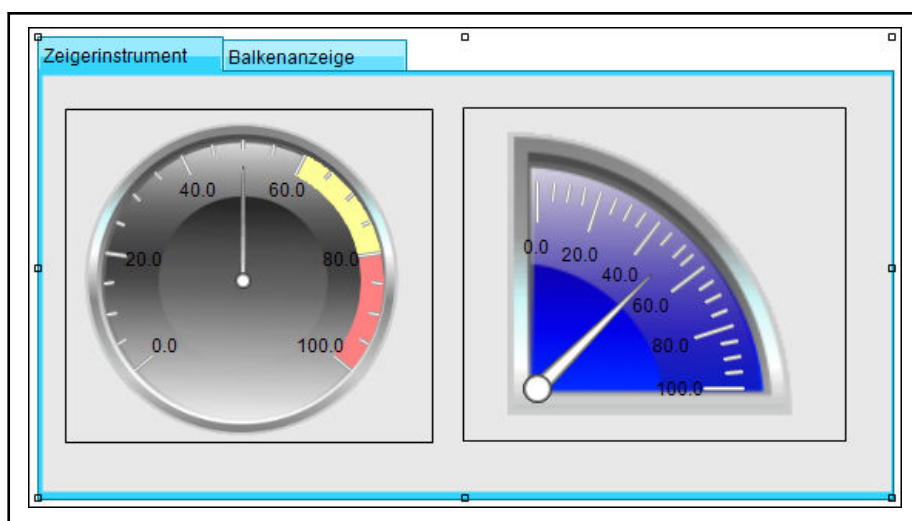


Fig. 20-47: Register control element, style 8

If not all tabs can be displayed due to a narrow element, two navigation arrows are attached to the element at the top right corner.

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties Element name

All element properties and their descriptions are listed below.

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Tab control element

Tab width	Defines the width of the tab in pixel.
Scaling type	<i>Specify how the window is to react to changes in size:</i> • Isotropic: the referenced element retains its proportions; thus, the height/width ratio is not changes even if the height and width are changes. • Anisotrope: the tab control element depends on the size so that height and width of the referenced visualization can be modified independently. • Unscaled: The original visualization size is retained irrespective of the tab control element size. • Unscaled and scrollbar: Use this option to display the referenced visualization without scaling. If it is larger than the frame window area, a scrollbar is added to the frame to be able to move the displayed visualization area. If you want to set the scrollbar position using a variable, use the properties Variable scroll position horizontal or Variable scroll position vertical, see ""Scrollbar settings" element property" on page 886.
Disabling the background drawing	To optimize the visualization performance, the non-animated elements of the frame element can be drawn as background bitmap. This can result in a different element order. Disable this function to avoid this behavior.

Referenced visualizations

Referenced visualizations	The command Configure opens the dialog chapter 20.2.10 ""Frame selection" commands" on page 772. Referenced element visualizations are listed.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

"Scrollbar settings" element property

It is recommended to use the variable client-specifically. If the variables change or if a scrollbar is moved using the mouse, the changes only affect the frame of the affected client in this case. Otherwise, all clients are updated.

[chapter "Element 'Frame'" on page 866](#).

Variable scroll position horizontal	Enter a variable containing the position of the horizontal scrollbar.
Variable scroll position vertical	Enter a variable containing the position of the vertical scrollbar.

Switching variable This property facilitates switching visualizations within a frame.

Property	Value description
Variable	Specify an integer variable whose value contains the ID of the visualization to be displayed. The ID of a visualization is determined by its order in the list of the assigned visualizations in the dialog Configuration of frame visualizations . The first entry in this list results in the ID 0 for the integer value, the second value results in ID 1, etc.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element 'Button'

A visualization user clicking on the **button** or executing another event, triggers a defined follow-up action. Optically adjust the button, e.g. by assigning an image or by configuring an etched view.

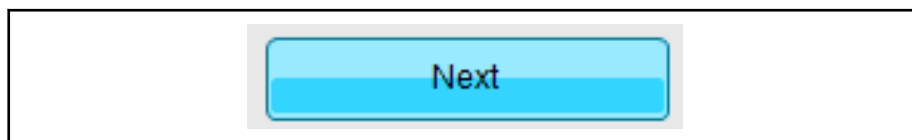


Fig. 20-48: Button, style 8

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**

- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Button

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Center By editing this value, the element  is repositioned in the editor.

Property	Value description
X	X-position of the rotation point of the element
Y	Y-position of the rotation point of the element

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Color	Select a color to fill the element in normal state. This is the case if the variable in Color variables ► Toggle color is FALSE. By editing the value, the element changes color.
Alarm color	Select a color to fill the element in alarm condition. This is the case if the variable in Color variables ► Toggle color is TRUE.

Use color gradient	By default, the checkbox is disabled. If it is enabled, the element is drawn with color gradient.
Gradient setting	Dialog opens. chapter "Gradient editor" on page 814

Button height	Height of etched view in pixel, e.g. 

Element property "Image"

Static ID	Reference to an image in an image pool in the format <image pool name>.<image ID>, e.g. Imagepool.GreenButton If the image is part of the GlobalImagePool, the image pool name is not required as IndraLogic always searches this pool first. : Input assistance dialog opens and lists all project-wide available image pools and their images.
Scale type	<i>Image behavior in case of a button size change</i> • Isotropic: The image retains its proportions. The height-width ratio is retained even if the height or width of the button is changed individually. • Anisotropic: The image adjusts to the button size. • Unscaled: The image retains its original size even if the size of the button changes.
Transparency	The visualization displays the color selected in Transparency color in the image as transparent.
Transparency color	Color which is transparent in the image, e.g. white. If the image background referenced in Static ID is white, this background is displayed as transparent. When clicking on , the color selection dialog is opened. Prerequisite: Transparency option is enabled.
Orientation horizontal	<i>Horizontal alignment of the bitmap</i> • Left • Centered • Right
Orientation vertical	<i>Vertical alignment of the bitmap</i> • Top • Centered • Down

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties These properties are used for static definition of the font.

Property	Value description
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property "Absolute motion" The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).
Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point; Center . positive value = mathematically negative = clockwise Take into consideration that the element itself is not rotating in contrast to the behavior for Internal rotation.
Scaling	Entering an integer variable, whose value defines the current scale value (in percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation Center .
Internal rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around the rotation point; Center . positive value = mathematically negative = clockwise In contrast to Rotation, the element actually rotates.

Element property 'Relative movement' The element can be displayed in relation to its fixed position defined in **Position**. The top left and bottom right corner of the element are displaced by the values (pixel) specified in the integer variable in x- or y-direction. In contrast to absolute motion, a relative position is defined, i.e. the distance to the original position. The shape of the element can thus be modified.

See **Absolute motion** for displacing the entire element.

Movement top-left	
X	Integer variable whose value specifies the number of pixels by which the top left corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the top left corner is moved in y-direction.

Movement bottom-right	
X	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in x-direction.
Y	Integer variable whose value specifies the number of pixels by which the bottom right corner is moved in y-direction.

"Text variables" element property Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Dynamic texts These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. <code>'M1'</code> or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. <code>'T1'</code> or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables' Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties** ► **Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>

Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables"

The property facilitates dynamic toggling of colors by means of IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Example of color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: `8F`, green component: `E0`, blue component: `3F`

- Grey: `dwGrey := 16#00888888`
- Red: `dwRed := 16#00FF0000`
- Black: `dwBlack := 16#00000000`

Toggle color	Control variable as IEC variable of type <code>BOOL</code>, e.g. <code>prgA.bToggleColor</code>. <i>At runtime, the visualization dynamically controls the color display of the element</i> • TRUE: Element is displayed in color, e.g. grayed out • FALSE: Element is displayed in alarm color, e.g. red
Color	IEC variable of type <code>DWORD</code> for the color, e.g. <code>prgA.dwColor</code>
Alarm color	IEC variable type <code>DWORD</code> for the alarm color, e.g. <code>prgA.dwAlarmColor</code>

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Button state variable" element property

Boolean value	<i>Control variable</i> - as IEC variable of type <code>BOOL</code>, e.g. <code>prgA.bStateButton</code> In this case, the visualization dynamically controls the button state at runtime <code>TRUE</code>: Button is displayed as pressed <code>FALSE</code>: Button is displayed as not pressed - or as placeholder <Toggle variable/key variable> In this case, no individual variable is assigned to control the button. Instead, the state is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
	IndraLogic inserts the placeholder <Toggle variable/key variable> .

"Image ID variable" element property

Image ID	IEC variable of type <code>STRING</code> for the image ID

Also refer to

- [chapter 9.9 "Using image pools" on page 121](#)

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed**
- OnMouseClicked**
- OnMouseDown**
- OnMouseEnter**
- OnMouseLeave**
- OnMouseMove**
- OnMouseUp**

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys

Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration' on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

“Group box” element

Elements can be moved to a group box using drag-and-drop to form an optical unit. They can be nested multiple times.

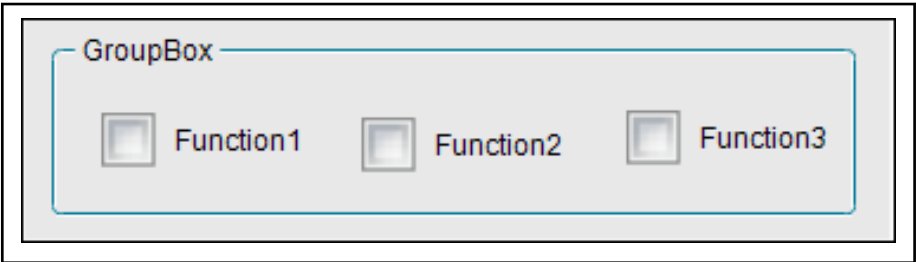


Fig. 20-49: Group box

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.
[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties
Element name

All element properties and their descriptions are listed below.

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Group box

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The

positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties

These properties are used for static definition of the font.

Property	Value description
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Table" element

Page content:

- ["Element properties" on page 899](#)

- ["Assign data to the table" on page 903](#)
- ["Configure table properties" on page 904](#)

A table element can be inserted into a visualization to visualize one- or two-dimensional arrays, structures or the local variables of a POU.

	X	Y	Vel
0			
1			
2			
3			
4			

Fig. 20-50: Table without assigned data array, style 8

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input:

- The column width can be directly changed by moving the hyphens between two columns to the desired position while the cursor has the form .
- A table column can be **duplicated** by pressed <Ctrl> and while the button is pressed, click on the button of the other column.
- A table column can be **deleted** by pressing <Ctrl> + <Shift> and while the keys are pressed, click on the button of the other column.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)



If the variable size on which the table is based (array limits, number of structure components etc.) are changed in the IEC code but the affected table element was not updated in the visualization, a compiler error message is output.

Open the property configuration and modify the table, e.g. press <Enter> in the value field of the "Data array" property.

However, the error message can also be opened by double-clicking. This opens a message box to specify **Yes** or **No** to update the table configuration immediately.

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Table

Data array	The variable to be visualized with the complete path is specified. The variable structure determines the number of columns and rows of the table. If the number of array elements of the variable varies, use Max. array index . To update the table if the size of array or structure variables has changes, position the cursor in the data array value field and press <Enter>.
Max. array index	Assign a numerical variable that specifies the maximum array index dynamically .



If the **data array** value is changed, i.e. a new variable is assigned, all other table properties are reset to default.

Element property "Columns"

The table element visualizes a variable in tabular view. The index of the array or the components of the structure are displayed in a column or line. In case of two-dimensional arrays or an array of structures, several columns are used. The table column design is defined here, in which the values of the individual array elements/structure components/variables are displayed. An individual configuration is possible for each column that is associated with a specific index.

Column • [<n>]	According to the variable structure defined in Data array , the number of columns is identified and indexed in <n> . var_data_array: ARRAY [2..8] OF INT: one Column Column [0] is identified. var_data_array: ARRAY[1..3][1..10] OF INT: The three columns [0], [1] and [2] are identified
Show line header	Checkbox to disable or enable the line labeling display as assigned line indexes
Show column header	Checkbox to disable or enable the column labeling display as assigned line indexes
Line height	Height of the table lines in pixel

Width of line header	Width of the column containing the line header in pixel
Width of the scrollbar	Width of vertical or height of horizontal scroll bar in pixel

Element property "Columns": Column [*<n>*]

Column header	By default the name of the array/structure along with the index/component associated with the column are used as a heading. The column label can be changed by entering a new title here.
Width	Column width in pixels
Image column	If this option is enabled, images can be displayed in this column. Images of the global image pool or of the user-defined image pools in this column of the table can be displayed. Image IDs as defined in the image pool are specified in the table cells.
Image configuration - Fill mode - Transparency - Transparency color	Fill mode Fill cell: The image adjusts to the cell size without taking the height-width ratio into consideration. Centered: The image is centered in the cell and retains its proportions, i.e. the height-width ratio is retained. If Transparency is enabled, the color entered in the Transparent color is displayed as transparent. Transparency color: Specify a color. A color can also be selected using , opening a color selection dialog. The color is transparent if the option Transparency is enabled.
Text alignment of the header	<i>Alignment of column header:</i> - Left - Centered - Right
Use template	When the option is activated, another visualization element will be added into each cell (element type Rectangle , Rounded Rectangle or Ellipse). The properties list is automatically extended to include the properties of this element in Template .
Text alignment of the template header	Selecting this checkbox only takes effect if Use template is also activated. When it is activated, the settings selected in the inserted template for font (size) and alignment are also applied to the column header.
Template	This entry is only available if Use template is activated. Under Template the element properties assigned to the column are successively listed and can be edited as described under chapter "Table" element" on page 897.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

Dynamic texts These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. 'M1' or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. 'T1' or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables' Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ► Font**.

Font name	Enter a variable of type <code>STRING</code> containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type <code>INT</code> containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>

Flags	Entering a variable of type <code>DWORD</code> that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A <code>DWORD</code> variable can be used to input this number (see also the Script definition in the default font dialog)
Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is <code>TRUE</code> , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is <code>TRUE</code>, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Selection" element property

Selection color	Fill color for selected table cells.
Selection type	<i>Defines the selection made upon click on the table cell:</i> • <code>Select none</code>: No selection • <code>Cell selection</code>: only the highlighted cell is selected. • <code>Line selection</code>: Line selection signifies that the line where the cell is highlighted is selected. • <code>Column selection</code>: Column selection signifies that the column where the cell is highlighted is selected. • <code>Line and column selection</code>: Line and column selection signifies that the column where the cell is highlighted is selected.
Frame around selected cells	If this option is enabled, a frame is drawn around the selected cells.

Variable for column selection	Variable type INT, containing the array index of the column of the selected cell. In case the data array points to a structure, the structure components starting with 0 are indexed. Note that the index only represents the correct position in the array if no columns were removed from the table display.
Variable for line selection	Variable type INT, containing the array index of the line of the selected cell.
Variable for validity of column selection	Boolean variable that is TRUE if variable for column selection contains a valid value.
Variable for validity of line selection	Boolean variable that is TRUE if variable for line selection contains a valid value.

Assign data to the table

First, specify the variable to be visualized in the table view. Enter a variable with complete path in **data array**. The variable structure determines the number of lines and columns in the table.

If the array is two-dimensional, the **columns** represent the second index. In case of a structure, the individual components represent the structure, in case of a POU represent the local variables. For example, a one-dimensional array of elements of a simple basic type is represented using a single column, a structure is represented by a single line. However, an array of structure type elements is displayed in a table in which the number of lines corresponds to the number of array elements and the number of columns corresponds to the number of structure components.

Subsequently, a table can be configured using more properties.

Variable declaration	Resulting table																				
arrDim1 : ARRAY[2..5] OF INT;	<table><tr><td></td><td>arrDim1[INDEX]</td></tr><tr><td>2</td><td></td></tr><tr><td>3</td><td></td></tr><tr><td>4</td><td></td></tr><tr><td>5</td><td></td></tr></table> Note that the indexing in the line labeling matches the indexes of the first dimension of the array; for this reason, the line numbering in this example starts with "2"!		arrDim1[INDEX]	2		3		4		5											
	arrDim1[INDEX]																				
2																					
3																					
4																					
5																					
arrDim2 : ARRAY[0..2, 0..3] OF INT;	<table><tr><td></td><td>arrDim2[0,INDEX]</td><td>arrDim2[1,INDEX]</td><td>arrDim2[2,INDEX]</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td></tr></table>		arrDim2[0,INDEX]	arrDim2[1,INDEX]	arrDim2[2,INDEX]	0				1				2				3			
	arrDim2[0,INDEX]	arrDim2[1,INDEX]	arrDim2[2,INDEX]																		
0																					
1																					
2																					
3																					
TYPE MyStruct : STRUCT iNo: INT; bBool : BOOL; sText:STRING; END_STRUCT END_TYPE varStruct: MyStruct;	<table><tr><td></td><td>varStruct.iNo</td><td>varStruct.bBool</td><td>varStruct.sText</td></tr><tr><td>0</td><td></td><td></td><td></td></tr></table>		varStruct.iNo	varStruct.bBool	varStruct.sText	0															
	varStruct.iNo	varStruct.bBool	varStruct.sText																		
0																					

IndraLogic 2G visualization

Variable declaration	Resulting table																																			
arrStruct : ARRAY[0..3] OF MyStruct;	<table><tr><th></th><th>arrStruct[INDEX].iN</th><th>arrStruct[INDEX].bBool</th><th>arrStruct[INDEX].sText</th></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td></tr></table>		arrStruct[INDEX].iN	arrStruct[INDEX].bBool	arrStruct[INDEX].sText	0				1				2				3																		
	arrStruct[INDEX].iN	arrStruct[INDEX].bBool	arrStruct[INDEX].sText																																	
0																																				
1																																				
2																																				
3																																				
varFB: TON;	<table><tr><th></th><th>varFB.iN</th><th>varFB.PT</th><th>varFB.Q</th><th>varFB.ET</th><th>varFB.M</th><th>varFB.StartTime</th></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		varFB.iN	varFB.PT	varFB.Q	varFB.ET	varFB.M	varFB.StartTime	0																											
	varFB.iN	varFB.PT	varFB.Q	varFB.ET	varFB.M	varFB.StartTime																														
0																																				
arrFB : ARRAY[0..3] OF TON;	<table><tr><th></th><th>arrFB[INDEX].iN</th><th>arrFB[INDEX].PT</th><th>arrFB[INDEX].Q</th><th>arrFB[INDEX].ET</th><th>arrFB[INDEX].M</th><th>arrFB[INDEX].StartTime</th></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		arrFB[INDEX].iN	arrFB[INDEX].PT	arrFB[INDEX].Q	arrFB[INDEX].ET	arrFB[INDEX].M	arrFB[INDEX].StartTime	0							1							2							3						
	arrFB[INDEX].iN	arrFB[INDEX].PT	arrFB[INDEX].Q	arrFB[INDEX].ET	arrFB[INDEX].M	arrFB[INDEX].StartTime																														
0																																				
1																																				
2																																				
3																																				

Tab. 20-7: Assigned data and resulting table

Configure table properties

In addition to a few basic properties, e. g. size, position or line width in the table, a wide variety of specific properties can be set for the table. A selection of these properties is explained in details using a step-by-step continuation of the **example** arrStruct. The consequences are explained in the right column of the following table.

Changed properties

Change the default entries in the column header column to more meaningful titles, adjust the width of the columns and specify the alignment of the header, e.g. for the first column

Columns

[-] Column

[-] [0]

Column header

units ordered

Column width

150

Text in headline

LEFT

There are two options to change the column width: 1. In the element properties: Enter the width in **Column ► Width** or 2. directly in the table element: Click on the separator between two column in the header and drag it to the desired position.

Set the column height to 20 pixels and disable the line labeling so that now the first column that contains the array indexes is missing. The image to the right also shows the now wider scrollbars (**Scrollbar size**), which are inserted in the table as soon as their size decreases so that it is less than the comprehensive line heights or column widths.

Columns

+ Column

Row header

☐

Column header

☒

Row height

20

Row header width

15

Scrollbar size

40

Resulting table

	units ordered	on stock?	product code
0			
1			
2			
3			

ordered	on stock?	product code

IndraLogic 2G visualization

Changed properties	Resulting table																																																																				
The table fields can be edited line-by-line: For the column selected, tick the Use template check box; then, within the column properties another Template is inserted and expanded: <table><tr><td>Use template</td><td>☒</td></tr><tr><td>Use text alignment in heading</td><td>☐</td></tr><tr><td colspan="2">Template</td></tr><tr><td>Type of element</td><td>VISU_ST_RECTANGLE</td></tr><tr><td colspan="2">Colors</td></tr><tr><td colspan="2">Normalstate</td></tr><tr><td>Framecolor</td><td> Coral</td></tr><tr><td>Fillcolor</td><td> Bisque</td></tr><tr><td colspan="2">Alarmstate</td></tr><tr><td>Framecolor</td><td> OrangeRed</td></tr><tr><td>Fillcolor</td><td> LightSalmon</td></tr><tr><td colspan="2">Elementlook</td></tr><tr><td>Linewidth</td><td>1</td></tr><tr><td>Fillattributes</td><td>BS_SOLID</td></tr><tr><td>Frameattributes</td><td>PS_DASH</td></tr><tr><td colspan="2">Texts</td></tr><tr><td>Text</td><td>%s</td></tr><tr><td>Tooltip</td><td></td></tr><tr><td colspan="2">Text properties</td></tr><tr><td>HorizontalAlignment</td><td>RIGHT</td></tr><tr><td>VerticalAlignment</td><td>VCENTER</td></tr><tr><td>Font</td><td> Copperplate Gothic Light; 1</td></tr><tr><td colspan="2">Text variables</td></tr><tr><td>Text Variable</td><td>PLC_PRG.arrStruct[INDEX].iNo</td></tr></table> Three types of templates are available: by default, a rectangle is entered that can be changed to a rectangle with rounded corners or an ellipse by clicking in the corresponding field. The configuration options display the element properties of the selected template: select the fill and frame colors for the normal and alarm states. They can also be toggled with a variable. The alignment of the entries in the column cells is now specified by the template setting (with the exception of the heading). Note: Instead of the array components entered by default, any other variable of the project can be entered in the Text variable field. In this way it is possible to display the array elements in the table in a different order. The columns can be exchanged in the table. Within the title header, click in the column center and drag and drop the column to its new position.	Use template	☒	Use text alignment in heading	☐	Template		Type of element	VISU_ST_RECTANGLE	Colors		Normalstate		Framecolor	 Coral	Fillcolor	 Bisque	Alarmstate		Framecolor	 OrangeRed	Fillcolor	 LightSalmon	Elementlook		Linewidth	1	Fillattributes	BS_SOLID	Frameattributes	PS_DASH	Texts		Text	%s	Tooltip		Text properties		HorizontalAlignment	RIGHT	VerticalAlignment	VCENTER	Font	 Copperplate Gothic Light; 1	Text variables		Text Variable	PLC_PRG.arrStruct[INDEX].iNo	In the example, a template was used for the first and third columns, where the template of the first column has the properties displayed to the left. <table><tr><th></th><th>units ordered</th><th>on stock?</th><th>product code</th></tr><tr><td>0</td><td>250</td><td>TRUE</td><td>DD32F7098367</td></tr><tr><td>1</td><td>480</td><td>FALSE</td><td>DD33F1782648</td></tr><tr><td>2</td><td>1500</td><td>TRUE</td><td>DD45G2637526</td></tr><tr><td>3</td><td>2500</td><td>TRUE</td><td>AD70F1829388</td></tr></table>		units ordered	on stock?	product code	0	250	TRUE	DD32F7098367	1	480	FALSE	DD33F1782648	2	1500	TRUE	DD45G2637526	3	2500	TRUE	AD70F1829388
Use template	☒																																																																				
Use text alignment in heading	☐																																																																				
Template																																																																					
Type of element	VISU_ST_RECTANGLE																																																																				
Colors																																																																					
Normalstate																																																																					
Framecolor	 Coral																																																																				
Fillcolor	 Bisque																																																																				
Alarmstate																																																																					
Framecolor	 OrangeRed																																																																				
Fillcolor	 LightSalmon																																																																				
Elementlook																																																																					
Linewidth	1																																																																				
Fillattributes	BS_SOLID																																																																				
Frameattributes	PS_DASH																																																																				
Texts																																																																					
Text	%s																																																																				
Tooltip																																																																					
Text properties																																																																					
HorizontalAlignment	RIGHT																																																																				
VerticalAlignment	VCENTER																																																																				
Font	 Copperplate Gothic Light; 1																																																																				
Text variables																																																																					
Text Variable	PLC_PRG.arrStruct[INDEX].iNo																																																																				
	units ordered	on stock?	product code																																																																		
0	250	TRUE	DD32F7098367																																																																		
1	480	FALSE	DD33F1782648																																																																		
2	1500	TRUE	DD45G2637526																																																																		
3	2500	TRUE	AD70F1829388																																																																		

Changed properties	Resulting table																																		
The alarm state can not only be triggered by a variable value, but also by highlighting a table cell by clicking on it in online mode. The behavior of the selected region can be specified in the Selection section: Selection <table><tr><td>Selection color</td><td> HotPink</td></tr><tr><td>Apply to columns</td><td> Apply </td></tr><tr><td>Selection type</td><td>SEL_ROW</td></tr><tr><td>Frame around selected cells</td><td> </td></tr><tr><td>Variable for selection X</td><td>PLC_PRG.SelectedPosX</td></tr><tr><td>Variable for selection Y</td><td>PLC_PRG.SelectedPosY</td></tr><tr><td>Variable for valid selection</td><td>PLC_PRG.IsSelected</td></tr></table> The specified background color for highlighted elements applies to all highlighted cells for which no alarm color was specified (within a template). So in our example, a highlighted cell in the second column is filled with hot pink, while a highlighted cell in the first column is filled with light salmon, the alarm color of the associated template.	Selection color	HotPink	Apply to columns	Apply	Selection type	SEL_ROW	Frame around selected cells		Variable for selection X	PLC_PRG.SelectedPosX	Variable for selection Y	PLC_PRG.SelectedPosY	Variable for valid selection	PLC_PRG.IsSelected	The table element in online mode: After the cell with the entry FALSE is selected, all cells in the second line are highlighted. Only the highlighted cell in the second column has the selected color because the color for the alarm state in the other columns is set by the templates: <table><tr><td></td><td>units ordered</td><td>on stock?</td><td>product code</td></tr><tr><td>0</td><td>250</td><td>TRUE</td><td>DD32F7098367</td></tr><tr><td>1</td><td>480</td><td>FALSE</td><td>DD33F1782648</td></tr><tr><td>2</td><td>1500</td><td>TRUE</td><td>DD45G2637526</td></tr><tr><td>3</td><td>2500</td><td>TRUE</td><td>AD70F1829388</td></tr></table>		units ordered	on stock?	product code	0	250	TRUE	DD32F7098367	1	480	FALSE	DD33F1782648	2	1500	TRUE	DD45G2637526	3	2500	TRUE	AD70F1829388
Selection color	HotPink																																		
Apply to columns	Apply																																		
Selection type	SEL_ROW																																		
Frame around selected cells																																			
Variable for selection X	PLC_PRG.SelectedPosX																																		
Variable for selection Y	PLC_PRG.SelectedPosY																																		
Variable for valid selection	PLC_PRG.IsSelected																																		
	units ordered	on stock?	product code																																
0	250	TRUE	DD32F7098367																																
1	480	FALSE	DD33F1782648																																
2	1500	TRUE	DD45G2637526																																
3	2500	TRUE	AD70F1829388																																

Changed properties	Resulting table																																							
When pressing Apply, the alarm colors of special templates are replaced by the selected color, selected under Selection color. As a result, the following entries are made in the corresponding fields in the template for the first column of our example: Template Type of elementVISU_ST_RECTANGLE Colors Normalstate FramecolorCoral FillcolorBisque Alarmstate FramecolorOrangeRed FillcolorHotPink	After applying the selected color to the templates as well, the highlighting looks different: <table><tr><th>units ordered</th><th>on stock?</th><th>product code</th></tr><tr><td>0250</td><td>TRUE</td><td>DD32F7098367</td></tr><tr><td>1480</td><td>FALSE</td><td>DD33F1782648</td></tr><tr><td>21500</td><td>TRUE</td><td>DD45G2637526</td></tr><tr><td>32500</td><td>TRUE</td><td>AD70F1829388</td></tr></table>	units ordered	on stock?	product code	0250	TRUE	DD32F7098367	1480	FALSE	DD33F1782648	21500	TRUE	DD45G2637526	32500	TRUE	AD70F1829388																								
units ordered	on stock?	product code																																						
0250	TRUE	DD32F7098367																																						
1480	FALSE	DD33F1782648																																						
21500	TRUE	DD45G2637526																																						
32500	TRUE	AD70F1829388																																						
Selection type determines which elements are selected by selecting individual table cells: after line selection was selected, and so all of the cells that are in the same line as the selected cell are highlighted and framed. If no columns of the display in the table were excluded, the position of the selected cells within the table represents the indexes of the visualized array associated with the entry. These indexes can be saved in variables that are entered in the last three lines of the Selection section for this purpose. The third and fourth variable is Boolean and is set to TRUE as long as the selection is valid, regarding the line or column.	The array index related to the selected cell is assigned to the associated variables as a value: <table><tr><th>Expression</th><th>Type</th><th>Value</th></tr><tr><td> arrDim1</td><td>ARRAY [2..5] OF INT</td><td></td></tr><tr><td> arrDim2</td><td>ARRAY [0..2, 0..3] O...</td><td></td></tr><tr><td> varStruct</td><td>MyStruct</td><td></td></tr><tr><td> arrStruct</td><td>ARRAY [0..3] OF My...</td><td></td></tr><tr><td> varFB</td><td>TON</td><td></td></tr><tr><td> arrFB</td><td>ARRAY [0..3] OF TON</td><td></td></tr><tr><td> iCounter</td><td>INT</td><td>745</td></tr><tr><td> iRow</td><td>INT</td><td>0</td></tr><tr><td> iCol</td><td>INT</td><td>7</td></tr><tr><td> SelectedPosX</td><td>INT</td><td>2</td></tr><tr><td> SelectedPosY</td><td>INT</td><td>1</td></tr><tr><td> IsSelected</td><td>BOOL</td><td>TRUE</td></tr></table>	Expression	Type	Value	 arrDim1	ARRAY [2..5] OF INT		 arrDim2	ARRAY [0..2, 0..3] O...		 varStruct	MyStruct		 arrStruct	ARRAY [0..3] OF My...		 varFB	TON		 arrFB	ARRAY [0..3] OF TON		 iCounter	INT	745	 iRow	INT	0	 iCol	INT	7	 SelectedPosX	INT	2	 SelectedPosY	INT	1	 IsSelected	BOOL	TRUE
Expression	Type	Value																																						
 arrDim1	ARRAY [2..5] OF INT																																							
 arrDim2	ARRAY [0..2, 0..3] O...																																							
 varStruct	MyStruct																																							
 arrStruct	ARRAY [0..3] OF My...																																							
 varFB	TON																																							
 arrFB	ARRAY [0..3] OF TON																																							
 iCounter	INT	745																																						
 iRow	INT	0																																						
 iCol	INT	7																																						
 SelectedPosX	INT	2																																						
 SelectedPosY	INT	1																																						
 IsSelected	BOOL	TRUE																																						

(not yet implemented: \$ROWCONST\$, \$COLCONST\$, invisible, deactivate inputs)

“Text field” element

Using this element, text can be displayed that is directory assigned in the properties or variable in **Text variable**. In contrast to the rectangle, the frame can be displayed with shadows.

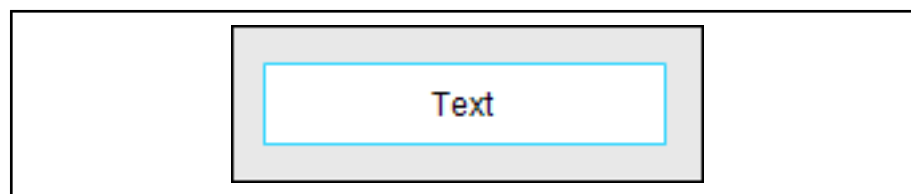


Fig. 20-51: Text field

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Text field

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Colors

A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are

available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Normal state - Frame color - Fill color	Select a frame and fill color for normal state. In case that the variable is defined in Color variables ► Toggle color is FALSE, the element is in normal state.
Alarm state - Frame color - Fill color	Select a frame and fill color with which the element is displayed in alarm condition. This is the case if the variable in Color variables ► Toggle color is TRUE.

Appearance

Property	Value description
Line width	Weight of frame line in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is required, set Line type to <i>invisible</i> .
Fill type	Select one of the fill types used in Fill color - Filled: The fill color is visible - Invisible: The fill color is invisible
Line style	<i>Select one of the line types used in the contour:</i> - Solid - Dashes - Points - Dash dot - Dash dot dot - Invisible: Contour invisible.

Shadow type	<i>Shadow type of contour</i> - Sunken - No shadow - Elevated - Of style

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Element properties "Color variables"

The element property facilitates dynamic switching of the element colors using IEC variables for control and colors.

The IEC variable for color contains this as hexadecimal color definitions that codes all red/green and blue components of the color in the RGB color space: `16#00RRGGBB`. A byte for 256 color grading (from 0 to 255) is reserved for each color component. 0 indicates that no color portion is contained. 255 refers to 100% pigment content. The leading byte is always `16#00`.

Example:

Color variable

```
dwFillColor := 16#008FE03F;
```

Coding: Red component: 8F, green component: E0, blue component: 3F

- Grey: dwGrey := 16#00888888
- Red: dwRed := 16#00FF0000
- Black: dwBlack := 16#00000000

Toggle color	<i>Control variable</i> • IEC variable of type BOOL, e.g. <code>prgA.bToggleColor</code> At runtime, the visualization dynamically controls the color display of the element – TRUE: Element in Normal state, e.g. in gray with a black frame – FALSE: Element in Alarm condition, e.g. in yellow with red frame. • or as placeholder <Toggle variable/key variable> No individual variable assigned to toggle color. Instead, the toggle color is controlled using the variable assigned in Input configuration, Toggle or Keys. Requirement: a control variable has to be configured. If a control variable is defined in Shift as well as Keys, the variable specified in Keys is used. Remove the placeholder by selecting <Toggle variable/key variable> and by selecting <Del>.  is displayed in the value field.
Click on  into the value field of Toggle color	IndraLogic inserts the placeholder <Toggle variable/key variable> .
Normal state	Color variables of element during smooth operation
• Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColor</code>
• Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColor</code>
Alarm state	Color variables of element during an alarm
• Frame color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwBorderColorAlarm</code>
• Fill color	IEC variable of type DWORD for frame color, e.g. <code>prgA.dwFillColorAlarm</code>

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys

Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle

Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration' on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

Element "Scroll bar"

When you drag the scroll box to another position at visualization runtime, the visualization sets the **Value** property accordingly.

Scroll Bar element in Flat style:



Fig. 20-52:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input:

- When dragging and dropping a scrollbar to the visualization editor, the scroll bar is displayed as horizontal. Change the size using the mouse. As soon as the height exceeds the width, the scroll bar is displayed as vertical and IndraLogic updates the relevant property **Bar ► Alignment** to **Vertical**.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Scroll bar

Value	IEC variable with integer data type containing the scroll bar position in the scroll bar
Minimum value	Smallest value for the scroll box
Maximum value	Largest value for the scroll box

Page size	<i>Page size</i> - As a fixed value, for example 10 - As an IEC variable of data type integer. Click on  to open the input assistance. Prerequisite: Visible when the Move to click property is not selected.
Move to click	Function of the scrollbar at visualization runtime when it is clicked: : The scroll bar moves to the clicked position. : The scroll bar moves to one Page size in the direction of the click.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Bars"

Alignment	<i>Alignment</i> Horizontal: The bar is horizontal Vertical: The bar is vertical Modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the scroll bar.
Direction of motion	The drop-down list depends on the scroll bar alignment. <i>Horizontal</i> From left to right: Scale starts at the left. Right to left: Scale starts at the right. <i>Vertical</i> Bottom to top: Scale start bottom. From left to right: Scale start top.

Colors A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:
 0; 128; 64

Property	Value description
Color	Select a color to fill the element in normal state. This is the case if the variable in Color variables ► Toggle color is FALSE. By editing the value, the element changes color.
Alarm color	Select a color to fill the element in alarm condition. This is the case if the variable in Color variables ► Toggle color is TRUE.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Text properties

These properties are used for static definition of the font.

Property	Value description
Orientation horizontal	<i>Define the horizontal text alignment:</i> • Left • Centered • Right
Orientation vertical	<i>Define the vertical text alignment:</i> • Top • Centered • Down
Fonts	<i>Selection from predefined fonts:</i> • Default • Heading • Large • Title • Note Click on  to open the dialog for user-defined properties of the font.
Font color	Select a font color for the element. Select either from the drop-down list or by the setting in the dialog that opens when clicking on  .

"Text variables" element property

Text saved in a variable can be displayed. First add a formatting sequence to the text, defined under the **Texts** property. Assign a variable. In online mode, the formatting sequence is replaced by the variable content ([chapter "Using text and language " on page 796](#)).

Example:

Example %f

Enter `result: %2.5f` into **Text** property. Enter `rValue` into the **Text variable** property. An IEC variable has to be defined. In online mode, the element is labeled with `result: 12.12345` if `rValue = 12.12345`.

Text variable	Enter a variable (standard data type) that contains the information to be displayed. The type has to be compatible with the formatting sequence in Texts ► Text .
Tooltip variable	Enter a STRING variable containing the tooltip text. The input in Texts ► Tooltip has to contain %s.

Dynamic texts These parameters are used to define dynamic texts generated by test lists (this allows language switching) ([chapter 18.2.15 "Menu 'Text list'" on page 684](#)).

Property	Value description
Text list	Name of the text list (string) as used in POU's or device windows, e.g. <code>errorlist</code>
Text index	Index (ID) of the text as defined in the text list (string); direct ID specification, e.g. 'M1' or string variable, e.g. <code>error_22</code> .
Tooltip index	Index (ID) of the tool tip text is defined in the text list (string); direct ID specification, e.g. 'T1' or string variable, e.g. <code>error_Tooltip_22</code> .

Element property 'Font variables' Font variables are used in dynamic font definitions of the element text via project variables. Static definitions are configured under **Text properties ► Font**.

Font name	Enter a variable of type STRING containing the font to be used to label the element. (Name specification as in standard font type dialog). Example: <code>PlcProg.stFontVar := 'Arial';</code>
Size	Entering a variable of type INT containing the size of the element text in pixel as well the font used in the standard dialog. Example: <code>PlcProg.iFontHeight := 16;</code>
Flags	Entering a variable of type DWORD that specifies the font representation by means of the flag values mentioned in the following. A combined definition can be achieved by adding the respective flag values and entering the sum. • 1: Italics • 2: Bold • 4: Underlined • 8: Deleted Example of a text that is bold and underlined: <code>PlcProg.dwFontType := 6;</code>
Character set	The character set to be used for the font can be defined using the default character set number. A DWORD variable can be used to input this number (see also the Script definition in the default font dialog)

Color	Entering a <code>DWORD</code> variable in which the element text color is specified.
Flags for text alignment	Variable with integer data type such as <code>PlcProg.dwTextAlignment</code>. At runtime, the visualization analyzes the variable and aligns the text accordingly. <i>Text alignment</i> • 0: Top left • 1: Horizontally centered • 2: Right • 4: Vertically centered • 8: Bottom

Element properties "Color variables"

Toggle color	Boolean variable to toggle the color of the element. <code>TRUE</code> : The visualization displays the element in the alarm color.
Color	Variable of type <code>DWORD</code> for the element color
Alarm color	Variable of type <code>DWORD</code> for the alarm color

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is <code>TRUE</code> , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is <code>TRUE</code>, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element "Slider"

In a visualization, use the mouse to move the slider along the slider bar. A value is assigned to a variable that corresponds to the position of the slider in the bar. You can define the value range of the slider bar in the **Scale start** and **Scale end** properties.

Slider element in `Flat` style:

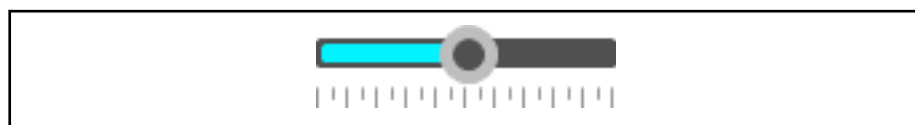


Fig. 20-53:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input:

- If the slider is moved to the visualization editor, it is displayed as horizontal. Change the size using the mouse. As soon as the height ex-

ceeds the width, the slider is displayed as vertical and IndraLogic updates the relevant property **Bar ► Alignment** to *Vertical*.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Slider

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	IEC variable with any numeric data type. The visualization assigns the value to the slider at runtime to correspond with the position of the slider in the bar. Click on  to open the input assistance.
Page size	<i>Page size</i> - As a fixed value, for example 10 - As an IEC variable of data type integer. Click on  to open the input assistance. Prerequisite: Visible when the Move to click property is not selected.
Move to click	Function of the slider at visualization runtime when it is clicked: : The slider moves to the clicked position. : The slider moves to one Page size in the direction of the click.

Element property "Scale"

Show scale	 : The slider has a visible scale.
Scale start	Value at start of scale and lower limit of the value range of the slider bar. For example: 0 You can define the value range of the slider bar in the Scale start and Scale end properties.
Scale end	Value at end of scale and upper limit of value range of the slider bar. For example: 100
Main scale	Distance between two scale lines of the main scale (example: 10)
Subscale	Distance between two lines on the fine scale, e.g. 2. By setting the value to 0, the fine scale can be hidden.
Scale format (C Syntax)	Formatting of the scale labeling, e.g. %d s

Element property "Bars"

Diagram type	The list box varies depending on the alignment of the diagram. <i>Horizontal</i> • Top: Scale is above the slider • Down: Scale is below the slider • Top and down: Two scales frame the slider top and bottom <i>Vertically</i> • Left: Scale is left of the slider • Right: Scale is right of the slider • Left and right: Two scales frame the slider from left to right
Alignment	<i>Slider alignment</i> • Horizontal • Vertical Modify the alignment in the visualization editor by using the pointing device to adjust the width and height of the scroll bar. The element is displayed horizontally or vertically depending on the width/height ratio.
Direction of motion	The list box varies depending on the alignment of the slider. <i>Horizontal</i> • From left to right: Scale starts at the left. • Right to left: Scale starts at the right. <i>Vertical</i> • Bottom to top: Scale start bottom. • From left to right: Scale start top.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"SpinControl" element

Clicking the arrow buttons of the element at visualization runtime either increments or decrements the output value of the element. You can configure the output value in the property **Variable**.



Fig. 20-54:

Slider element in Flat style

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	SpinControl

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	IEC variable. Click on the arrow buttons at visualization runtime to either increment or decrement the IEC variable by the value in the property Interval .
Number format	Format to display the value. For example <code>Value: %i</code> for a variable of type <code>INT</code>
Interval	Interval for incrementing or decrementing the variable

Element property 'Value range'

Minimum value	<i>Lower limit of the output value</i> - As fixed value - IEC variable of type <code>INT</code>:
Maximum value	<i>Upper limit of the output value</i> - As fixed value - IEC variable of type <code>INT</code>

Element property 'Text properties'

Usage of	Default style values: The element text is displayed with default style properties. Individual settings: The element text is displayed according to the following individual properties.

These properties are available only if you select the option **Individual settings** (property **Usage of**).

Orientation horizontal	Left Right Centered
Orientation vertical	Left Right Centered
Fonts	: The dialog opens to select a font - The list box offers the font style properties defined in the style
Font color	<i>Color of the font</i> : The dialog opens to select a color - The list box offers the color style properties defined in the style
Transparency	Value (0-255) of transparency of font color

Element properties "Color variables"

Toggle color	Boolean variable to toggle the color of the element. The element color and alarm color are defined in the visualization style. <code>TRUE</code> : The visualization displays the element in the alarm color.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Input configuration'

Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.
OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Hotkey

Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configuration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> - No action - MouseDown action: OnMouseDown action when the key is pressed. - MouseUp action: OnMouseUp action if the key is released. - MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

"Invisible input" element

The element is displayed as dashed line in the editor but in online mode, the line is invisible. Specify the element behavior in the input configuration.

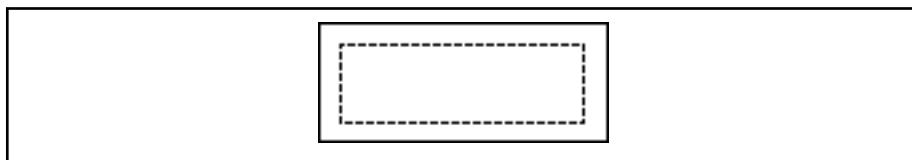


Fig. 20-55: Invisible input

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Invisible input

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

State variables

Disable inputs	Assign a Boolean variable. If the variable value is TRUE, the user input does not have any effect on the element.

Element property 'Input configuration' Define here which follow-up actions are executed if the user makes an entry in the element during online mode.

Input events with an element:

- OnDialogClosed
- OnMouseClicked
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

Click on **Configure...** to open the [chapter "Input configuration" on page 800](#) dialog in which follow-up actions can be assigned. Click in the field next to the input to be configured. For each selected follow-up action, an input is generated in the properties table. Each input can have a user-defined number of follow-up actions.

OnDialogClosed	This event occurs if a dialog window opened for user input in the visualization is closed.
OnMouseClicked	This mouse event occurs if the cursor points to an element and a mouse-click (pressing and releasing the mouse key) is executed.

OnMouseDown	This mouse event occurs if the mouse key is pressed while the cursor points to an element. (Pressing the mouse key)
OnMouseEnter	This mouse event occurs if the cursor is dragged to the element.
OnMouseLeave	This mouse event occurs if the cursor leaves the element.
OnMouseMove	This mouse event occurs if the cursor is moved within the element.
OnMouseUp	This mouse event occurs if the mouse key is released. The previously executing pressing of the mouse key was then however executed outside of the element.

Input configuration: Keys Use **Tap** to configure that upon the "Tap" event, the value of a Boolean variable is set depending on the mouse behavior.

Property	Value description
Variable	To enter a Boolean variable with the value TRUE, press the mouse button while the cursor is on the element. The value becomes FALSE again when releasing the mouse button.
FALSE keys	Enabling this option reverses the behavior described above for the variable entered. That means, if the mouse button is pressed, the variable value is set to FALSE and when the button is released again, the variable value is set to TRUE.
Tap upon entering if mouse caught	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse: If the element area is left while the mouse key is still pressed, the variable turns FALSE and TRUE again when returning to the element area. Thus, it is considered that the mouse is "caught" as long as the mouse button is pressed even if the element area is left.

Input configuration: Toggle Use **Toggle** to configure that upon this event, the value of a Boolean project variable is set irrespective of the mouse behavior.

Property	Value description
Variable	Assign a Boolean variable. Input assistance opened with the  button. The variable value is toggled between TRUE and FALSE upon each mouse click. If the cursor is moved outside the element while the mouse key is pressed, toggling does not take place. Thus, the started toggling input can be canceled.
Toggle upon release if mouse freezes	If this option is enabled, the variable value behaves as described under Variable as long as the mouse is within the element area. Additionally, it is reacted to dragging the mouse. If the element area is left while the mouse key is still pressed and if then the mouse key is released, the variable is toggled. Thus, it is taken into consideration that the mouse "freezes" as long as the mouse key is pressed even if the element area is left.

Input configuration: Hotkey Use **Hotkey** to define a key, also with an additional key, and link the key to a follow-up action (MouseDown, MouseUp) to be executed upon an event of the key (KeyDown, KeyUp).

By default, KeyDown is used to execute the MouseDown action and KeyUp for the MouseUp action.

This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This key configuration of an element is also managed under the "Key configu-

ration" tab of the visualization ([chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#)). Changes are always synchronized between this editor and the "element properties" editor.

Property	Value description
Key	Key assignment. A drop-down list provides the currently supported key such as <M>.
Input(s)	Define the event to be executed if the key or key and modifiers are to be used. <i>Possible values provided in a drop-down list:</i> • No action • MouseDown action: OnMouseDown action when the key is pressed. • MouseUp action: OnMouseUp action if the key is released. • MouseDown/MouseUp action: OnMouseUp and OnMouseDown action when the button is pressed or released.
Toggle	If this option is activated, the key must be used in combination with the <Shift> key
Control	If this option is activated, the key must be used in combination with the <Ctrl> key
Alt	If this option is activated, the key must be used in combination with the <Alt> key

"Progress bar" element

In a progress bar, the value of a variable is represented as bar. The variable limits as well as the layout can be set.

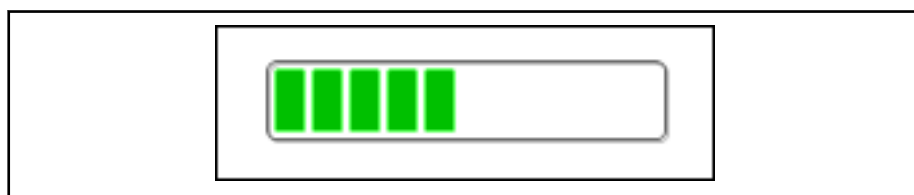


Fig. 20-56: Progress bar

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Progress bar

Variable	Assign a variable whose value is shown as progress bar.
Minimum value	Enter the smallest possible value.
Maximum value	Enter the largest possible value.
Style	<i>Selection:</i> • Blocks • Bars

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Texts" These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .

State variables

Property	Value description
Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.

"Radio button" element

A radio button facilitates the configuration of a user-defined number of options by using the  Create new interface to align it in one or several columns.

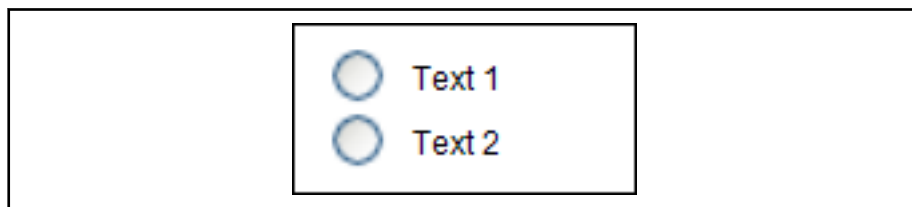


Fig. 20-57: Radio button

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Radio button

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The

positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Assign an integer variable that is written with the index of the selected selection button.
Number of columns	Enter the number of columns with which the radio button is to be assigned.
Radio button arrangement	From left to right: The radio buttons are aligned line-by-line until the number of columns is reached. From left to right: The radio buttons are aligned column-by-column until the number of columns is reached.

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property 'Radio button settings'

Radio button Areas [<n>]	Create new: Clicking this button creates a new selection button in the editor and lists an additional area in the properties editor. For each radio button, an area is visible that records the settings. [<n>]: The number indexes the area. Clicking Delete will delete the associated radio button with its settings Text, Tooltip, and Line spacing (in pixels).

Element property Radio button settings:

Text Radio button: Ranges [<n>]	The button name is specified here. Default value: Radio button
Tooltip	Text is specified here that is displayed in a tooltip.
Line spacing (in pixels)	The distance (in pixels) to the upper button can be specified here.

"Checkbox" element

Use a checkbox to set or reset a Boolean variable. If the box is ticked, the variable is set to TRUE.



Fig. 20-58: Checkbox

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Checkbox

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Texts" These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced ([chapter "Using text and language " on page 796](#)).

Text	Enter a text. It is used to label the element. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Text variable .
Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

20.4.4 'Measuring devices'

"Bar diagram" element, "Bar diagram freely configurable"

A bar diagram can be added to the visualization. **Bar diagram** has a predefined design in which you can select the background color. Use **Bar diagram freely configurable**, an image of an image pool can be assigned to the element as background. This is the only difference.

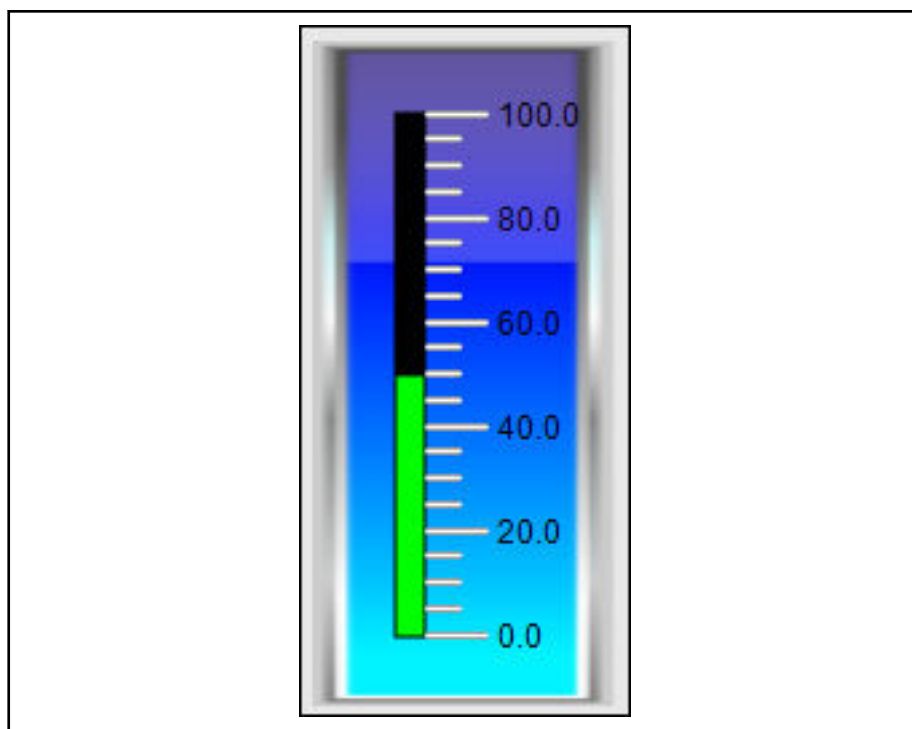


Fig. 20-59:

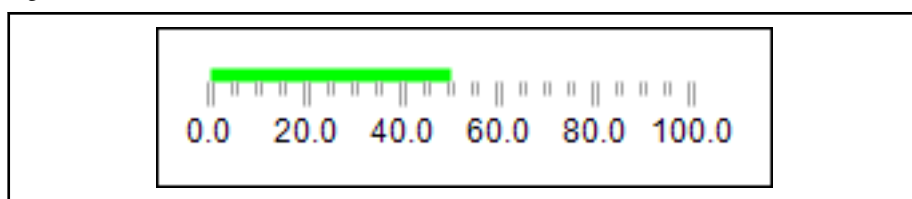


Fig. 20-60:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

Element-specific mouse input

- The bar is horizontal by default. If an element was added, its contour can be changed using the mouse. As soon as the element height exceeds the width, the bar is vertical and **Bar ► Alignment** is set to **Vertical**.

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	"Bar diagram" or "Bar diagram freely configurable"

Value	Assign a variable whose value is displayed as bar length.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Background" If no freely configurable **Bar diagram** is used, the **Background** has the following effect:

Background image	<i>Select a color for the bar:</i> • Yellow • Red • Green • Blue • Grey

If **Bar diagram freely configurable** is used, **Background** has the following effect:

Background image	Assign an image of the GlobalImagePool by specifying the image file name or its ID.

Element property "Bars"

Diagram type	<i>The drop-down list provides different options on how to position the bar and the scale:</i> • Scale besides bar • Scale in bar • Bar in scale • No scale.
Alignment	The bar can be aligned horizontally or vertically. The alignment is composed of the ratio width to height and cannot be edited here. This change is done in the visualization editor by "gripping" a corner point of the element using the mouse and drag in horizontal or vertical direction (chapter ""Bar diagram" element, "Bar diagram freely configurable"" on page 934).
Direction of motion	<i>In case of horizontal alignment, select from:</i> • Left to right • Right to left <i>In case of vertical alignment, select from:</i> • From left to right • From right to left

Element property "Scale"

Scale start	Value that provides the lower limit for the scaling display
Scale end	Value that provides the upper limit for the scaling display
Main scale	Distance between two lines on the large scale
Subscale	Distance between two lines on the fine scale. Distance between two lines on the fine scale; can be set to 0 if the large scale is not to be subdivided.
Scale line width	The defines the scale line width.
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.
Scale in 3D	If the option is enabled, the scale is displayed in 3D.
Element frame	If this option is enabled, a frame is drawn around the bar display.

Label

Property	Value description
Unit	The entered text labels the element. String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit).
Fonts	Determine the font for unit and scaling: <i>Select from:</i> • Default • Header • Large • Title • Annotation Click on  to open a dialog for user-defined font property settings.

IndraLogic 2G visualization

Property	Value description
Scale format (C Syntax)	Use the C-syntax to enter the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can be selected via the drop-down list or the  button in the color selection standard dialog.

Element property "Positioning"

Horizontal offset	Enter the distance in pixel of the scale/bar to the horizontal element frame.
Vertical offset	Enter the distance in pixel of the scale/bar to the vertical element frame.

Element property "Colors"

Graph color	Specify a color for the bar.
Bar background	Default: Bar diagram background is white. If the option is enabled, the background color is changed to black.
Frame color	Color of the frame that surrounds the bar diagram if the Element frame checkbox is enabled.
Use color scales	When this checkbox is selected the specified color scales are displayed as defined under Color scales
Switch whole color	If this option is enabled, the entire bar changes color if the variable value of the starting/end value of the scale is fallen below or exceeded.
Use gradient color for bar	If this option is selected the bar is displayed with color gradient.
Color range markers	<i>Select from</i> - No marking - Marking forward - Marking backwards
Color scales • [<n>]	Create new: Click on this button and a new color scale is created and listed in the properties.

Element property "Colors" Color scales: Areas [**<n>**]

Number of color scales in ascending order that cannot be edited.

After  **Delete** is clicked, the related color scale is deleted and the numbering is reassigned.

Begin of area	Value that provides the lower limit for the color scale
End of range	Value that provides the upper limit for the color scale
Color	Color in which the color scale is to be displayed

Element "Pointer instrument 90°"

Use the **pointer** (meter) to add a rev counter to the visualization. The needle moves acc. to the value of the assigned variable. It has a preconfigured design. A background color can be set.

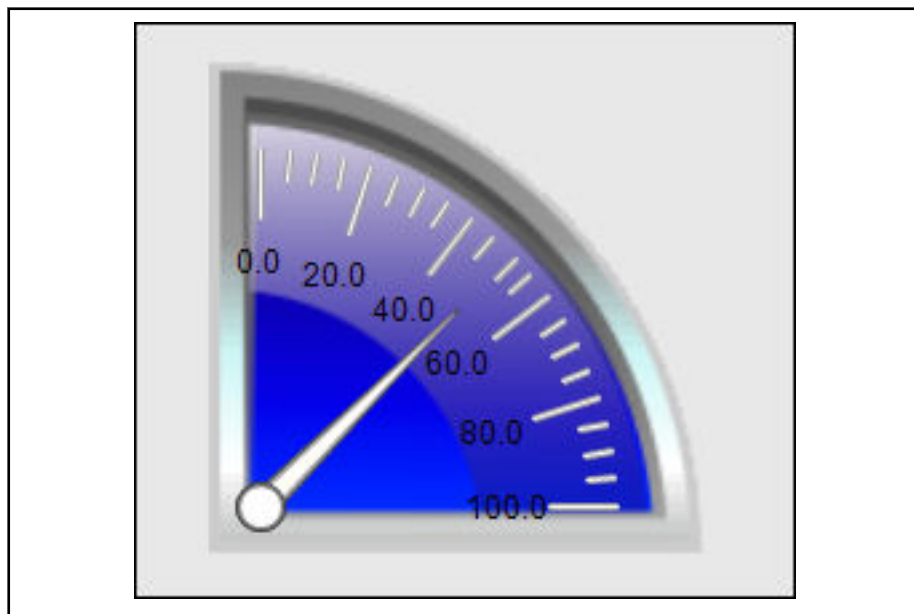


Fig. 20-61:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Pointer instrument 90°

Value	Assign a variable whose value is shown as needle deflection .

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Background"

Background image	<i>Select a color for the bar:</i> • Yellow • Red • Green • Blue • Grey

"Pointer" element property

Pointer type	<i>Select from:</i> • Normal pointer • Thin arrow • Thick pointer • Thin needle • Thin 3D pointer • Thin 3D pointer
Color	Color used to display the pointer.

Angular range	<i>Select a 90° view:</i> • Top right • Top left • Bottom left • Bottom right
Additional pointer	If the option is enabled, an additional pointer to the scale opposite the needle is visible.

Scale

Property	Value description
Subscale position	<i>The subscale can be displayed at the outer and inner radius of the scale ring:</i> • Outside • Inside
Scale type	<i>The scale can be displayed as:</i> • Lines • Points • Squares
Scale start	Value that provides the lower limit for the scaling display.
Scale end	Value that provides the upper limit for the scaling display.
Main scale	Distance between two lines on the large scale.
Subscale	Distance between two lines on the fine scale. It can be set to 0 if the large scale should not to be subdivided.
Scale line width	Width of scale in pixel
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.
Scale in 3D	If the option is enabled, the scale is displayed in 3D.
Show scale	If no scale view is displayed, disable this option.
Internal frame	If this option is enabled, the scale ring is displayed with an internal frame. Disabled by default.
External frame	If this option is enabled, the scale ring is displayed with an external frame. Disabled by default.

Label

Property	Value description
Label	<i>Select from</i> • Outside: Scale values are placed outside of the scale. • Internal: Scale values are placed inside of the scale.
Unit	String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit)

Property	Value description
Fonts	Font to display the unit and the scale range <i>Specify the font by selecting from:</i> • Default • Heading • Large • Title • Note Click on  to open a dialog for user-defined setting of the font.
Scale format (C Syntax)	Use the C-syntax to specify the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can selected via the drop-down list or the  button in the color selection standard dialog.

Element property "Positioning"

Pointer distance	Enter the pointer length in pixels.
Label offset	Enter a distance in pixel to position the label.
Unit offset	Enter a vertical distance to position the text entered in Label ► Unit .

Colors

Property	Value description
Color scales • Durable color scales • [<n>]	Durable color scales: The effect is only visible in online mode. If the option is enabled, the color scales are displayed permanently. If the option is disabled, only the color scale the arrow is pointing at is visible. Create new: By clicking on this button, a new color scale is created and displayed.

Colors: Color scales: Areas: [<n>] Number of color scales in ascending order that cannot be edited.

Delete: Click on the  button to remove the color scale from the list and to rearrange the order.

Property	Value description
Begin of area	Value that provides the lower limit for the color scale.
End of range	Value that provides the upper limit for the color scale.
Color	Color in which the color scale is to be displayed.

Element "Pointer instrument 180°"

Use the **pointer** (meter) to add a rev counter to the visualization. The needle moves acc. to the value of the assigned variable. It has a preconfigured design. A background color can be set.

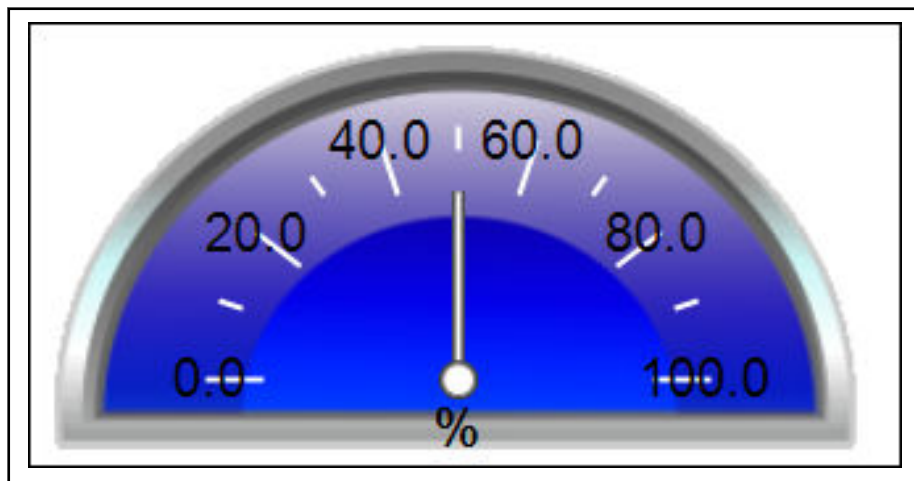


Fig. 20-62:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Pointer instrument 180°

Value	Assign a variable whose value is shown as needle deflection .

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Background"

Background image	<i>Select a color for the bar:</i> • Yellow • Red • Green • Blue • Grey

"Pointer" element property

Pointer type	<i>Select from:</i> • Normal pointer • Thin arrow • Thick pointer • Thin needle • Thin 3D pointer • Thin 3D pointer
Color	Color used to display the pointer.

Angular range	<i>Select a 180° view:</i> • Top • Down • Left • Right
Additional pointer	If the option is enabled, an additional pointer to the scale opposite the needle is visible.

Scale

Property	Value description
Subscale position	<i>The subscale can be displayed at the outer and inner radius of the scale ring:</i> • Outside • Inside
Scale type	<i>The scale can be displayed as:</i> • Lines • Points • Squares
Scale start	Value that provides the lower limit for the scaling display.
Scale end	Value that provides the upper limit for the scaling display.
Main scale	Distance between two lines on the large scale.
Subscale	Distance between two lines on the fine scale. It can be set to 0 if the large scale should not to be subdivided.
Scale line width	Width of scale in pixel
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.
Scale in 3D	If the option is enabled, the scale is displayed in 3D.
Show scale	If no scale view is displayed, disable this option.
Internal frame	If this option is enabled, the scale ring is displayed with an internal frame. Disabled by default.
External frame	If this option is enabled, the scale ring is displayed with an external frame. Disabled by default.

Label

Property	Value description
Label	<i>Select from</i> • Outside: Scale values are placed outside of the scale. • Internal: Scale values are placed inside of the scale.
Unit	String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit)

Property	Value description
Fonts	Font to display the unit and the scale range <i>Specify the font by selecting from:</i> • Default • Heading • Large • Title • Note Click on  to open a dialog for user-defined setting of the font.
Scale format (C Syntax)	Use the C-syntax to specify the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can selected via the drop-down list or the  button in the color selection standard dialog.

Element property "Positioning"

Pointer distance	Enter the pointer length in pixels.
Label offset	Enter a distance in pixel to position the label.
Unit offset	Enter a vertical distance to position the text entered in Label ► Unit .

Colors

Property	Value description
Color scales • Durable color scales • [<n>]	Durable color scales: The effect is only visible in online mode. If the option is enabled, the color scales are displayed permanently. If the option is disabled, only the color scale the arrow is pointing at is visible. Create new: By clicking on this button, a new color scale is created and displayed.

Colors: Color scales: Areas: [<n>] Number of color scales in ascending order that cannot be edited.

Delete: Click on the  button to remove the color scale from the list and to rearrange the order.

Property	Value description
Begin of area	Value that provides the lower limit for the color scale.
End of range	Value that provides the upper limit for the color scale.
Color	Color in which the color scale is to be displayed.

Element "Pointer instrument 360°"

Use the **pointer** (meter) to add a rev counter to the visualization. The needle moves acc. to the value of the assigned variable. It has a preconfigured design. A background color can be set.

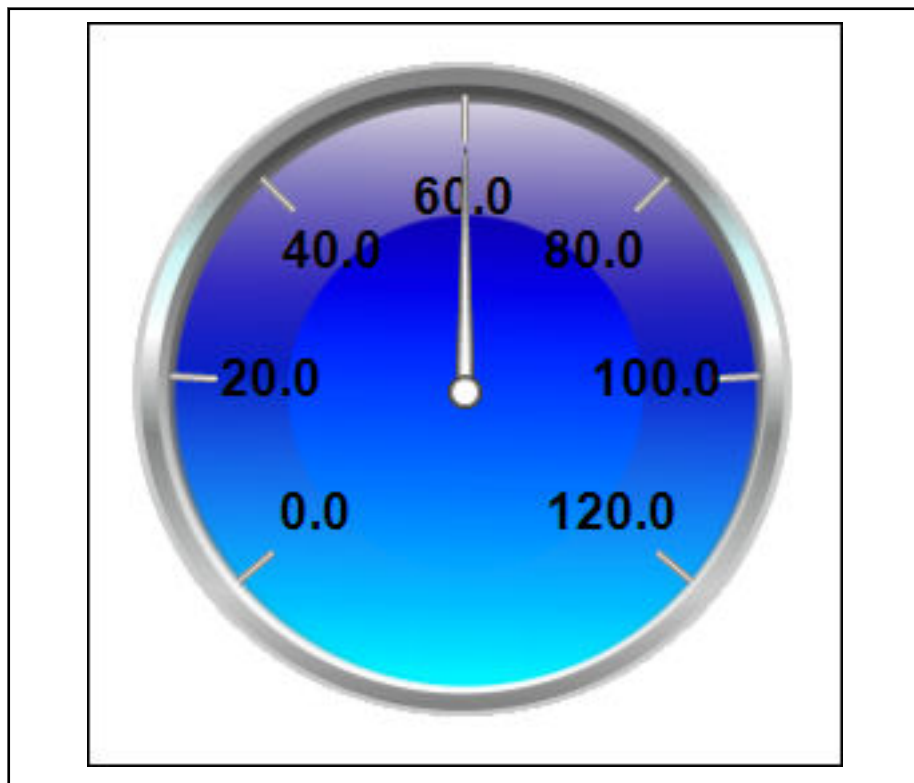


Fig. 20-63: Pointer instrument 360°

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.

- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.
[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Pointer instrument 360°

Value	Assign a variable whose value is shown as needle deflection .

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Background"

Background image	<i>Select a color for the bar:</i> • Yellow • Red • Green • Blue • Grey

Pointer

Property	Value description
Pointer type	<i>Select from:</i> • Normal pointer • Thin arrow • Thick pointer • Thin needle • Thin 3D pointer • Thin 3D pointer
Color	Color used to display the pointer.
Pointer start	The value to be entered here is interpreted as angle in degree. The angle spans from at the zero point of the scale to the start of the scale range in counterclockwise direction. The zero point of the scale is at "3 o'clock". The value range of the angle ranges from 0° to 360°. The center of rotation is the x/y-position.
Pointer end	The value to be entered here is interpreted as angle in degree. The angle spans from at the zero point of the scale to the end of the scale range in counterclockwise direction. The zero point of the scale is at "3 o'clock". A negative value can be entered. The center of rotation is the x/y-position. Pointer end has to be set so that from a clockwise point of view, it is to the right of pointer start . The value range of the angle ranges from 0° to max. 350°.
Additional pointer	If the option is enabled, an additional pointer to the scale opposite the needle is visible.

Scale

Property	Value description
Subscale position	<i>The subscale can be displayed at the outer and inner radius of the scale ring:</i> • Outside • Inside
Scale type	<i>The scale can be displayed as:</i> • Lines • Points • Squares
Scale start	Value that provides the lower limit for the scaling display.
Scale end	Value that provides the upper limit for the scaling display.
Main scale	Distance between two lines on the large scale.
Subscale	Distance between two lines on the fine scale. It can be set to 0 if the large scale should not to be subdivided.
Scale line width	Width of scale in pixel
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.
Scale in 3D	If the option is enabled, the scale is displayed in 3D.
Show scale	If no scale view is displayed, disable this option.

IndraLogic 2G visualization

Property	Value description
Internal frame	If this option is enabled, the scale ring is displayed with an internal frame. Disabled by default.
External frame	If this option is enabled, the scale ring is displayed with an external frame. Disabled by default.

Label

Property	Value description
Label	<i>Select from</i> • Outside: Scale values are placed outside of the scale. • Internal: Scale values are placed inside of the scale.
Unit	String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit)
Fonts	Font to display the unit and the scale range <i>Specify the font by selecting from:</i> • Default • Heading • Large • Title • Note Click on  to open a dialog for user-defined setting of the font.
Scale format (C Syntax)	Use the C-syntax to specify the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can be selected via the drop-down list or the  button in the color selection standard dialog.

Positioning

Property	Value description
Pointer distance	Enter the pointer length in pixels.
Label offset	Enter a distance in pixel to position the label.
Unit offset	Enter a vertical distance to position the text entered in Label ► Unit .
Origin offset	This value defines the element offset. It can be used to achieve exact positioning in relation to the background image.

Colors

Property	Value description
Color scales • Durable color scales • [<n>]	Durable color scales: The effect is only visible in online mode. If the option is enabled, the color scales are displayed permanently. If the option is disabled, only the color scale the arrow is pointing at is visible. Create new: By clicking on this button, a new color scale is created and displayed.

Colors: Color scales: Areas: [<n>] Number of color scales in ascending order that cannot be edited.

Delete: Click on the  **Delete** button to remove the color scale from the list and to rearrange the order.

Property	Value description
Begin of area	Value that provides the lower limit for the color scale.
End of range	Value that provides the upper limit for the color scale.
Color	Color in which the color scale is to be displayed.

“Potentiometer” element

A potentiometer with default design can be added to a visualization. A control element displays a variable value using arrow/pointer that can be moved using the user input.

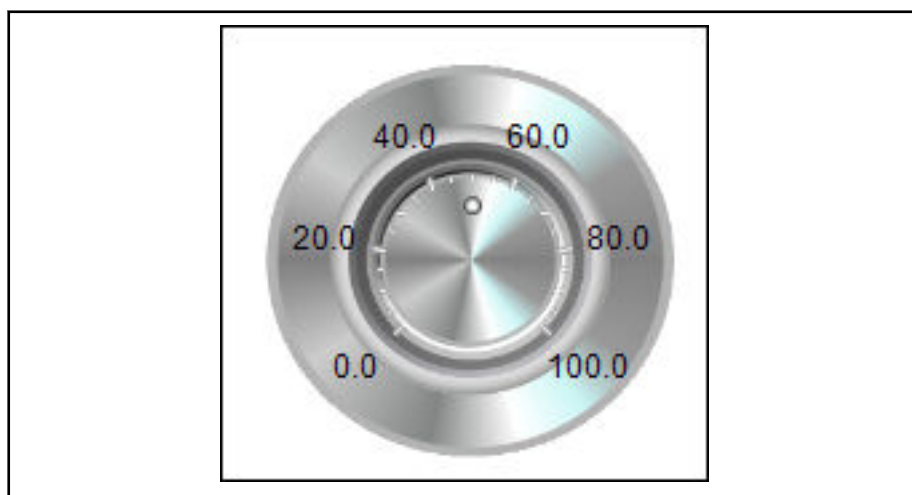


Fig. 20-64: Potentiometer

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemlnst_x", "x" indicates a continues numbering.

Element type	Potentiometer

Variable	Assign a numerical variable that contains the potentiometer position.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Background

Property	Value description
Background image	Assign an image from the GlobellImagePool by either specifying the name of the image file or the image ID.
Transparency color	Specify a color for images with a transparent background.

Pointer

Property	Value description
Pointer type	<i>Select from:</i> • Normal pointer • Thin arrow • Thick pointer • Thin needle • Thin 3D pointer • Thin 3D pointer
Color	Color used to display the pointer.
Pointer start	The value to be entered here is interpreted as angle in degree. The angle spans from at the zero point of the scale to the start of the scale range in counterclockwise direction. The zero point of the scale is at "3 o'clock". The value range of the angle ranges from 0° to 360°. The center of rotation is the x/y-position.
Pointer end	The value to be entered here is interpreted as angle in degree. The angle spans from at the zero point of the scale to the end of the scale range in counterclockwise direction. The zero point of the scale is at "3 o'clock". A negative value can be entered. The center of rotation is the x/y-position. Pointer end has to be set so that from a clockwise point of view, it is to the right of pointer start . The value range of the angle ranges from 0° to max. 350°.

Scale

Property	Value description
Subscale position	<i>The subscale can be displayed at the outer and inner radius of the scale ring:</i> • Outside • Inside
Scale type	<i>The scale can be displayed as:</i> • Lines • Points • Squares
Scale start	Value that provides the lower limit for the scaling display.
Scale end	Value that provides the upper limit for the scaling display.
Main scale	Distance between two lines on the large scale.
Subscale	Distance between two lines on the fine scale. It can be set to 0 if the large scale should not to be subdivided.
Scale line width	Width of scale in pixel
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.
Scale in 3D	If the option is enabled, the scale is displayed in 3D.
Show scale	If no scale view is displayed, disable this option.

IndraLogic 2G visualization

Property	Value description
Internal frame	If this option is enabled, the scale ring is displayed with an internal frame. Disabled by default.
External frame	If this option is enabled, the scale ring is displayed with an external frame. Disabled by default.

Label

Property	Value description
Label	<i>Select from</i> • Outside: Scale values are placed outside of the scale. • Internal: Scale values are placed inside of the scale.
Unit	String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit)
Fonts	Font to display the unit and the scale range <i>Specify the font by selecting from:</i> • Default • Heading • Large • Title • Note Click on  to open a dialog for user-defined setting of the font.
Scale format (C Syntax)	Use the C-syntax to specify the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can selected via the drop-down list or the  button in the color selection standard dialog.

Positioning

Property	Value description
Pointer distance	Enter the pointer length in pixels.
Label offset	Enter a distance in pixel to position the label.
Unit offset	Enter a vertical distance to position the text entered in Label ► Unit .
Origin offset	This value defines the element offset. It can be used to achieve exact positioning in relation to the background image.

Colors

Property	Value description
Color scales • Durable color scales • [<n>]	Durable color scales: The effect is only visible in online mode. If the option is enabled, the color scales are displayed permanently. If the option is disabled, only the color scale the arrow is pointing at is visible. Create new: By clicking on this button, a new color scale is created and displayed.

Colors: Color scales: Areas: [<n>] Number of color scales in ascending order that cannot be edited.

Delete: Click on the  button to remove the color scale from the list and to rearrange the order.

Property	Value description
Begin of area	Value that provides the lower limit for the color scale.
End of range	Value that provides the upper limit for the color scale.
Color	Color in which the color scale is to be displayed.

"Histogram" element

Page content:

- ["Element properties" on page 956](#)
- ["Configuration of a histogram" on page 958](#)

This element adds a histogram view to the visualization to display array values. The minimum and maximum display value can be specified. Specific colors can be defined for certain value ranges.

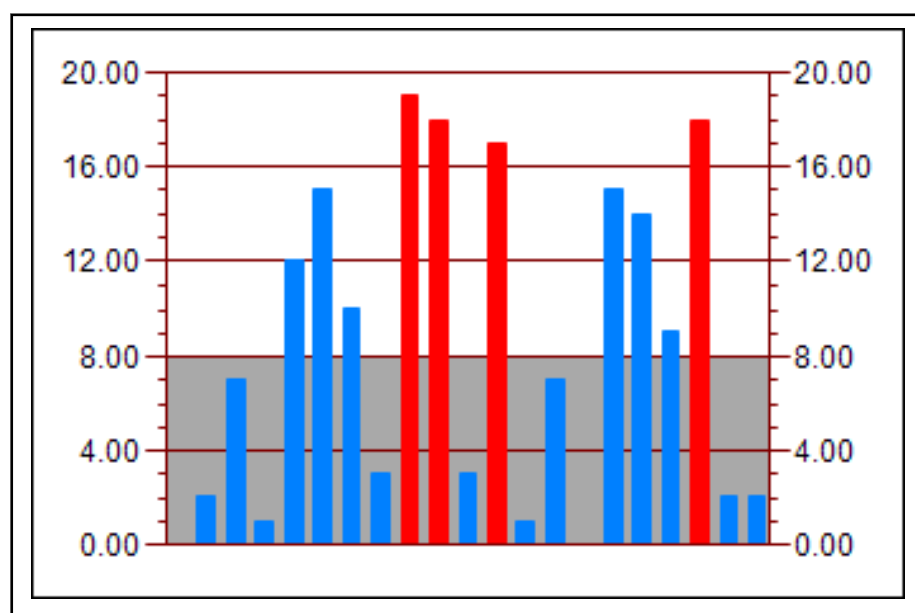


Fig. 20-65: Histogram

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Histogram

Data array	Assign a data array to be visualized.

**Element property
"Subrange of array"**

Use subrange	If this option is enabled, only the subrange of the assigned data array is used.
Start index	Indexes the first data set of the assigned data array.
End index	Indexes the last data set of the assigned data array.

Display type	• Bars • Lines • Curve
Line width	Prerequisite: Curve is selected as Display type . Line width of the curve in the histogram. Default setting: 1
Show horizontal lines	If the option is enabled, horizontal lines are drawn at the main scale.
Relative bar width	This value defines the relative bar width or the relative line width in pixels.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Scale"

Scale start	Value that provides the lower limit for the scaling display
Scale end	Value that provides the upper limit for the scaling display
Main scale	Distance between two lines on the large scale
Subscale	Distance between two lines on the fine scale. Distance between two lines on the fine scale; can be set to 0 if the large scale is not to be subdivided.
Scale color	Set the color from the drop-down list or press the button  from the standard color selection dialog.

Label

Property	Value description
Unit	The entered text labels the element. String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit).
Fonts	Determine the font for unit and scaling: <i>Select from:</i> • Default • Header • Large • Title • Annotation Click on  to open a dialog for user-defined font property settings.
Scale format (C Syntax)	Use the C-syntax to enter the formatting for the scale label. For example, when entering the string "%3.2f s" in this field to display the scale labels with 3 positions, out of which 2 are decimal positions, followed by the letter "s"
Max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.

Property	Value description
Text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.
Font color	The font color can be selected via the drop-down list or the  button in the color selection standard dialog.

Element property "Colors"

Graph color	Select a color for the graph:
Use color areas	If this option is selected, the specified color scales are shown as defined in Color scales .
Color scales [<n>]	Create new: Clicking on this button creates a new color scale. It is listed in the properties.

Colors: Color scales: Areas: [<n>] Number of color scales in ascending order that cannot be edited.

Delete: Click on the  button to remove the color scale from the list and to rearrange the order.

Property	Value description
Begin of area	Value that provides the lower limit for the color scale.
End of range	Value that provides the upper limit for the color scale.
Color	Color in which the color scale is to be displayed.

Configuration of a histogram

The following provides an example of inserting and configuring a histogram element:

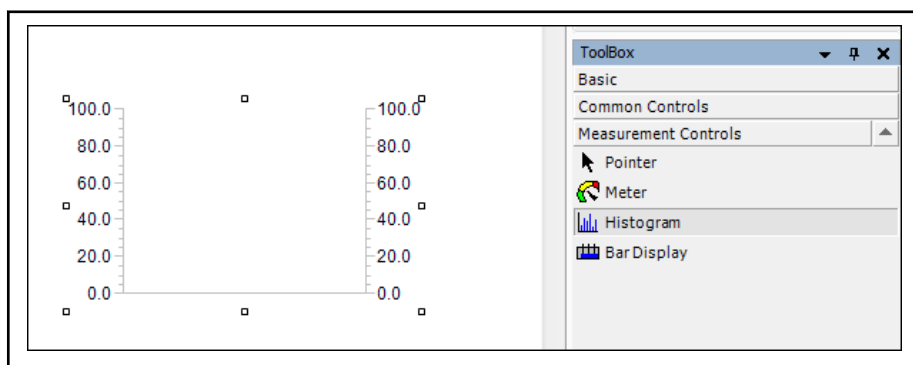


Fig. 20-66: Example - Inserting the histogram element

The assigned data array - in the example: an area by the name of `histogram` - has to be specified in the element properties of the histogram element. After you click in the input field of the **Data array** property, the button  is available. It can be used to browse the project for the variable. Make sure that you specify the input variable by entering its complete path in the device tree.

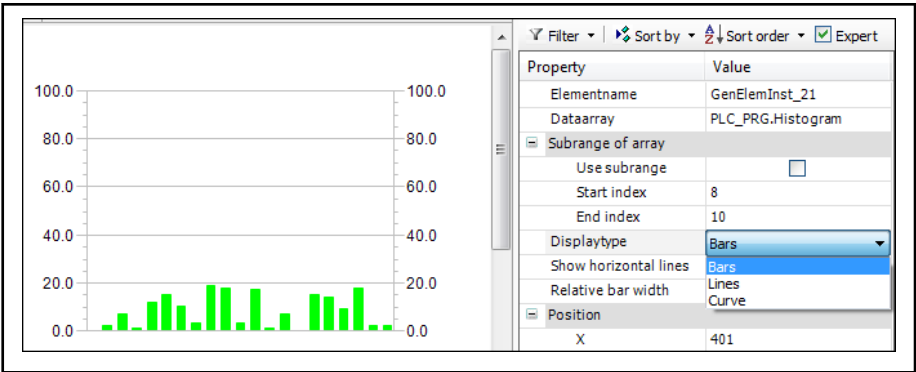
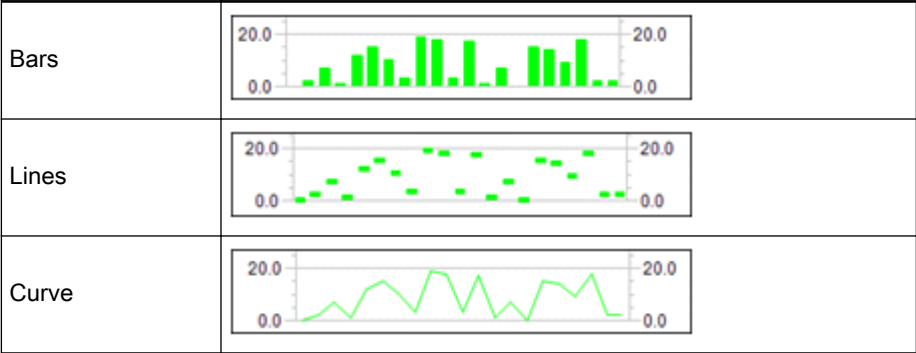


Fig. 20-67: Example - Configuration of the display type

Use subrange: This option is used if not all array elements are to be displayed but only elements of index **Start index** to **End index**.

Graph type: The data array can be displayed in three different forms:



Show horizontal lines: If the option is enabled, horizontal lines are drawn at the main scale.

Relative bar width: This value defines the relative bar width or the relative line width

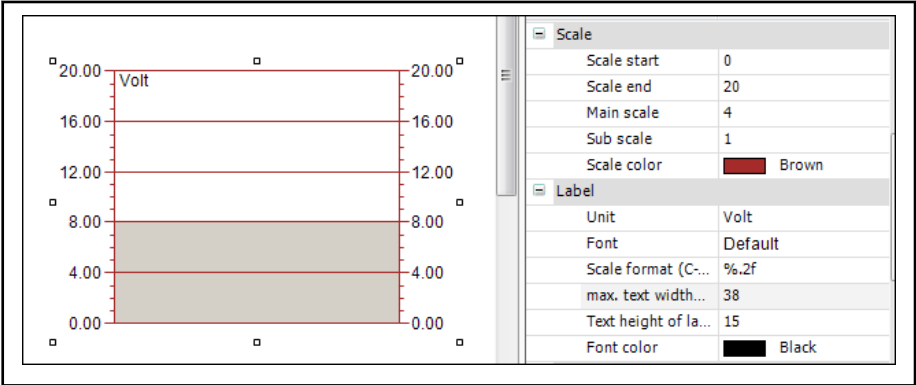


Fig. 20-68: Example - Scaling of the histogram

In the **Scale** properties, the numerical scaling range and the division by main and subscale can be defined. The value range of the scale is limited by the value in **Scale start** and by the value in **Scale end**. The value of **Scale start** has to be lower than the value of **Scale end**. Both distances specified in **Main scale** and **Subscale** can be set to 0 to disable a display of the scale marks: If the value for rough scaling (main scale) is set to 0, no scaling marks are drawn, regardless of the value set for precision scaling. If precision scaling (Sub scale) is set to 0, only the scaling marks for rough scaling are displayed.

To change the scale color, click into the scale color field. A dialog in which the color can be selected is opened.

In the **Label** section, the scaling representation can be specified. The entry in the input field **Unit** is intended to specify the scaling unit and is displayed at the top left of the histogram. After a font (color) has been selected, the label format can be adjusted: The formatting of the numerical values has to be specified according to the syntax of the programming language C (use %d for integer values and %X.f for floating point numbers, where X has to be replaced by the number of decimal points).

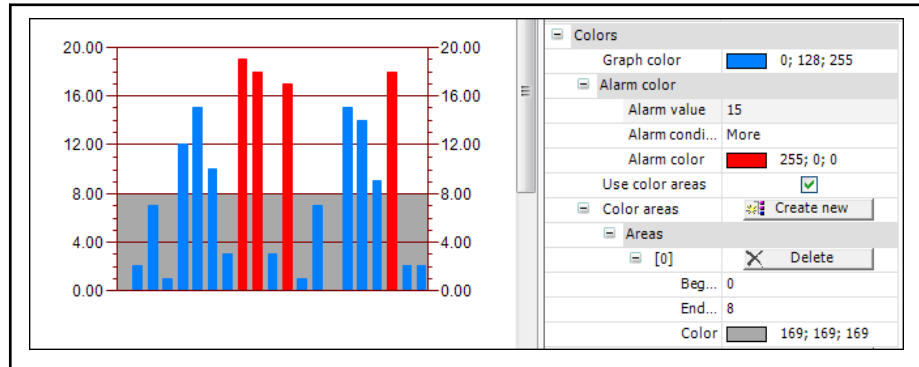


Fig. 20-69: Example - Configuration of colors of the histogram

Finally, the coloring of the element can be selected in the **Alarm color** section:

First, select the histogram color. In the **Alarm color** subsection, an alarm color can be defined in which the individual bar is to be displayed if the current value of the input variable meets the requirements of the alarm state. This condition originates from the alarm value definition which either must not be exceeded (when setting the **Alarm condition** to **More**) or which must not be fallen below (when setting the **Alarm condition** to **Less**).

Subsections of the scale can be assigned a certain color by creating a color scale with the **Create new** button. The color scales are numbered in ascending order. Each color scale has its own input fields within the element properties:

The limits of the subscale can be specified in **Scale start** and **Scale end**. The coloring can be selected from the drop-down list. A created color scale can be deleted with **Delete**.

20.4.5 'Lamps/switches/images'

"Image slider" element

The **Image switcher** element shows one bitmap from the three referenced bitmaps. If a mouse click action occurs at the visualization runtime, the displayed image is toggled. The results of mouse actions are set in the **Element behavior**.

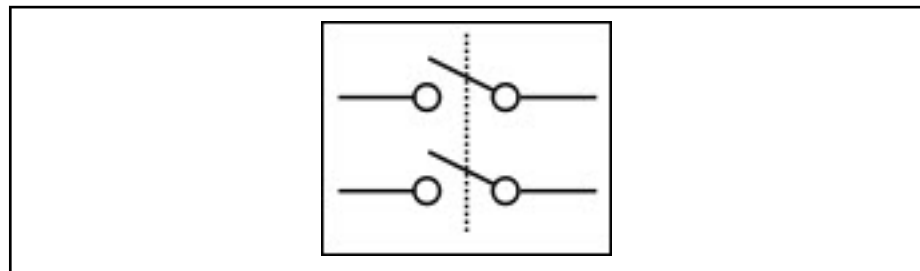


Fig. 20-70: Example of "Image on"

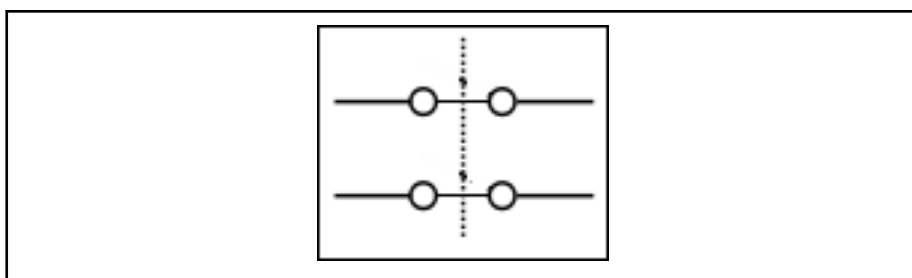


Fig. 20-71: Example of "Image off"

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Image switcher

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Boolean variable whose state changes according to the user input. Depending on the element behavior , the variable is TRUE as long as the mouse key is pressed (image tapper) or the value changes upon each mouse-click (image toggler).

Element property "Image settings" To maintain uniform entries for image names, you should add the name of the image pool as a prefix. This is not necessary if the images are managed in the global Image pool because it is searched first.

Image on	Bitmap ID from an image pool. The image status display is on. STRING type
Image off	Bitmap ID from an image pool. The image status display is off. STRING type
Image clicked	Bitmap ID from an image pool. The visualization displays the referenced bitmap during run-time as soon as the element receives the mouse action "Right mouse key pressed". Prerequisite: Element behavior is set to Image toggler .
Transparency	 Bitmap is displayed completely transparent in a partial area for all pixels in the Transparent color .
Transparency color	Color displayed completely transparent in the bitmap. Example: If the bitmap background is white and the transparent color is white, then the background is completely transparent. Prerequisite: Transparency is activated.
Scaling type	<i>Specify how the element should response when the frame is resized:</i> • Isotropic: Height and width of the screen are changed proportionally to the frame. Please note the following if you want to retain the element alignment within a scaled "Frame" element. For example, a lamp has been aligned in the center above a switch and the lamp is to remain in this horizontally centered position even if the frame size changes. To avoid undesired horizontal or vertical offsets, explicitly define the horizontal or vertical alignment with centered (see below). • Anisotropic: The image fills the entire frame regardless of its proportions. • Unscaled: The image size is defined and not changed if the frame is resized.

Orientation horizontal	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the horizontal alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> • Left • Centered • Right
Orientation vertical	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the vertical alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> • Top • Centered • Down
Element response	• Image toggler: The status of the element and therefore the image and variable toggle with each mouse-click. • Image tapper: Image on is displayed and the Variable value is <code>TRUE</code> while the mouse button is pressed. In this state, the image referenced in Image pushed is not used.
FALSE keys	When deactivated (default setting), Variable is set to <code>TRUE</code> by pressing the mouse button. When activated, Variable is set to <code>FALSE</code>. Requirement: Image switcher is selected in Element behavior.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is <code>TRUE</code> , the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is <code>TRUE</code>, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

"Lamp" element

The lamp flashes if the assigned variable is set. The lamp color can be selected in the **Background image**.

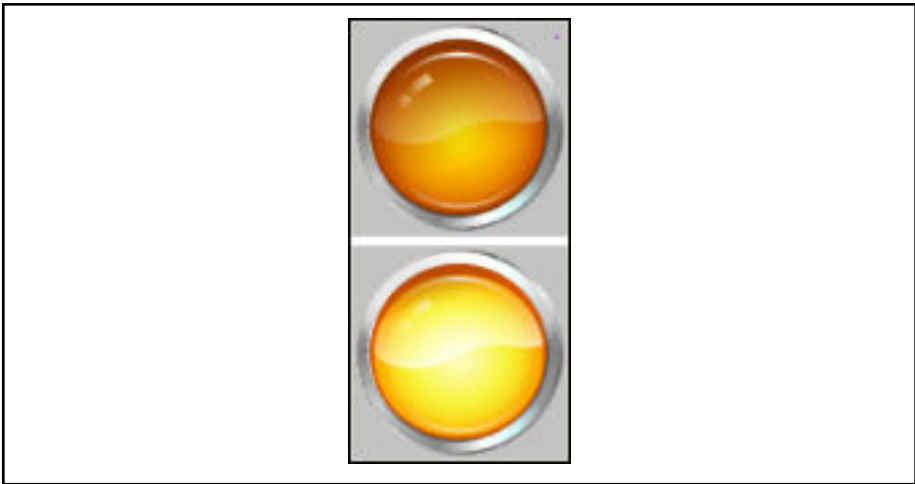


Fig. 20-72:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.
[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties
Element name

All element properties and their descriptions are listed below.

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Lamp

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Assign a Boolean variable whose state disables the state of the lamp. If TRUE, the lamp is switched on and flashes.

Element property "Image settings"

Scaling type	<i>Specify how the element should response when the frame is resized:</i> - Isotropic: Height and width of the screen are changed proportionally to the frame. Please note the following if you want to retain the element alignment within a scaled "Frame" element. For example, a lamp has been aligned in the center above a switch and the lamp is to remain in this horizontally centered position even if the frame size changes. To avoid undesired horizontal or vertical offsets, explicitly define the horizontal or vertical alignment with "centered" (see below). - Anisotropic: The image fills the entire frame regardless of its proportions. - Unscaled: The image size is defined and not changed if the frame is resized.
Orientation horizontal	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the horizontal alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> - Left - Centered - Right
Orientation vertical	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the vertical alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> - Top - Centered - Down

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ▶ Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property "Background"

Background image	<i>Select a color for the lamp:</i> • Yellow • Red • Green • Blue • Grey

Element “dip switch”, “power switch”, “push switch”, “push switch LED” and “rocker switch”

The switch is used to set the value of a Boolean variable

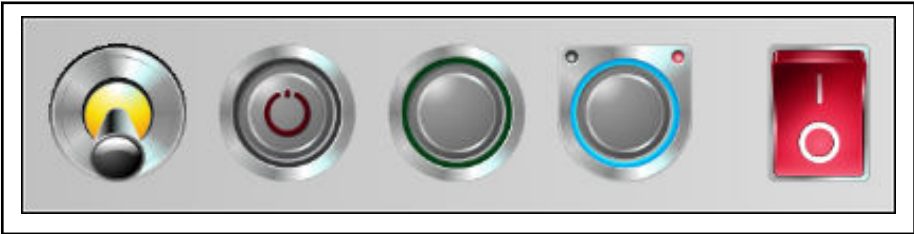


Fig. 20-73: Dip switch, power switch, push switch, push switch LED and rocker switch

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Type of the selected element: Dip switch, power switch, push switch, push switch LED or rocker switch.

Position

Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Boolean variable whose value changes according to the user input. Depending on the element behavior , the variable is TRUE as long as the mouse key is pressed image tapper or the value changes upon each mouse-click image toggler .

Element property "Image settings"

Scaling type	<i>Specify how the element should response when the frame is resized:</i> • Isotropic: Height and width of the screen are changed proportionally to the frame. Please note the following if you want to retain the element alignment within a scaled "Frame" element. For example, a lamp has been aligned in the center above a switch and the lamp is to remain in this horizontally centered position even if the frame size changes. To avoid undesired horizontal or vertical offsets, explicitly define the horizontal or vertical alignment with "centered" (see below). • Anisotropic: The image fills the entire frame regardless of its proportions. • Unscaled: The image size is defined and not changed if the frame is resized.
Orientation horizontal	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the horizontal alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> • Left • Centered • Right
Orientation vertical	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the vertical alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> • Top • Centered • Down

Element response	• Image toggler: The switch is shown as "on" and the Variable value is TRUE while the mouse button is pressed. • Image switcher: the display of switch and variable toggles with each mouse-click.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property "Background"

Background image	Select a color to label the switch: • Yellow • Red • Green • Blue • Grey

"Rotary switch" element

The rotary switch is used to set the value of a Boolean variable.



Fig. 20-74:

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Rotary switch

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Variable	Boolean variable whose value changes according to the user input. Depending on the element behavior , the variable is TRUE as long as the mouse key is pressed image tapper or the value changes upon each mouse-click image toggler .

Element property "Image settings"

Scaling type	<i>Specify how the element should response when the frame is resized:</i> - Isotropic: Height and width of the screen are changed proportionally to the frame. Please note the following if you want to retain the element alignment within a scaled "Frame" element. For example, a lamp has been aligned in the center above a switch and the lamp is to remain in this horizontally centered position even if the frame size changes. To avoid undesired horizontal or vertical offsets, explicitly define the horizontal or vertical alignment with "centered" (see below). - Anisotropic: The image fills the entire frame regardless of its proportions. - Unscaled: The image size is defined and not changed if the frame is resized.
Orientation horizontal	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the horizontal alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> - Left - Centered - Right
Orientation vertical	<i>This is available only if the check box option "Expert" is selected in the properties editor and the image scaling type is "Isotropic". Is used to explicitly retain the vertical alignment to another element (as defined in the basic visualization), if the visualization is used within a scaled frame. See above: "Isotropic" setting</i> - Top - Centered - Down
Element response	- Image toggler: The rotary switch is "on" and the Variable value is TRUE while the mouse button is pressed. - Image switcher: The state of the rotary switch and variable toggles with each mouse-click.
Alignment	- Top: Switch position: top. - Lateral: Switch position: lateral.
Color change	If the option is disabled, the button flashed if it is enabled. If the option is enabled, the button does not flash even if it is enabled.

Element property "Texts"

These properties are used for a static definition of the element labeling. Each property can contain a formatting sequence such as %s. In online mode, the sequence with the variable content defined in **Text variables** is replaced.

[chapter "Using text and language " on page 796](#)

Tooltip	Enter a text. It is used as tooltip of the element. The text is displayed in the visualization only in online mode. The text can contain a formatting sequence such as %s. The variable is assigned in Text variables ► Tooltip variable .

Element property 'State variables'

Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.
Disable inputs	Entering a Boolean variable; if the variable value is TRUE, none of the element inputs has an effect. The element is disabled in the visualization to show that no user input is possible. If the visualization uses the user management, the elements for user groups are disabled with the access right "only visible".

Element property "Background"

Background image	<i>Select a color to label the switch:</i> • Yellow • Red • Green • Blue • Grey

20.4.6 'Special control elements'

"Trace" element and their dialogs

"Trace" element

Page content:

- ["Element properties" on page 973](#)
- ["How to configure a minimum Trace visualization element" on page 975](#)
- ["Effects of "Show cursor"" on page 978](#)
- ["How to configure a visualization trace with trigger" on page 979](#)

The element facilitates the integration of a trace graph into the visualization. The trace graph permanently monitors and displays variable values. The trace graph is configured in the element properties. Additionally, control elements controlling the trace functionality have to either be manually programmed or have to be programmed using [chapter "Dialog "Trace wizard"" on page 988](#).

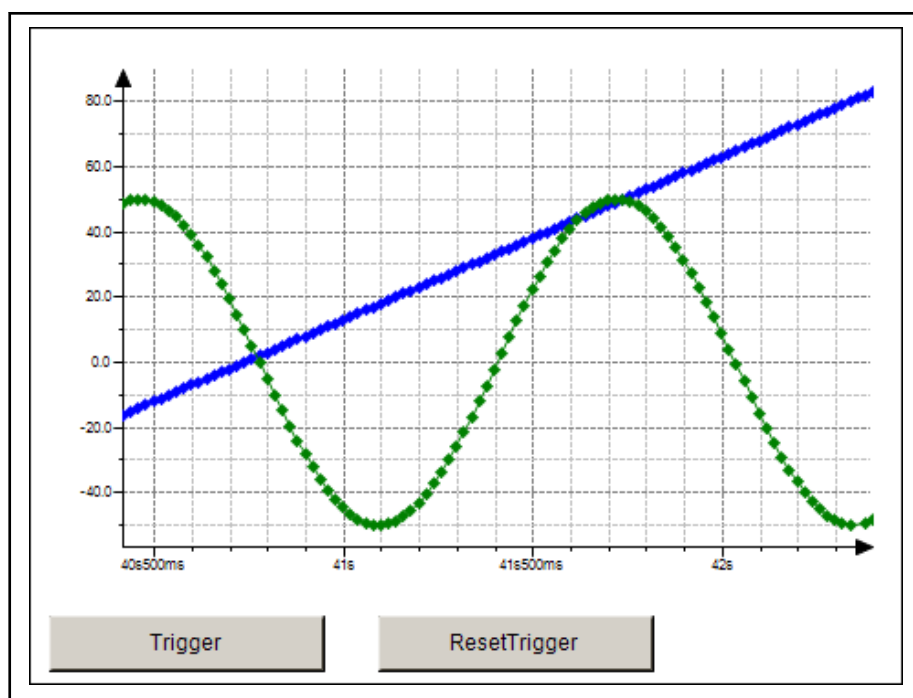


Fig. 20-75: Visualization trace online

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**
- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties

All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Application	Specify the application under which the trace variables are implemented. The application under which the visualization can be found is the default.

Element type	Trace

Trace	Click on the (unselected) field to open the chapter ""Trace configuration, recording settings" dialog" on page 981 dialog to configure the trace graph.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Show cursor	If this option is enabled, the trigger and variable values the cursor is pointed to are displayed as tooltip.

Overwrite existing trace	If a trace with the same name exists on the PLC, it is overwritten during a download with the configuration created here

Number format	Format definition for the variable value in the tooltip, for example %2.5f

Also refer to

- [chapter "Using text and language " on page 796](#)

Element property "Control variables"

Reset trigger	Enter a Boolean variable. If the value is TRUE, the trigger is reset.
Start trace	Enter a Boolean variable. If the value is TRUE, the trace is started.
Stop trace	Enter a Boolean variable. If the value is TRUE, the trace is stopped.

<i>Save trace to file</i> • Save trace • File name	If you want to save the current trace configuration in a file, enter a Boolean variable as value of Save trace and a string variable as value of File name . If the variable Save trace is TRUE, the configuration is saved under the specified name.
<i>Load trace from file</i> • Load trace • File name	If you want to load a trace configuration, enter a Boolean variable as value of Load trace and a string variable as value of File name . If the variable Load trace is TRUE, the configuration of the specified file is loaded.

How to configure a minimum Trace visualization element

Example: "MinimumTrace"

Step width

1. Declare a project variable for starting and stopping the trace in the IEC code. Implement a trace variable.
2. Select the settings in the **Trace** properties of the visualization element. A minimum trace contains a trace variable
3. Assign the project variables under **Control variables**, at least for **Stop trace** and **Start trace**.
4. Implement the code for trace control

In this example, a visualization element Trace is connected to the input buttons via project variables to control the trace and to display the value curve of the IEC variables "iVar".

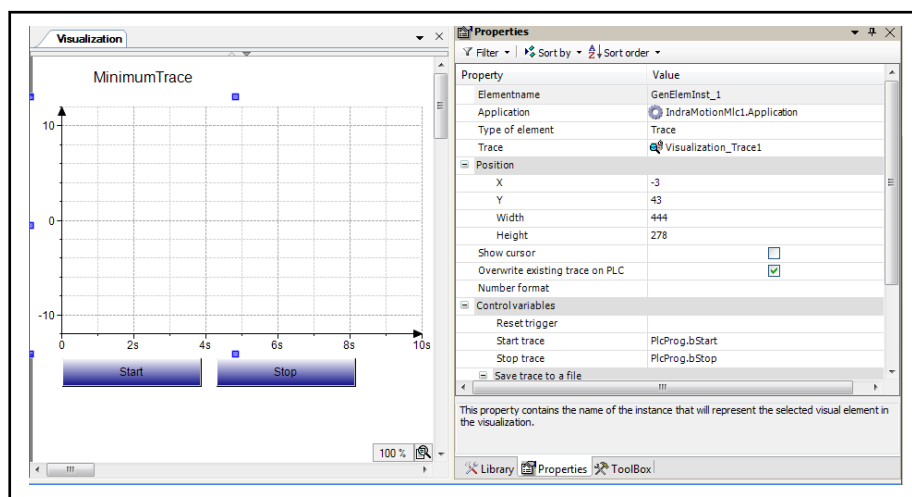


Fig. 20-76: Visualization "MinimumTrace"

Step 1: Save IEC code in program "PlcProg" under task "PlcTask":

```
PROGRAM PlcProg
VAR
  iVar: INT;
END_VAR
VAR_INPUT
  bStart: BOOL;
  bStop: BOOL;
END_VAR
iVar := iVar + 1;
```

Step 2: Add the "MinimumTrace" trace object to the application and configure the variable "iVar" as trace variable under task "PlcTask".

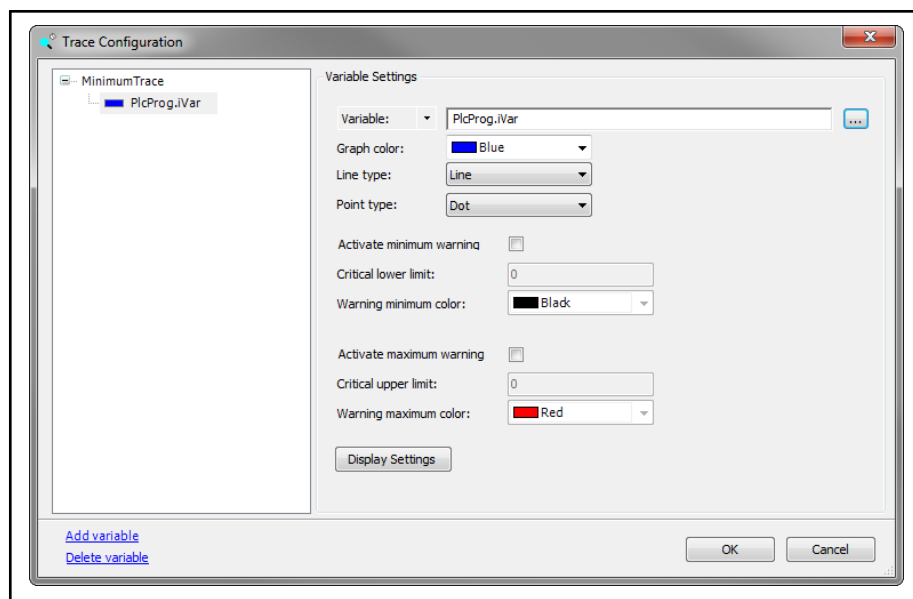


Fig. 20-77: Trace configuration "Minimum Trace", add variable

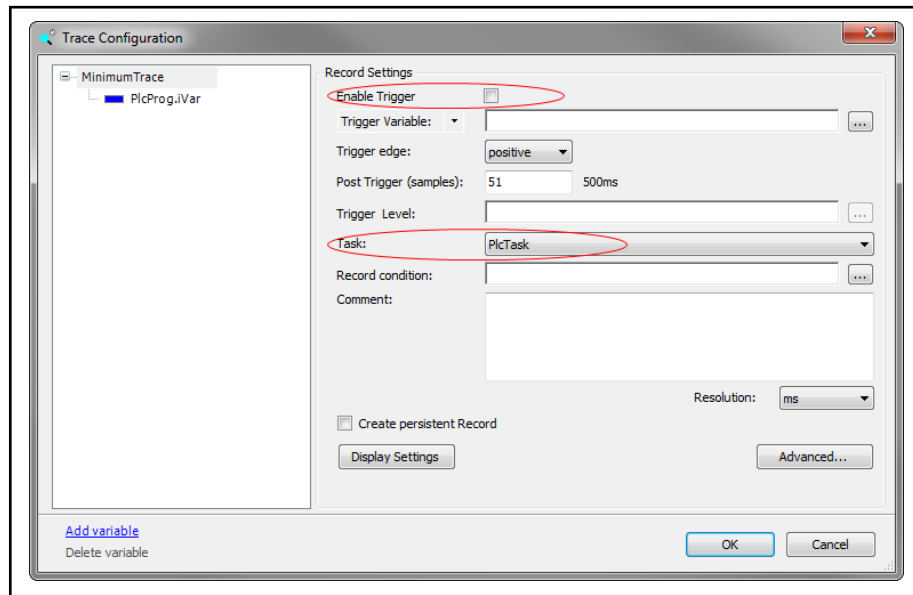


Fig. 20-78: Trace configuration "Minimum Trace", configuration

Step 3: Add the visualization object "Visualization" to the application.

The visualization consists of the text field "MinimumTrace", two buttons "Start" and "Stop" (general control elements) and the visualization element "Trace" (special control elements).

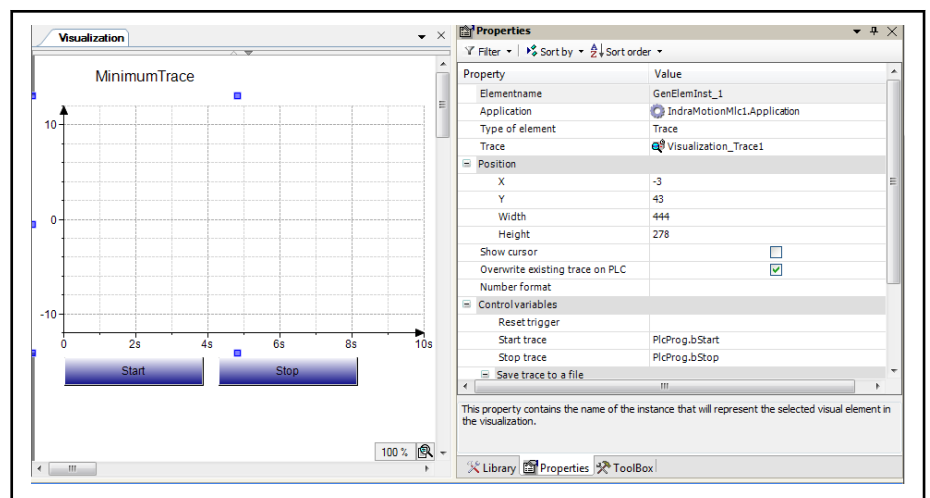


Fig. 20-79: Properties of the "Visualization_Trace1" trace object

Settings in the **Control variables** property

Settings in the Control variables property

Property	Value
Start trace	PlcProg.bStart
Stop trace	PlcProg.bStop

Step 4: Configure the "Start" and "Stop " buttons of the trace control

ST code for the "Stop" button programed in the **Properties** under **Input configuration** ► **OnMouseDown** ► **Execute ST code**

```
PlcProg.bStart := FALSE;
PlcProg.bStop := TRUE;
```

ST code for the "Start" button programed in the **Properties** under **Input configuration** ► **OnMouseDown** ► **Execute ST code**

```
PlcProg.bStart := TRUE;
PlcProg.bStop := FALSE;
```

Step 5: Connect Trace object and Visu object trace

Select visu object trace, context menu "Configure trace", establish connection.

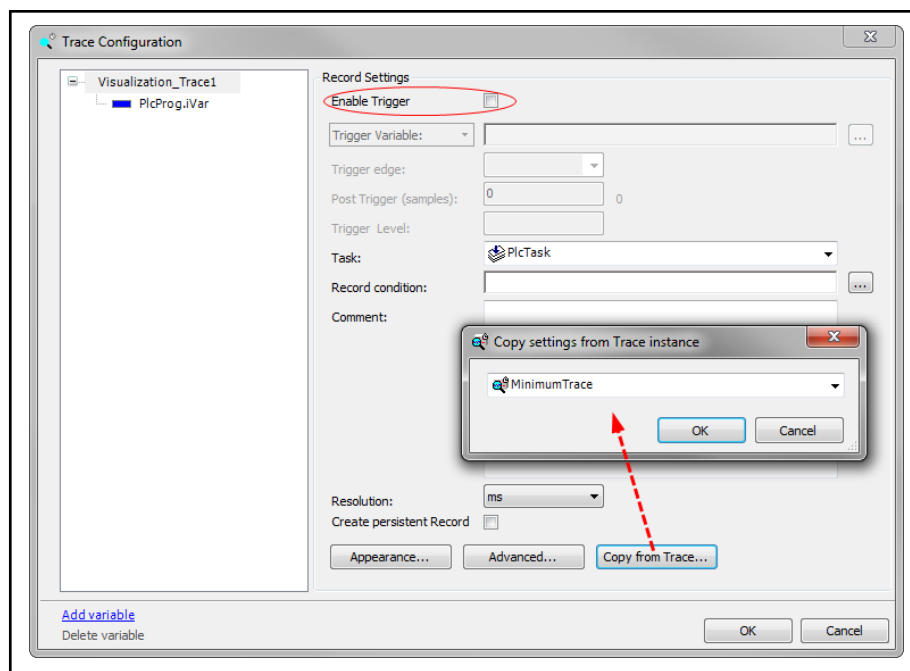


Fig. 20-80: Establish connection to the trace instance "MinimumTrace"

Step 6: Log in application login and start; start and stop "MinimumTrace".

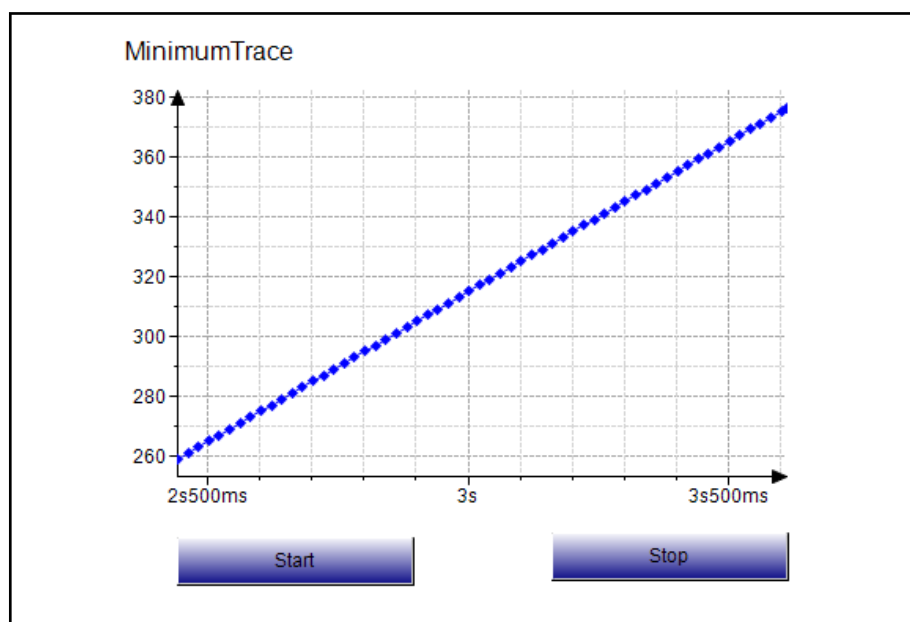


Fig. 20-81: "MinimumTrace" online

Effects of "Show cursor"

If the option **Show cursor** is enabled in the element properties of the trace visualization element, the values of the trigger and the trace variables can immediately be displayed in the tooltip at the position the cursor is at in the trace visualization element.

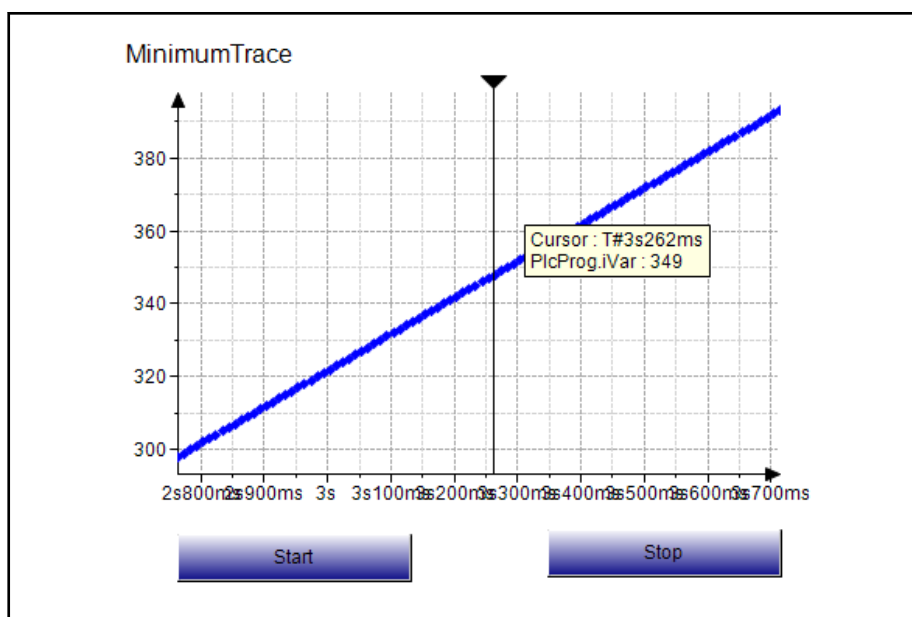


Fig. 20-82: Effect of the "Show cursor" property

How to configure a visualization trace with trigger

Configure the following properties

1. Declare the required project variables in IEC code and implement the variables to trigger and reset the trace.
2. **Trace:** Configure the trigger system in dialog [chapter "Trace configuration, recording settings" dialog](#) on page 981
3. **Control variables:** Assign a project variable for **Reset trigger**.
4. Implement a trace control for trigger and reset

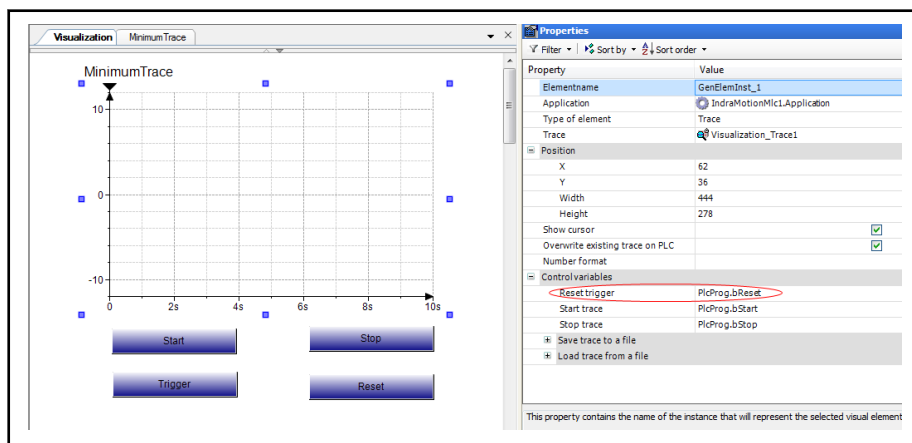


Fig. 20-83: Expand the "MinimumTrace" by "Reset" and "Trigger"

Step 1: Save IEC code in program "PlcProg" under task "PlcTask":

```
PROGRAM PlcProg
VAR
    iVar: INT;
END_VAR
VAR_INPUT
    bStart: BOOL;
    bStop: BOOL;
    bTrigger: BOOL;
    bReset: BOOL;
END_VAR
iVar := iVar + 1;
```

Step 2: Enable the trigger in the "MinimumTrace" trace object, configuration

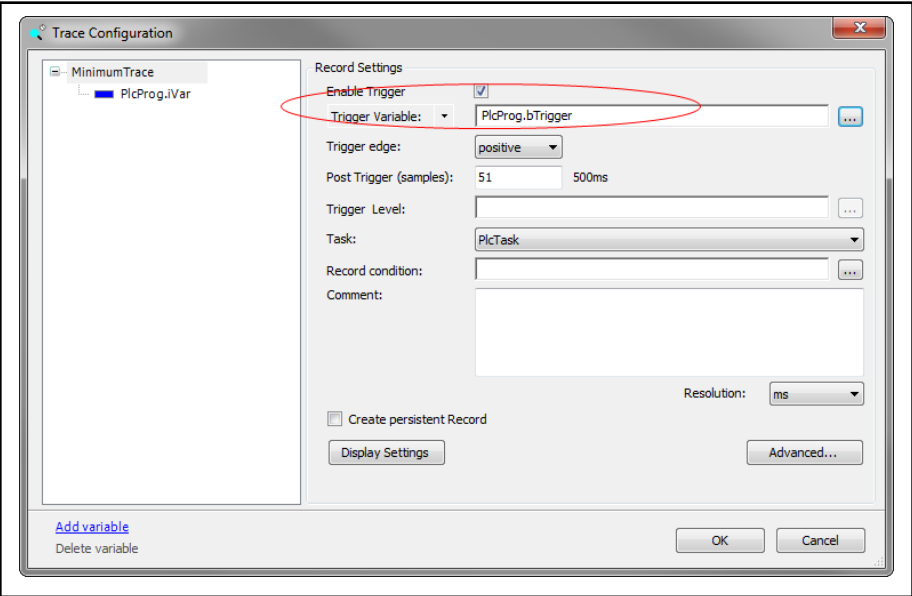


Fig. 20-84: Extension of trace configuration

Step 3: Setting in the Control variables property

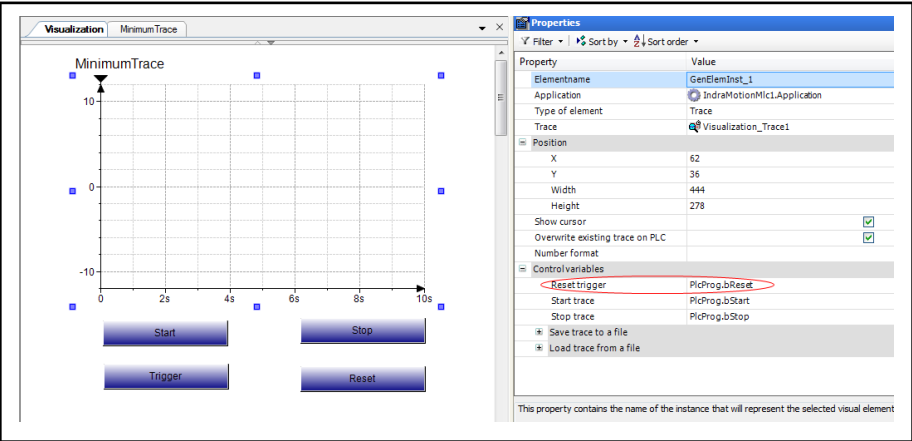


Fig. 20-85: Expand the "MinimumTrace" by "Reset"

 "Show cursor", please disable.

Reset	PlcProg.bReset

Step 4: Configure the "Start" and "Stop " buttons of the trace control

"Trigger" button: Triggering the trace by setting the corresponding trace variables to TRUE is enabled.

Button "Reset" resets the trigger.

ST code for the "Trigger" button, **Properties** under **Input configuration** ► **On-MouseDown** ► **Execute ST code**

```
PlcProg.bReset := FALSE;
PlcProg.bTrigger := TRUE;
```

ST code for the "Reset" button, **Properties** under **Input configuration** ► **On-MouseDown** ► **Execute ST code**

```
PlcProg.bTrigger:=FALSE;  
PlcProg.bReset := TRUE;
```

The following figure shows the trace visualization in online mode after the trigger variable has been activated by pressing the **Trigger** button:

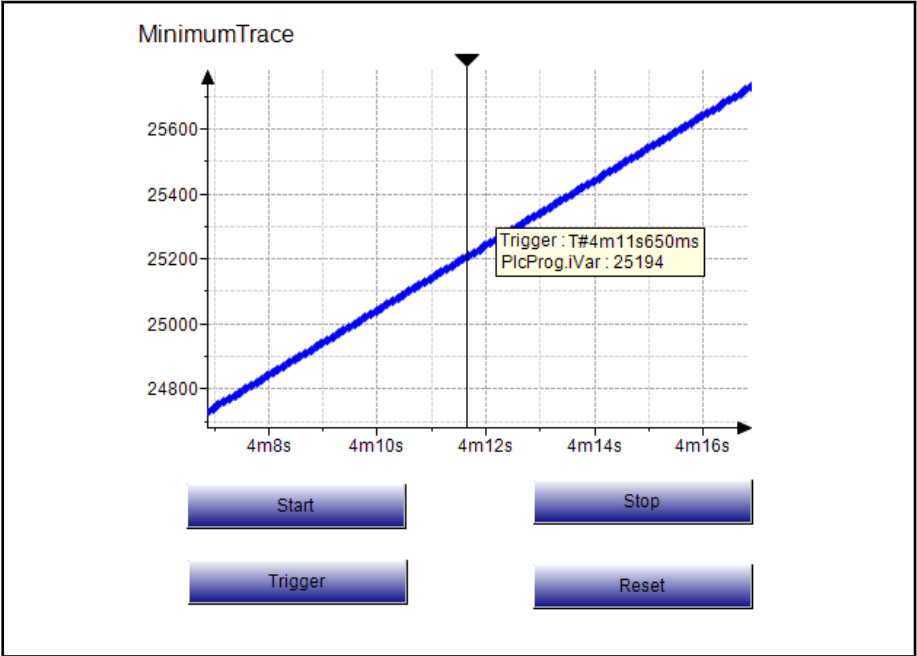


Fig. 20-86: Visualization in online mode

A black vertical line marks the edge of the trigger variables. Recording the variables is continued for the special trigger level for the entire trigger interval after the visualization was interrupted.

“Trace configuration, recording settings” dialog

The dialog **Trace configuration, recording settings** opens when clicking on the visualization trace in **View ▶ Element properties ▶ Trace**. The dialog is almost identical to "Trace configuration, recording settings" of the trace editor ([chapter "Trace configuration" dialog on page 737](#)).

Basics on the trigger

Recording of trace variables can be controlled as follows:

- by starting and stopping the recording
- by linking the start with a recording condition
- by starting the trigger which ends the recording.

Tree view of the trace configuration

The tree view shows the trace configuration and allows navigation to the settings. The top node contains the trace name. Click on this entry to open the **Record settings**. Under the node there are nodes for each trace variable whose data is acquired. Click on an entry to open the respective **Variable settings**.

Export trace configuration	The Export trace configuration dialog is displayed. Prerequisite: The top node is selected and the context menu is open.
Add variable	IndraLogic generates another trace variable that is integrated in the tree view and that is displayed with the corresponding Variable settings Prerequisite: A node is selected and the context menu is open.

Tab. 20-8: Available command

Also refer to

- [chapter "Command "Export trace configuration"" on page 988](#)

Determining and changing the record (trigger) settings

Prerequisite: Trace configuration name, i.e. the top node in the tree view is selected.

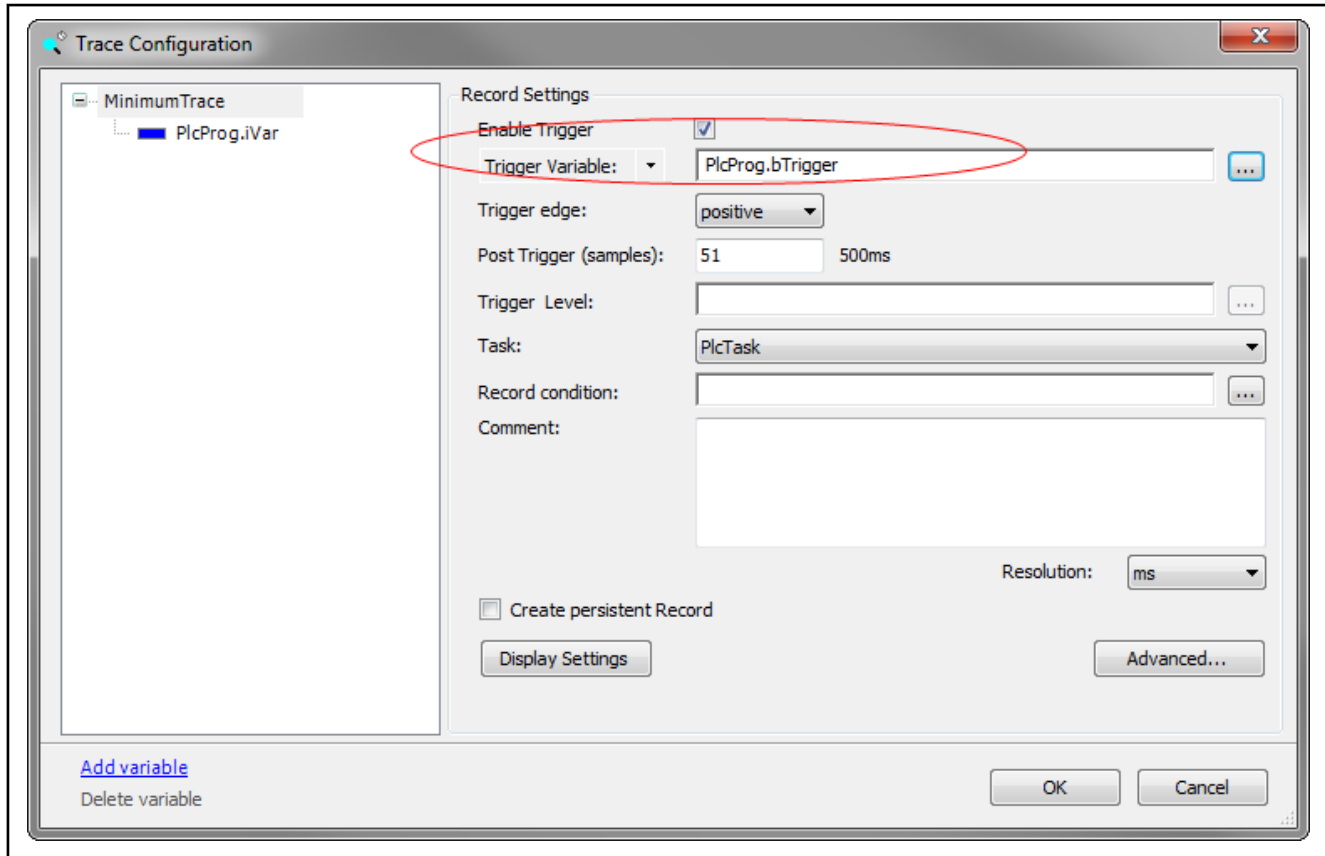


Fig. 20-87: "Trace configuration" dialog with "Record settings"

Enable trigger: Tick the checkbox to enable the trigger system according to the settings. If the trigger system is disabled, the system runs independently.

Trigger variable: Specify, which signal is to be used as trigger by entering the name (and path) of a signal.

A valid trigger signal is an IEC variable, a property, a reference, a pointer, an array element of the application, or an expression. Permitted types are all IEC-based types **except** STRING, WSTRING or ARRAY. Enumerations are permitted if the basic type is no STRING, WSTRING or ARRAY. The content of a pointer is not a valid signal. The input assistance [...] can be used to get valid entries.

To use a device parameter as trigger, click on  to switch to **Trigger parameter**. Open the input assistance using [...] and select **Traceable** parameters. The parameters available in the system are listed under **Elements**.

Trigger edge:

Positive	Trigger event at a rising edge of the Boolean trigger variable - or in case of an analog trigger variable - as soon as the value defined in the trigger level is reached "from bottom".
Negative	Trigger event at a falling edge of the Boolean trigger variable - or in case of an analog trigger variable - as soon as the value defined in the trigger level is reached "from bottom".
Both	The trigger event occurs at a falling as well as at a rising edge of the trigger variables or if the set trigger level is reached.

Post trigger (recordings): Number of recordings per input signal after the trigger has been started. Default value is 50, the value range is between 0 and $(2^{32}-1)$.

Trigger level: Enter a literal upon which the trigger is to start.

Task: From the list of available tasks, select the tasks in which data are to be recording.

Recording condition: Enter a variable of type BOOL or bit access. The content of a pointer is also an invalid entry. Properties are supported.

Comment: Enter a comment about the current recording conditions for example.

Resolution: Set the resolution of the trace time stamp to "ms" or "µs". For each signal to be detected, data pairs consisting of value and time stamp are saved and transferred to the programming system. The time stamps to be transferred are relative and relate to the tracing start. If the task cycle time specified is smaller than 1 ms, it is recommended to set the resolution of the time stamp to "1µs".

Create persistent record: If this option is selected, the trace configuration and the latest content of the trace buffer of the runtime system are saved persistently on the target system. This option is only available if a trace manager traces in the runtime system.

Appearance....: [chapter "Dialog 'Edit Appearance Settings'" on page 986](#)

Advanced....: [chapter "'Advanced trace settings' dialog" on page 985](#)

Copy from trace: [chapter "Dialog 'Copy settings from trace instance'" on page 988](#)

"Trace configuration, variable setting" dialog

Symbol: 

Function: This dialog box includes the variables configuration for a trace in the visualization.

Call: Double-click a trace variable in the tree view of the trace configuration

Also refer to

- [chapter 18.3.10 "'Trace configuration' dialog" on page 736](#), Trace object

All trace variables are displayed in the left section of the window in a tree structure. The top node contains the trace name.

Tree view of the trace configuration

The tree view shows the trace configuration and allows navigation. The top entry contains the trace name. Clicking this entry opens the associated configuration in the right of **Record Settings**. An entry is located here for each variable that data was recorded. Clicking this entry opens the associated configuration in the right of **Variable Settings**.

Command	Description
Add variable	A new, blank entry is added to the trace configuration. The configuration of a new entry is displayed on the right below Variable settings to make immediate settings.
Remove variable	The selected variable is removed

Tab "Variable settings" Includes the settings for variable that is selected in the tree view.

Variable	Valid variable; value recorded with full instance path <i>Valid</i> • IEC variable with valid type • Property • Reference Book • Content of the pointer • Array element based on a valid type • Enumeration based on a valid type <i>Valid data type: IEC-based type except</i> • STRING • WSTRING • ARRAY When the runtime system uses the <code>CmpTraceMgr</code> component, a property that is linked to the "monitoring" attribute can then be recorded as a variable.
Parameters	Parameters, whose data is recorded. Prerequisite: Runtime system with <code>CmpTraceMgr</code> component The Input Assistant dialog lists all valid system parameters in the Parameters category of the Categories tab.
	Toggle between Variable and Parameter
Associated Axis	Y-axis of the trace diagram for the Variable . The drop-down list provides the standard y-axis and the additional configured y-axes that are configured in the Edit Display Settings dialog box.
Show Variable Name	☒: The visualization shows the name of the IEC variable in the trace diagram at runtime. Either only the name is displayed or the name in parentheses after the Description. Default. ☐: The name of the IEC variable is not shown and does not appear in parentheses after the Description. Prerequisite: If any text is typed in Description, then you can disable the option.
Description	Text for the tooltip, e.g. <code>Sensor A</code>. When a visualization user focuses the cursor in the trend diagram, the visualization shows the tooltip. The text is typed into the Global-TextList object and can be localized there. When Show Variable Name is enabled, the text is supplemented with the variable name in parentheses. <code>Sensor A (PlcProg.iSensor_A)</code> If Description does not contain any text, then Display Variable Name is enabled. The name detached, without parentheses, for example <code>PLC_PRG.iSensor_A</code>
Color	Color of the variable in the trace diagram

Line type	<i>Display as line chart</i> • Line: Values are linked to form a line • Step: Values are linked in the form of steps • None: Values are not linked
Point type	<i>Display as scatter chart</i> • Point: Value as point • Cross: Value as cross • None: Value is not displayed
Warning at minimum	 : When below the lower limit, the visualization shows the trace graphs in the alert color.
Lower limit	Value (for example, 10)
Color	Warning color when fallen below
Enable upper limit	 : When above the upper limit, the visualization shows the trace graphs in the alert color.
Upper limit	Value, e.g. 90
Color	Warning color when exceeded
Dynamic Display Options	Control variable for programming the behavior of the visualization at runtime
Variable for visibility	IEC <code>BOOL</code> or <code>BIT</code> variable that controls the visibility of the variables in the trace diagram • <code>TRUE</code>: Visible • <code>FALSE</code>: Not visible

Also refer to

- [chapter "Dialog 'Edit Appearance Settings'" on page 986](#), Visu object trace

"Advanced trace settings" dialog

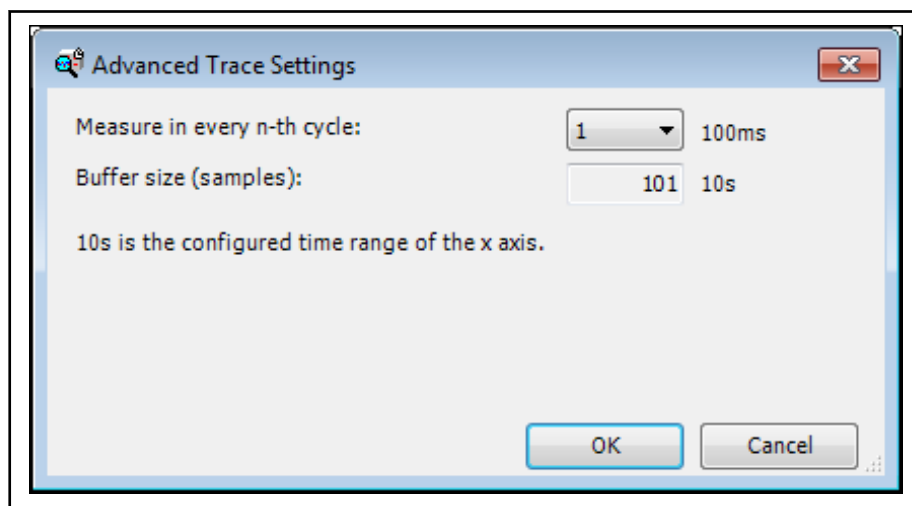


Fig. 20-88: Advanced trace settings

This dialog allows to configure the size of the trace buffer in the runtime of the trace visualization element. The number of measuring points is calculated from the following values:

- Cycle time of the task in which the trace is running
- **Measurement in every nth cycle**
- Length of X-axis as configured in the recording settings: [chapter "Dialog 'Edit Appearance Settings'" on page 986](#)

Buffer size (samples): The result is displayed.

Dialog 'Edit Appearance Settings'

Function: The dialog includes the configuration for the display settings of the **Trace** visualization element, for the x-axis and the y-axis at visualization run-time and provides a preview in the trace diagram

Call: Click on the **Display** button

Prerequisite: The **Trace Configuration - Record Settings** dialog is open and in focus.

Also refer to

- [chapter ""Trace configuration, recording settings" dialog" on page 981](#)

"X-axis" tab

Display mode	• Auto : The visualization scales automatically. • Fixed length : The visualization displays a segment of constant Length. The trace diagram continues running when new data is available. • Fixed : The visualization displays a segment from Minimum to Maximum.
Minimum	Start value on the x-axis Prerequisite: Fixed is selected.
Maximum	End value on the x-axis Prerequisite: Fixed is selected.
Length	Fixed length with start value adapted automatically Prerequisite: Fixed length is selected
Grid	 : Trace diagram with grid lines in the x-direction in the selected color  : The list box of colors for the visualization style opens

Fixed spacing	 : Axis scales with tick marks per Distance and Subdivisions
Distance	Distance between tick marks, for example 20 ms
Divisions	Number of subdivisions between tick mark, for example 5

Tab. 20-9: **Scale lines**

Fonts	Font for the axis label  : The list box of fonts for the visualization style opens

"Y-axis" tab

• Auto	 : The visualization scales automatically.
• Fixed:	 : Fixed range from Minimum to Maximum
Minimum	Start value of the range Prerequisite: Fixed is selected.
Maximum	End value of the range Prerequisite: Fixed is selected.
Grid	 : Trace diagram with grid lines in the y-direction in the selected color  : The list box of colors for the visualization style opens
Description	 : Text to label the y-axis with (for example, DC/mA)

Tab. 20-10: *Display mode*

Fixed spacing	 : Axis scale with tick marks for Distance and Subdivisions
Distance	Distance between scale lines, for example, 20 ms
Divisions	Number of subdivisions between tick marks (for example, 5)

Tab. 20-11: *Scale lines*

Fonts	Font for the axis label  : The list box of fonts for the visualization style opens

From visualization style	Background color in the defaults that is defined in the visualization style
Draw background	Background color that is selected in the lower input field
No background	Trace diagram without background color
Background color of the trend diagram	 : The list box of colors for the visualization style opens Prerequisite: Draw background is selected

Tab. 20-12: *Background*

Reset	Resets the default dialog settings
Use as default	Saves the dialog settings as standard

Add y-axis	Extends the trace diagram by one y-axis Consequence: The Trace configuration editor contains an extended selection of y-axes in the Additional axis option of the Variable settings .
Delete y-axis	Deletes the y-axis of the visible tab

Also refer to

- [chapter "Trace configuration, variable setting" dialog" on page 983](#)

Dialog “Copy settings from trace instance”

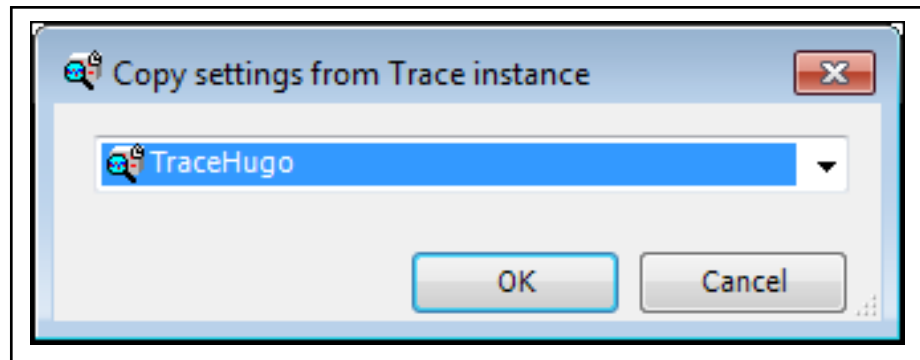


Fig. 20-89: Copy settings of trace instance

Select a trace object from the list to load its configuration to the **Trace configuration** dialog of the visualization element.

Dialog “Trace wizard”

This dialog opens when clicking on the **Insert elements for trace control...** command, contained in the context menu of the visualization editor or in the **Visualization** menu. It is only available if a trace element is selected.

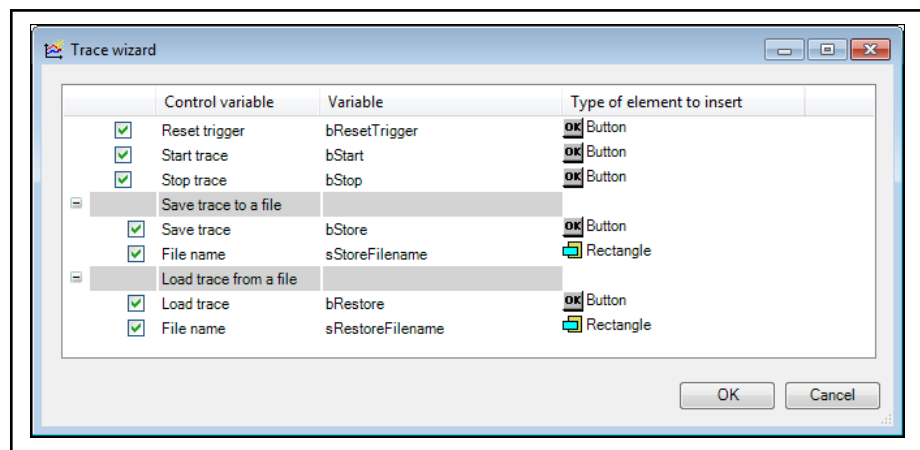


Fig. 20-90: Trace wizard

Control variable: In this column, the control variables of the trace element are listed. This list corresponds to the control variables available in the **Properties** view of the trace editor. Select the checkbox next to the control variable if a control element is to be generated for this variable in the visualization editor.

Variable: In this column, the variable names to be assigned to the trace variables, are listed. This list corresponds to the assignments defined in the **Properties** view of the trace editor. If nothing was assigned yet, a pre-assignment with standard variable names is provided. To change a name, open the line editor by double-clicking on the name or by clicking on the selected name field.

Type of element to insert: In this column, the types of the element to be inserted are listed. If the checkbox is selected, the element is inserted in the visualization. To insert another element type, double-click on the element to open the drop-down menu or click on the selected element and select the element. The drop-down menu contains all compatible elements.

Command “Export trace configuration”

Function: The command opens the **Export trace configuration** dialog.

Call: Context menu of the top node in the tree view of the trace configuration.

Prerequisite: The dialog **Trace Configuration** is active and the name of the trace configuration is selected in the tree view (example: `Visu_Trace1`).

Dialog "Export trace configuration"

The default dialog for file saving is used for exporting the trace configuration to a text file. The text file format complies with all requirements of the runtime component `CmpTraceMgr`. Thus, the component can read and use the text file.

File name	Name of text file to be created
File type	Trace file (*.trace): Format that the runtime system component <code>CmpTraceMgr</code> expects for reading

Also refer to

- [chapter "Trace configuration, recording settings" dialog" on page 981](#)

Element "ActiveX element"

The ActiveX element facilitates the implementation of an existing ActiveX component.



The ActiveX control can be used in the target visualization under Windows. On other target visualization platforms such as WinCE and Linux, it depends on the OEM which controls are supported.

Methods and parameters can be added or deleted using the  **Create new** and  **Delete** buttons.

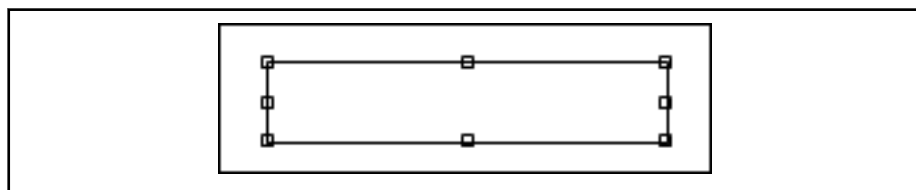


Fig. 20-91: ActiveX element

User input when editing an element

When editing an elements in the visualization editor, the following user inputs are provided:

- visualization commands for at least one element: [chapter 20.2 "Commands" on page 765](#)
- different mouse operations, applicable at the element: [chapter "User input when editing an element" on page 795](#)

User input when editing element properties

Properties define the characteristic features of a visualization element:

- Its appearance: Changes in properties defining the appearance of an element are visible immediately in the visualization.
- its order in the visualization.
- its animation: the visualization element can be animated by directly using IndraLogic project variables.
- Actions executed after a user input (event) of the element.

A property can be modified by editing the **Value** field. Depending on the property type, an input field, a drop-down list, a dialog or a checkbox to be activated is opened. Such a widget is open

- after a double-click
- after a single click in a **selected field**

- or via <space bar> if the field has already been selected.

if a variable is assigned:

- Enter its name.
- : Open the input assistance to select a variable. The category **Variables** lists all variables defined in the project.

[chapter 9.5 "Using input assistants" on page 107.](#)

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	ActiveX element

Element	Assign an installed ActiveX component.  is to be used when a component is to be selected using the input assistant. Enter a STRING parameter describing the name or ID of the components.

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Absolute motion" The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable. Absolute coordinate values are used here.

Motion • X • Y	• X: The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction (a positive value moves the element from the left to the right). • Y: The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in y-direction (a positive value moves the element from top to bottom).

State variables

Property	Value description
Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.

Element property 'Initial calls' Method calls, executed during the initialization can be defined under this property. They are exclusively executed in the first cycle.

Method calls Methods	Methods: Click on Create new to generate a new method call entry.

Element property 'Cyclic calls' Method calls, executed in each cycle can be defined under this property. They are executed in the refresh rate of the visualization.

Method calls Methods	Methods: Click on Create new to generate a new method call entry.

Element property 'Conditional calls' Additionally, conditioned calls can be assigned.
Conditioned method calls can be executed during the update interval of the visualization and can be defined here. In contrast to initial or cyclic calls, a call condition is defined under the **Methods ▶ [<Number>]** property. They are only executed at a rising edge of the call condition.

Method calls - Methods – [<Number>]	Methods: Click on Create new to generate a new method call entry.

"Initial/cyclic/conditioned calls" element property: Methods[<Number>]

[<Number>]	The number indexes a method call. Delete: Deletes the corresponding method call with all settings: Parameter, Event parameter and everything under Variable
Method	Enter the name of the method to be called.
Parameters - Parameters - Parameters : Variable	Assign the parameter variables. Create new: Upon each click, another Variable parameter entry is generated Variable: Assign a variable name for the parameter.
Result parameter Variable	Each method has a result parameter. Variable: Assign a variable that has to contain the result.
Call condition	Call condition is only available in Conditioned calls. Variable: Assign a Boolean variable. At a rising edge of the variable, the assigned method is called once.

"Busy symbol Flower" element

This animated element can be used to signalize that the system is busy or waits for data.



Fig. 20-92:

Element properties All element properties and their descriptions are listed below.

Element name

Property	Value description
Element name	The element name can be changed. Standard name is "GenElemInst_x", "x" indicates a continues numbering.

Element type	Busy symbol Flower

Position Define the position (X/Y-coordinate) and size (width and height) of the element in pixel. The origin is located at the upper left corner of the window. The positive x-axis runs to the right, the positive y-axis runs downwards. If the values are edited, the element displayed in the editor is also changed.

Property	Value description
X	Defines the horizontal position. X=0 is the left edge of the window. The positive axis points to the right.
Y	Vertical position. Y=0 is the top edge of the window. The positive axis points down.
Width	Horizontal element width.
Height	Vertical element height.

Element property "Colors" A color is defined using the hexadecimal number composed of red/green/blue (RGB) components. For each of these three colors, 256 values (0-255) are available. The settings are **static**. The color can be selected from a dropdown list or the color selection dialog opened via button . Example:  0; 128; 64

Frame color	Selecting a frame color
Fill color	Selecting a fill color

Appearance

Property	Value description
Line width	Weight of frame line in pixel. 0 codes the same as 1 and sets the line weight to 1 pixel. If no frame is required, set Line type to <i>invisible</i> .
Fill type	Select one of the fill types used in Fill color - Filled: The fill color is visible - Invisible: The fill color is invisible
Line style	<i>Select one of the line types used in the contour:</i> - Solid - Dashes - Points - Dash dot - Dash dot dot - Invisible: Contour invisible.

Symbol color	Selection of a color for the flower symbol
Line width	Specifies a number for the width of the symbol lines in pixels. 0 codes the same as 1 and sets the line weight to 1 pixel.

State variables

Property	Value description
Invisibility	Entering a Boolean variable; if the variable value is TRUE, the element is invisible in online mode.

20.4.7 Visualization Properties

Each element has element properties that can be used to configure the element. In **View ► Element properties**, the element properties are listed. The element property description can be found here or directly after the element description.

Property 'Diagram'

Chart	 : Click on the field and the Display settings dialog opens.

Property 'Element Name'

Element name	Default is GenElemInst_<x>, where x stands for a sequential number

Property 'Number format'

Number format	Format definition for the variable value in the tooltip, for example %2.5f

Also refer to

- [chapter "Using text and language " on page 796](#)

Property 'Alignment'

Alignment	Alignment of the component that was defined with the help of the trend wizard <i>Options</i> • Horizontal • Vertical

Property 'Position'

The position of the element is defined in X/Y-coordinates in pixels. The size of the element is specified in width and height. The origin is located at the upper left corner of the window. The positive X-axis points to the right, the positive Y-axis points downwards. The values correspond to the small blue boxes

in the visualization: .

X	Horizontal position. $x = 0$ is the left edge of the window. Positive axis points to the right.
Y	Vertical position. $y = 0$ is the top edge of the window. Positive axis points downwards.
Width	in pixels, e.g. 100
Height	in pixels, e.g. 50

Property 'Display frame'

Show frame	 : Activates the display of the diagram with frame

Property 'Display tooltip'

The property is available if the **Display cursor** property is activated.

Show tooltip	 : Activates the display of the tooltip. The tooltip contains the name and the value of the IEC variable at the point in time marked by the cursor in the trend diagram. The tooltip is available if the cursor is focused.

Property 'Text properties'

The property applies to text that is configured in **Texts ► Text** and **Texts ► Tooltip**.

Orientation horizontal	<i>Horizontal alignment of the text in the visualization element</i> • Left • Centered: Default setting • Right
Orientation vertical	<i>Vertical alignment of the text in the visualization element</i> • Top • Centered: Default setting • Down
Text format	<i>Text control in case the text is too long for the visualization element.</i> • Default: The text is truncated and only the part is displayed that fits in the visualization element. • Line break: The text is split into parts. • Ellipsis: The text is truncated with an ellipsis (. . .) to replace the missing part.
Fonts	<i>Text font. The list box is defined in the visualization style.</i> • Default • Heading • Large • Title • Note

Font color	Text color (for example, <code>Gray</code>). The list box is defined in the visualization style.
Transparency	Incremental value of the transparency that the visualization displays the text 0: fully transparent (invisible) 255: fully visible (default) Prerequisite: The visualization variant is configured for semitransparency.

Property 'Text properties'

Fonts	<i>Text font. The list box is defined in the visualization style.</i> • Default • Heading • Large • Title • Note
Font color	Text color (for example, <code>Gray</code>). The list box is defined in the visualization style.
Transparency	Incremental value of the transparency that the visualization displays the text 0: fully transparent (invisible) 255: fully visible (default) Prerequisite: The visualization variant is configured for semitransparency.

Property 'Texts'

Text	Text displayed to the left of, or above the button bar. If the element is horizontally aligned, then the text is left-aligned. If the element is vertically aligned, then the text is top-aligned. Default setting: <code>Zoom</code>

Property 'Labeling of the scale lines'

Two-line labeling	<i>Options</i> • : Date is displayed in the upper line and time in the lower line • : Display of date and time in one line
Omit unimportant labeling from the time stamp	• : Abridged display of the time stamp. The date is displayed with the first scale line; with the following scale lines only the time is displayed. • : Complete display of the time. The property Internationalization is available.
<i>Internationalization (format strings)</i> • Date • Time	<i>Entries</i> • Date: Date defines the date format. Default is the format set in Windows. "HH:mm:ss:ms" is the structure used in German-speaking countries. • Time: Defines the time format for the time of day. Default is the format that is set in Windows. The common order in German-speaking countries is "HH.mm.ss.ms".

Property 'Times'

Under **Times**, the interfaces are defined and configured in an array and the element time selection during runtime is provided. Thus, a user can select a certain time interval. The diagram that is controlled with the time selection is then switched to the configured time interval. The end time of the previous display is left unchanged and the start time is adapted.

Make "All" selection available	 : Time selector bar extended by All button. The diagram represents a time interval that covers all time stamps.
Times	 : On clicking this button a further button is inserted into the time selector bar and the array is extended by an entry. An additional index exists in the property Times ► Times ► Times ► [<new>]. Time is located under this index. The configuration of the button is to be entered there.
Times [Index] with index from {0, 1, 2,...}	Array of all buttons in the time selection bar. Index corresponds to the number of buttons.  : If Delete is clicked, the associated button is deleted from the time selector bar. The configuration entry is deleted from the Times property list.
[Index] Time	 : Time interval in standardized notation. For example: 3M for 3 months, 30m for 30 minutes. If a time interval is indicated in the field, then the button is labeled with it. If a user clicks on the button at runtime, the command is executed to switch the diagram to this time interval. The default is empty.

20.5 Visualization manager with clients

20.5.1 Visualization manager

When adding a visualization below an application in the Project Explorer, a visualization manager object  is automatically added.

The object manages the general settings of all visualizations assigned to the current application by clients.

Double-click on the Project Explorer entry to open the visualization manager object.

The editors opens in the "VisualizationManger <application path>" window, containing different tabs for **Settings**, **Standard keyboard configuration**, **Visualizations** and **Font settings**.

The applications are immediately applied upon changes.

"Visualization manager" - "Settings"

Symbol: 

Function: This tab includes settings that are valid across all applications for all visualizations.

Call: Visualization manager object type

Prerequisite: The application contains a Visualization manager

"General settings"

Use unicode string	 Visualization encodes character strings in Unicode
Use CurrentVisu variable	 The application recognizes and uses the global variable <code>VisuElems.CurrentVisu</code> of type <code>STRING</code> and maintains the name of the currently active visualization at runtime. The application has read access to the variable to get the name of the currently active visualization and write access to change the visualization. Prerequisite: The application contains a visualization calling other visualizations <i>Example</i> - Variable assignment: <code>VisuElems.CurrentVisu := strVisuName;</code> - Visualization name assignment: <code>VisuElems.CurrentVisu := 'visu1';</code>

Also refer to

- chapter "Using text and language " on page 796

"Style settings"

Selected style	- Style from the visualization style repository that every visualization of the application uses. Example: Flat style. <None>: Visualization represents its elements without any style or according to internal default setting. A standard dialog instead of a drop-down list facilitates value assignment in the element properties Color and Font.
	Drop down list with styles installed in the visualization styles repository is displayed.
Show all versions (for experts only)	 The list box includes only the latest version of the selected style and all other styles. If a newer version is installed for the selected style, it is also listed.  The drop-down list includes all installed styles with all versions.
Preview	The elements shown in the preview represent the style in Selected style .

Also refer to

- chapter 20.8.2 "Dialog "Options" - "Visualization styles"" on page 1016

"Language settings"

Selected language	Language used by the display variant when starting a visualization
'Settings for default text input'	For elements with default text input, a dialog appears at runtime to support the input. You can determine which dialog appears.
Numpad	Dialog called by the visualization when the user activates the input field for a number at runtime. The dialog includes a virtual numeric keypad. Preset: VisuDialogs.Numpad
Keypad	Dialog called by the visualization when the user activates the input field for a text at runtime. The dialog includes a virtual keyboard. Default setting: VisuDialogs.Keypad

'Settings for user management dialogs'

User management dialog	You can configure your visualization with a user management by configuring an input that displays the user management dialog. The <code>VisuUserManagement</code> library contains template visualizations. This is located in your installation directory, for example <code>C:\Program Files (x86)\3S CODESYS\CODESYS\Projects\Visu\Dialogs\VisuUserMgmtDialogs.library</code>. You can use other visualizations for the user management dialog by changing the default settings.
Dialog Login	User management dialog for login, prompting the user for a user name and password. The dialog opens as a follow-up action to the User management, Login action. Default setting: <code>VisuUserManagement.VUM_Login</code>
Dialog Change password	User management dialog for changing the password, prompting the user for the current and new passwords. The dialog opens as a follow-up action to the User management, Change user password action. Default setting: <code>VisuUserManagement.ChangePassword</code>
Dialog Change configuration	User management dialog for changing the configuration, displaying the current user configuration and the option to change it. The dialog opens as a follow-up action to the User management, Open user configuration action. Default setting: <code>VisuUserManagement.VUM_UserManagement</code>

"Additional settings"

Activate multitouch	 At runtime, the visualization expects user input by means of gesture recognition and touch events <i>Affected elements</i> • Elements with input configuration • Element type Frame • Element type Tab control element
Activate semi-transparent drawing	 The visualization draws the elements with semitransparency. IndraLogic provides the property Transparent for colors. Default setting: Activated. Prerequisite: You have created a new visualization and the display variants can draw with semitransparency

Also refer to

- [chapter "Using semitransparent colors" on page 815](#)

"Visualization manager" - "Default keyboard configuration"

This tab contains the keyboard configuration that applies to all visualizations of the application.

That means that the keys (hotkeys) defined can - if the respective device supports them - be used for user inputs in the visualization in online mode.

Operate the configurator as described in [chapter 20.3.4 "Tab 'Keyboard configuration'" on page 791](#).

In addition, several standard shortcuts can be used to navigate in a visualization, irrespective of the device: [chapter 20.11 "Keyboard operation in online mode" on page 1018](#)

"Visualization manager" - "Visualization"

The tab of the Visualization manager displays the currently available visualizations.

By default, the visualizations below the current application or directly referenced by these visualizations, are automatically loaded to the respective target system.

Referenced visualizations are either defined globally ("General module folder") or result from libraries.



This functionality replaces the previous use of "Visualization reference" objects. These objects can now not be inserted anymore. However, already inserted objects can still be used.

"Font settings" tab

The tab provides settings to adjust the font and size of a text of a language. The settings are applied to the texts of the visualizations that are located under the same application as the visualization manager.

Language	A column is created for each language that is used in the project. All text lists, including those from integrated libraries, are scanned for this.
Fonts	Font used by the visualization depending on the language.
Size factor	The factor affects the type size of all texts in the visualization. If the factor is smaller than 1, this leads to a reduction of the type size. If the factor is 1, all texts are displayed unchanged as defined in Properties . Default setting: 1
Context menu that can be activated when a cell is selected	• Delete language: The associated column is removed. This is advisable above all if settings in the column are highlighted in red. • Copy language settings: All settings in the column are copied to the clipboard • Paste language settings: All settings in the column are overwritten with the values from the clipboard
Red highlighting of a cell	The setting cannot be applied at runtime. The language is no longer present in the text lists of the project or the libraries

Tab. 20-13: *Language-specific font settings*

Also refer to

- [chapter "Using text and language " on page 796](#)

20.5.2 Clients and client editors

Diagnostic visualization

Page content:

- ["Restrictions in expressions and monitoring" on page 1000](#)
- ["Restrictions when using input events" on page 1000](#)
- ["Visualization interface" on page 1001](#)
- ["Data server" on page 1001](#)

Diagnostic visualization is a visualization client running on the programming system:

For diagnostic purposes, it is more user-friendly to run one or multiple visualization belonging to one application only within the programming system, without having to load a visualization code to the control.

The diagnostic visualization is available from V3.3.2.0 and is used automatically if no client object such as **Target visualization** or **Web visualization** is included below the visualization manager in the Project Explorer.

In case of a diagnostic visualization, **no** visualization code is generated or loaded to the control.

If you do not want to remove the **Target visualization** objects or the **Web visualization** objects below the visualization manager in your old project to obtain the diagnostic mode of the visualization, you can alternatively exclude the objects from compilation.

Use the setting **Exclude from compilation** in the respective "Properties" dialog of the object ([chapter 20.6 "Dialog "Properties" - "Visualization object" on page 1001](#)).



If a fixed compiler version was set in a project that was originally created with a version < V3.3.2.0, the project behavior corresponds to the options available in this version!

For versions smaller than V3.3.2.0, the setting **Do not use PLC for visualization** has to be selected so that a visualization only runs in the programming system.

Numerical values displayed in an element in a diagnostic visualization ("%s" in the "Text" property) are displayed in accordance with the currently set display format (binary, decimal, hexadecimal).



In contrast to the target visualization, VAR_INPUT variables behave like VAR_IN_OUTs variables in the diagnostic visualization.

Restrictions in expressions and monitoring

The diagnostic visualization only supports expressions that can be used by the monitoring mechanism of the programming system.

These are:

- Normal variable accesses such as `PlcProg.myPou.nCounter`
- Complex accesses as listed in the following; note that a runtime system of V3.3 SP2 or newer is required.
 - Access to an array of scalar data types in which one variable is used as index (`a[i]`)
 - Access to an array of complex data types (structures, function blocks, arrays) in which one variable is used as index (`a[i].x`)
 - Access to a multidimensional array by all types of data types with one or more variable indices (`a[i, 1, j].x`)
 - Access to an array with a constant index (`a[3]`)
 - Accesses as described above in which simple operators are used for the calculations within the index brackets (`a[i+3]`)
 - Nested combinations of the complex expressions listed above (`a[i + 4 * j].aInner[j * 3].x`)
- Operators supported in index calculations: +, -, *, /, MOD
- Pointer monitoring such as `asp^.x`
- Methods or function calls except are not supported, except for the following: all standard text functions, all type conversion functions such as INT_TO_DWORD, all operators such as SEL, MIN, ...

Restrictions when using input events

Within the input event **Execute ST code** only a list of assignments is supported.

Example

```
PlcProg.n := 20 * PlcProg.m;
// not allowed
IF PlcProg.n < MAX_COUNT THEN
```



```
PlcProg.n := PlcProg.n + 1;
END_IF
//use the following statement instead:
PlcProg.n := MIN(MAX_COUNT, PlcProg.n + 1);
```



If a list of assignments is used, the value on the left side is assigned in the next cycle. Immediate processing in the next line is not possible.

Visualization interface

Within the interface definition of a visualization, the type "interface"(INTERFACE) must not be used.

Data server

The diagnostic visualization is only intended to analyze the current application passively. Thus, the variable values provided from a different data source via the data server, are not refreshed automatically.

For these variables, only the initial values or the most recently transferred data are displayed.



If the values of these variables are edited in the own application, display is possible.

20.6 Dialog "Properties" - "Visualization object"

Symbol:

Function: This dialog contains the properties of a visualization object

Call: Menu **View** ► **Properties** of the selected visualization object or context menu of the object in the Project Explorer

The following properties are displayed and can be changed for the currently selected visualization object ("Properties" dialog, category "Visualization").

Use visualization as:

- **Visualization:** By default, the visualization is used as visualization object.
- **Dialog:** The visualization is a dialog and is thus automatically available in other visualizations as it is available in the **Input configuration** ► **Open dialog** of the visualization element.

Also refer to:

- [chapter "Input dialogs" on page 1002](#)
- [chapter "Programmed access to input dialogs" on page 1007](#)
- **Keypad/keyboard/input configuration dialog:** The visualization is a keypad or a keyboard and is automatically available in other visualizations as it is available in the input configuration, category **Write variable** ([chapter "Input configuration" on page 800](#)).

Note that the **interface** of a keypad/keyboard visualization has to be identical to the IndraLogic standard keypad/keyboard provided by the **VisuDialogs.library**.

Use the automatically determined visualization size: The size of the visualization which shows all currently contained visualization elements and the background image is determined.

If the **Include background image** option is, however, deactivated, the background image is cut off if it larger than the area including all visualization elements.

Use specified visualization size: The visualization size is defined by the displayed values (width, height) whereas it is not considered whether all visuali-

zation elements and the background image fit into this area, i.e. are completely visible.

Visualization size: Current **width** and **height** of the visualization object (number of pixels)

Internal: If the project is saved as library and if this option is activated, the visualization is only provided in the drop-down list of a frame or for a visu change within the library project.

If the library is embedded in a project, the visualization, however, is not available anymore.

The visualization is neither displayed in the visualization manager on the "Visualizations" tab. Such visualizations are only used as internal function blocks of the total visualization of a library function.

20.6.1 Use cases

Input dialogs

Page content:

- ["Example of generating and using user-defined dialog visualizations" on page 1002](#)

A visualization can be created as an input dialog for later use and is defined in its properties as **dialog** ([chapter 20.6 "Dialog "Properties" - "Visualization object" on page 1001](#)). Dialog visualizations can then be used in other visualizations to provide the user with an input mask. For example, dialog visualizations can be selected when configuring an **OpenDialog**, **CloseDialog** or **OnDialogClosed** of the **Input configuration** property of a visualization element.

VAR_INPUT and VAR_IN_OUT parameters of the dialog visualization have to be defined in the interface editor and have to be operated by another visualization element.

This other visualization element has to be configured to open the dialog ("Open dialog" configuration) ([chapter "Basic information on the interface editor" on page 782](#)).

This means that parameters can be used in a configuration of a different element.



For the diagnostic visualization (i.e the visualization running in the programming system), the VAR_INPUT parameters have to be of scalar type ([chapter "Diagnostic visualization" on page 999](#)).

Apart from user-defined dialog visualizations, **Standard dialog visualizations** can be used if `VisuDialogs` are integrated in the project. Currently, the standard dialog visualizations are visualizations of a login dialog and a dialog to open and save a file (`VisuDialogs.Login`, `VisuDialogs.FileOpenSave`).

Example of generating and using user-defined dialog visualizations

By clicking on an button, the dialog opens in which user data, i.e. user level and password, can be changed. Generate a visualization `ChangeUserLevel` for this dialog and a visualization `Visualization`, representing the button.

1. Generating the dialog visualization "ChangeUserLevel":

Generate a visualization object `ChangeUserLevel`, resembling the object in the following figure. In the object properties, define that you want to use the object as `Dialog`. In the current example, this is a dialog to enter the user

level and password. The dialog can only be closed with OK if the specified level and the password match.

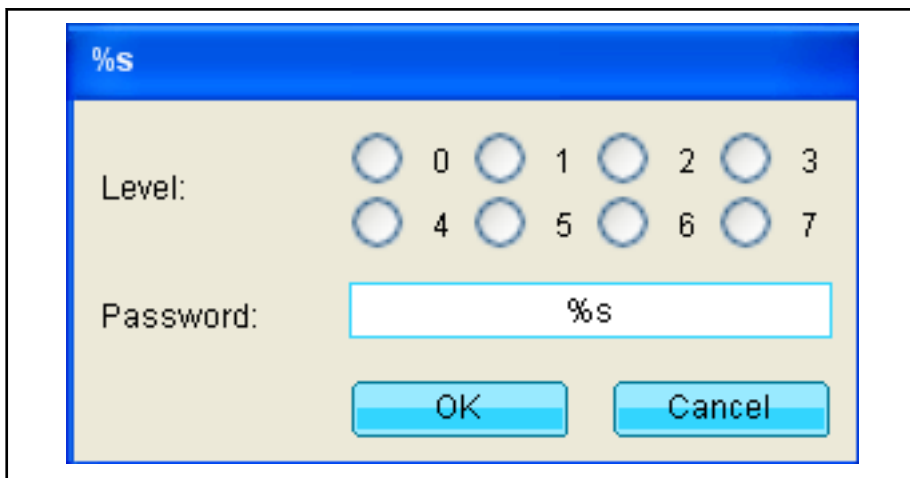


Fig. 20-93: *ChangeUserLevel* dialog visualization

The dialog consists of the following elements:

Element	Element properties
Image	Static ID: <code>Login</code> ; image of an empty dialog with a gray background and a blank, blue title bar. The image file can be provided via an image pool in which it is saved with the ID "Login".
Rectangle (only to display the dialog name in the title bar)	Texts: Text: <code>%s</code> ; is replaced by the value of parameter <code>stTitle</code> , see in the following; Text variables: Text variable: <code>stTitle</code> ; this variable is defined as input parameter of the visualization interface. Its value can be specified in the configuration dialog of the visualization element used to open the dialog (in the following example: "Change user level" button, see in the following).
Radio button (to enter the user level)	Number of columns: 4 Radio button arrangement: From left to right Radio button settings: Define the 8 areas [0] to [7] (respectively: Text: <code><n></code> , n is 0 - 7) Variable: <code>iLevel</code> (element input/output)
Text field (for password input)	Texts: Text: <code>%s</code> Text variables: Text variable: <code>stPwd</code> (element input/output) Input configuration: OnMouseDown: Write variable: Processing type: Edit , Use text output variable
Text ("Level")	Texts: Text: <code>Level:</code>
Text ("Password")	Texts: Text: <code>Password:</code>

Element	Element properties
Button (OK; color indicates if level and password match; only if they match, the dialog closes when clicking on OK)	Texts: Text: OK Colors: Color and alarm color must not be identical! Color variables: Change color: <code>stPwd <> MUX(iLevel, stLevel0, stLevel1, stLevel2, stLevel3, stLevel4, stLevel5, stLevel6, stLevel7);</code> (if password and level do not match, the color changes to gray) State variables: Disable inputs: <code>stPwd <> MUX(iLevel, stLevel0, stLevel1, stLevel2, stLevel3, stLevel4, stLevel5, stLevel6, stLevel7);</code> (if password and level do not match, the button is inactive) Input configuration: OnMouseDown: Close dialog: <code>ChangeUserLevel</code>, result: OK; (note: the input on the dialog is not written to the variable if the OK button is pressed)
Button, Cancel	Colors: Color = OK button color Texts: Text: Cancel Input configuration: OnMouseDown: Close dialog: <code>ChangeUserLevel</code>, result: Cancel

Open the interface button of the visualization and declare the following variable (the input variable values can be defined at a later point in the "Open dialog" input configuration, see **Open dialog** input configuration).

```

VAR_INPUT
    stTitle: STRING;
    stLevel0: STRING;
    stLevel1: STRING;
    stLevel2: STRING;
    stLevel3: STRING;
    stLevel4: STRING;
    stLevel5: STRING;
    stLevel6: STRING;
    stLevel7: STRING;
END_VAR
VAR_IN_OUT
    iLevel: INT;
    stPwd: STRING;
END_VAR

```

Example:

2. Create a visualization "Visualization"

A visualization `Visualization` with a button to open the "ChangeUserLevel" dialog and a display of the currently set level is created:

The following elements are used in the example:

Element	Element properties
Button (to open the "Change user level" dialog)	Texts ► Text: Change user level Input configuration: OnMouseDown: Open dialog: See the following figure
Text field	Texts ► Text: %s (dialog header); Text variables ► Text variable: <code>PlcProg.iLevel</code> (level inputs are assigned to program variables)

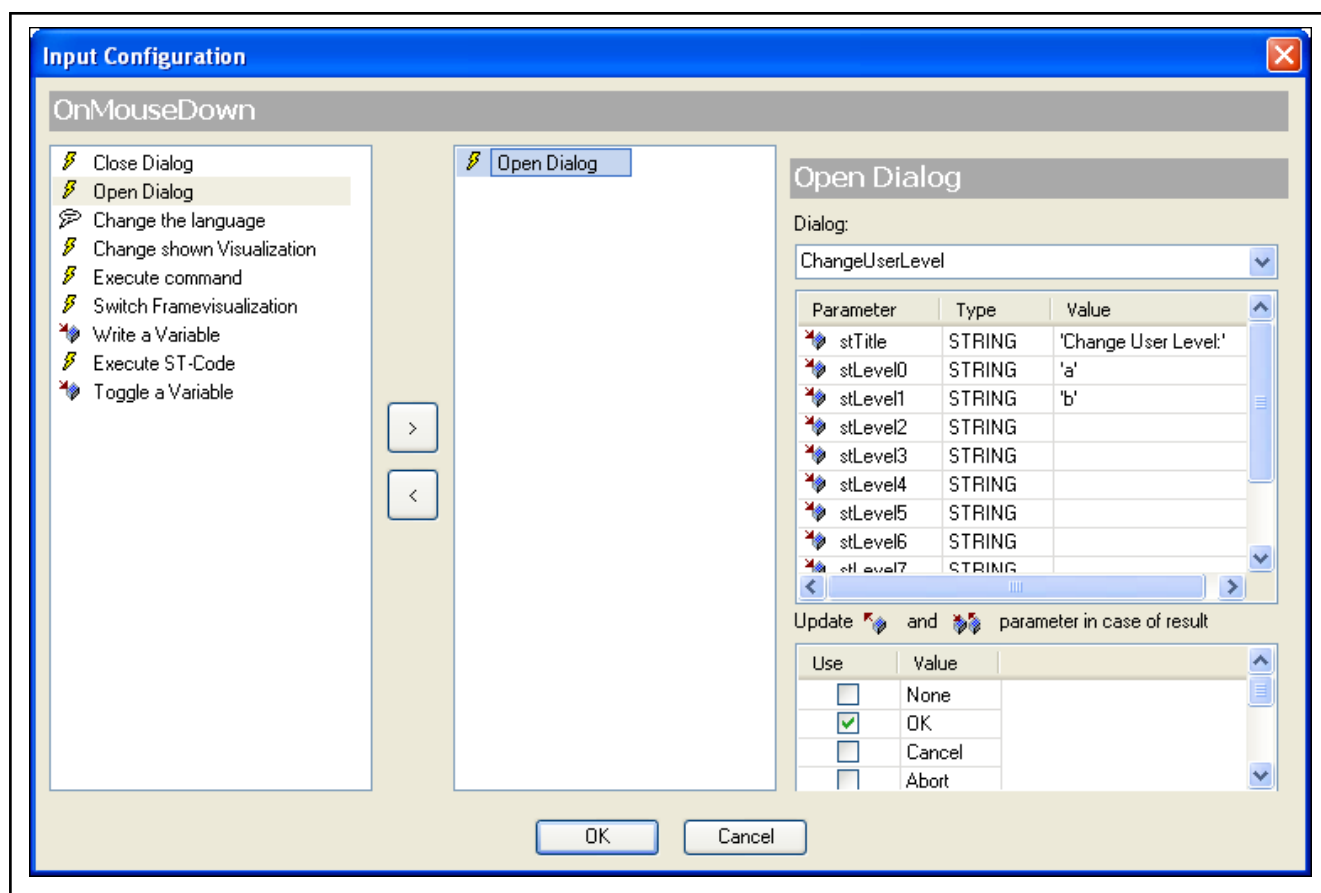


Fig. 20-94: Input configuration of the button that is used to open the user-defined dialog visualization *ChangeUserLevel*

A dialog has to be entered that can be opened with a click on the button. Select "ChangeUserLevel" from the drop-down list below **Dialog**. The input (VAR_INPUT) and input and output parameters (VAR_IN_OUT) of the dialog visualization ("ChangeUserLevel" interface) are listed. The user can select which values always have to be read by the ChangeUserLevel dialog. Please note that only 2 level-password combinations are configured for the example in the figure. If the user enters level 0, "a" has to be entered as password to activate the OK button. If level 1 is selected, "b" has to be entered as password. If one of the other levels is selected, no password has to be entered.

Also define upon which action in the ChangeUserLevel dialog the dialog is to be closed and when its outputs and inputs/outputs are to be written. In the example described here, select **use** of **OK**. Consequently, when using the OK button, the ChangeUserLevel dialog is closed and the `iLevel` and `stPwd` (VAR_IN_OUT) variables are written with the values of the user input.

For more information about the "Open dialog" configuration dialog, see ["Open dialog" on page 0](#).

Example:

3. Online mode of "Visualization"

When loading and starting the "Visualization" in online mode (either as target visualization, web visualization or "diagnostic visualization"), the ChangeUserLevel dialog can be opened by clicking on the "ChangeUserLevel" button in "Visualization":

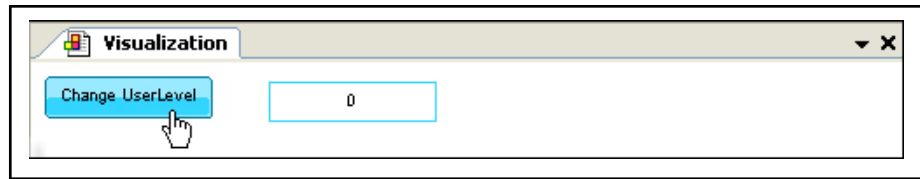


Fig. 20-95: Start "Visualization" and click on "ChangeUserLevel"

The dialog opens. Select a level, e.g. "1" (written to `iLevel`) and enter the matching password "b" (is written to `stPwd`). The values are compared by the OK button; the button is grayed out and inactive if the values do not match:

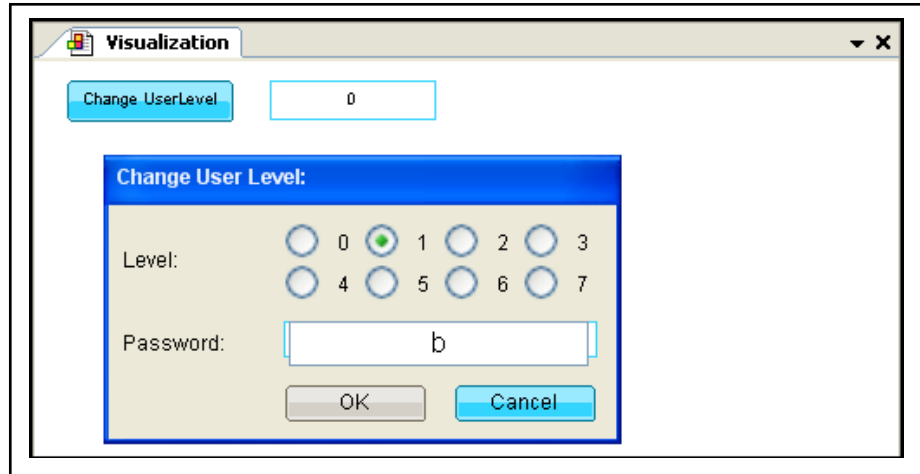


Fig. 20-96: Inputs in the "Change user level" dialog

Matching values have been entered, the OK button changes to blue and the dialog can be closed.

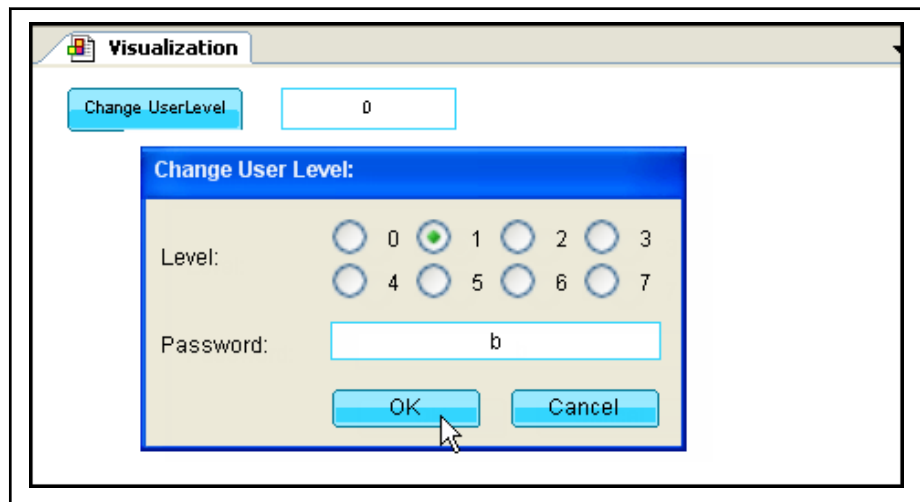


Fig. 20-97: OK button to close the dialog is available

The values are written (note that previously, the values were retained on the stack!) and the changed level ("1") is displayed.

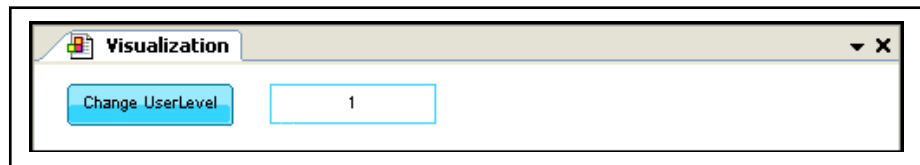


Fig. 20-98: The new value for the user level is displayed

Global dialog

A visualization can be configured so that it is displayed as global dialog on all active visualization devices, configured under the visualization manager. For example, a message window can be displayed on all connected visualization devices.

Inputs in one of the dialogs affect all dialogs. If one dialog is closed, all other dialogs are also closed. The dialog corresponds to a dialog visualization or an input dialog.

The `OpenDialog3` and `CloseDialog2` of the `VisuElems` library facilitates the IEC code implementation to generate and close a global dialog.

Implementing a global dialog

1. Implement a function accessing a visualization.

The visualization has to have the **Dialog** property and has to have the properties of an input dialog or a dialog visualization.

```
messageDialog := dialogMan.GetDialog('MessageDialog')
```

2. Set the `GlobalOpenCloseDialog` parameter of the dialog. The dialog is then globally displayed on all visualization devices.

```
dialogMan.OpenDialog3(messageDialog, pClientData,
TRUE, 0, 0, VisuElems.Visu_InputFlags.GlobalOpenCloseDialog);
```

3. Implement a function that calls the `CloseDialog2` dialog to close the dialogs on all visualization devices.

```
itfDialogManager2.CloseDialog2(messageDialog,
pClientData, VisuElems.Visu_InputFlags.GlobalOpenCloseDialog);
```

Also refer to

- [chapter "Input dialogs" on page 1002](#)

Programmed access to input dialogs

Page content:

- ["Dialog manager" on page 1007](#)
- ["Dialog manager methods" on page 1010](#)

Programmed access to an input dialog created in an IndraLogic visualization is possible as follows:

The parameters defined in the **interface** of a dialog visualization are automatically written to a structure `<DialogName>_VISU_STRUCT`.

(Dialog name = name of the visualization).

This structure can then be used by the application, e.g. using a function that is called if there is user input for a visualization element (configuration of the visualization element). Opening a dialog in a visualization and the reaction to user input in this dialog can be programmed.

Also refer to:

- [chapter "Input dialogs" on page 1002](#)
- [chapter "Basic information on the interface editor" on page 782](#)

Dialog manager

All visualizations that are configured as **Dialog** are automatically instanced and managed by the internal dialog manager. This manager can be ad-

addressed by the internal visualization manager by calling the `GetDialogManager()` method. The dialog manager provides some methods for the application to handle a visualization dialog.

The available methods are described in ["Dialog manager methods" on page 1010](#).

Using the login dialog from the "VisuDialog.library" library

Example of using the login dialog of the VisuDialog.library In this example, a button element in a visualization is configured so that if the "Login" button is clicked, a login dialog opens to enter a user name and a password. In this case, a standard login dialog is used that is provided by the VisuDialog.library. Dialogs generated by the user can also be handled analogously:

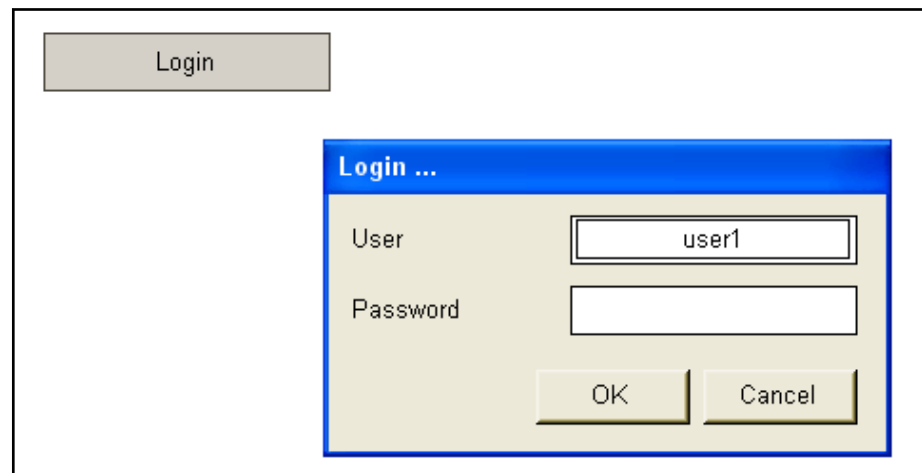


Fig. 20-99: Login dialog

Configuring the "Login" button:

A button element is added to a visualization. It is configured with the text property "Login" and the following two input properties:

1. OnMouseDown/Execute ST code: Function call 'OpenLoginDialog(pClientData)'. The variable 'pClientData' has to be transferred to a respective 'OpenLoginDialog()' function existing in the project.

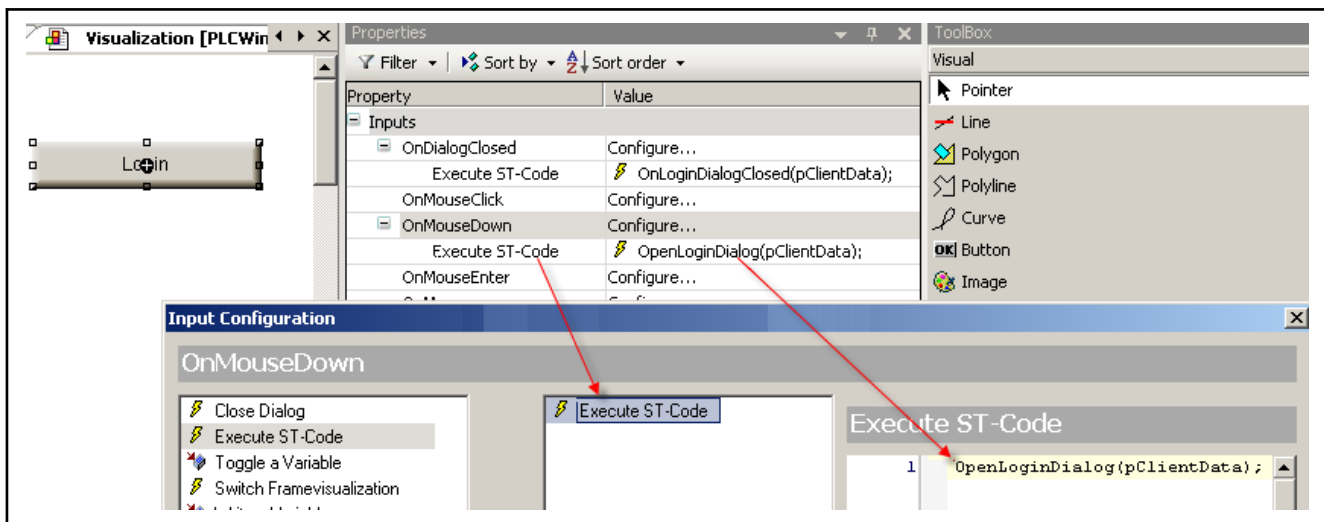


Fig. 20-100: Function call `OpenLoginDialog()`

2. OnDialogClosed / Close dialog / Execute ST code: Function call 'OnLoginDialogClosed(pClientData)'

The respective functions 'OpenLoginDialog()' and 'OnLoginDialogClosed()' are present as function blocks in the project. Refer to the following for a possible implementation:

Implementing the OpenLoginDialog() function

The function OpenLoginDialog() is started when the dialog opens, e.g. during an OnMouseDown action on the button element.

```

FUNCTION OnLoginDialogClosed : BOOL
VAR_INPUT
    pClientData : POINTER TO VisuStructClientData;
END_VAR

VAR
    dialogMan : IDialogManager;
    loginDialog : IVisualisationDialog;
    pLoginInfo : POINTER TO Login_VISU_STRUCT;
    result : Visu_DialogResult;
    stPassword: STRING;
    stUsername: STRING;
END_VAR

dialogMan := g_VisuManager.GetDialogManager();
// The DialogManager is provided via the implicitly available V
IF dialogMan <> 0 AND pVisuClient <> 0 THEN
    loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); // detects the l
    IF loginDialog <> 0 THEN
        result := loginDialog.GetResult(); // gets the result (OK, Cancel) of
        IF result = Visu_DialogResult.OK THEN
            loginDialog.SetResult(Visu_DialogResult.None); // reset to de
            pLoginInfo := dialogMan.GetClientInterface(loginDialog, pVisu
            // structure Login_VISU_STRUCT gets read;
            // in the following the structure parameters can be set
            IF pLoginInfo <> 0 THEN
                stPassword := pLoginInfo^.stPasswordpLoginInfo^.stPasswor
            // reset the password
                stUsername := pLoginInfo^.stUsername;
            END_IF
            ELSEIF result = Visu_DialogResult.Cancel THEN
                loginDialog.SetResult(Visu_DialogResult.None);
                // react on 'Cancel'
            ELSE
                // nothing to do here
            END_IF
        END_IF
    END_IF
END_IF

```

Implementing the OnLoginDialogClosed() function

OnLoginDialogClosed() defines the reaction to closing of the Login dialog.

```

FUNCTION OnLoginDialogClosed : BOOL
VAR_INPUT
    pClientData : POINTER TO VisuStructClientData;
END_VAR

VAR
    dialogMan : IDialogManager;
    loginDialog : IVisualisationDialog;
    pLoginInfo : POINTER TO Login_VISU_STRUCT;
    result : Visu_DialogResult;
    stPassword: STRING;
    stUsername: STRING;
END_VAR

dialogMan := g_VisuManager.GetDialogManager();
// The DialogManager is provided via the implicitly available V
IF dialogMan <> 0 AND pVisuClient <> 0 THEN
    loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); // detects the l
    IF loginDialog <> 0 THEN
        result := loginDialog.GetResult(); // gets the result (OK, Cancel) of
        IF result = Visu_DialogResult.OK THEN
            loginDialog.SetResult(Visu_DialogResult.None); // reset to de
            pLoginInfo := dialogMan.GetClientInterface(loginDialog, pVisu
            // structure Login_VISU_STRUCT gets read;

```

```
// in the following the structure parameters can be set
IF pLoginInfo <> 0 THEN
    stPassword := pLoginInfo^.stPasswordpLoginInfo^.stPassword := '';
// reset the password
    stUsername := pLoginInfo^.stUsername;
END IF
ELSIF result = Visu DialogResult.Cancel THEN
loginDialog.SetResult(Visu_DialogResult.None);
// react on 'Cancel'
ELSE
    // nothing to do here
END_IF
END_IF
END_IF
```

Dialog manager methods

GetDialog(String stName): Delivers the dialog currently affected by an event as an IVisualizationDialog:

METHOD **GetDialog**

Parameters:

Name	Type	Comment
GetDialog	VAR_OUTPUT	
stName	VAR_INPUT	

Fig. 20-101:

GetClientInterface(): Provides a pointer to the dialog structure

METHOD **GetClientInterface**

Parameters:

Name	Type	Comment
GetClientInterface	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

Fig. 20-102:

OpenDialog(): Opens the dialog for the client

METHOD **OpenDialog**

Parameters:

Name	Type	Comment
OpenDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	
bModal	VAR_INPUT	
pRect	VAR_INPUT	

Fig. 20-103:

CloseDialog(): Closes the dialog for the client

METHOD CloseDialog		
Parameters:		
Name	Type	Comment
CloseDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

Fig. 20-104:

Noticing events and input actions

Page content:

- ["Noticing closing of an edit control \(writing of a variable\)" on page 1011](#)
- ["Noticing keyboard events" on page 1013](#)

Using methods provided by the visualization libraries, the application can take notice of specific events triggered by user inputs in a visualization ([chapter 20.7 "Visualization libraries" on page 1015](#)).

Noticing closing of an edit control (writing of a variable)

The application can determine when an "Edit Control" is closed in an associated visualization, i.e. when a **variable value is written due to user input in an input field**. This can be useful if another action is to be executed after an Edit Control is closed.

To receive information about the closing of an Edit Control, a function block instance (implementing the interface `VisuElems.IEditBoxInputHandler` of the `VisuElemBase.library`) can be assigned to the global instance `VisuElems.Visu_Globals.g_VisuEventManager`; in this case, the method `SetEditBoxEventHandler` of this instance is used as shown in the following example.

`VisuElems.IEditBoxInputHandler` requires the method `VariableWritten`, which is called after a variable value is written due to user input in a visualization.

Example

There is a visualization with the following elements:

1. Two rectangles, each with an Edit Control field (input "Write variable") in which the user can input a value for variable i or j.

IndraLogic 2G visualization

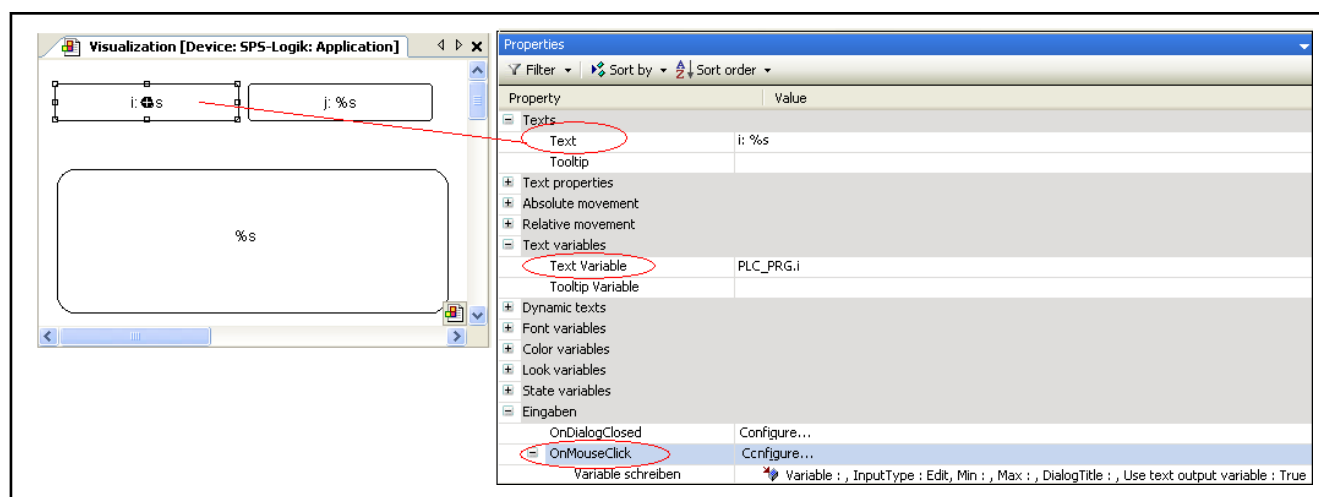


Fig. 20-105: Input elements

2. Another rectangle in which a text (text variable slnfo) with information is displayed if one of the input fields in either of the two upper rectangles is closed, i.e. if a value is written.

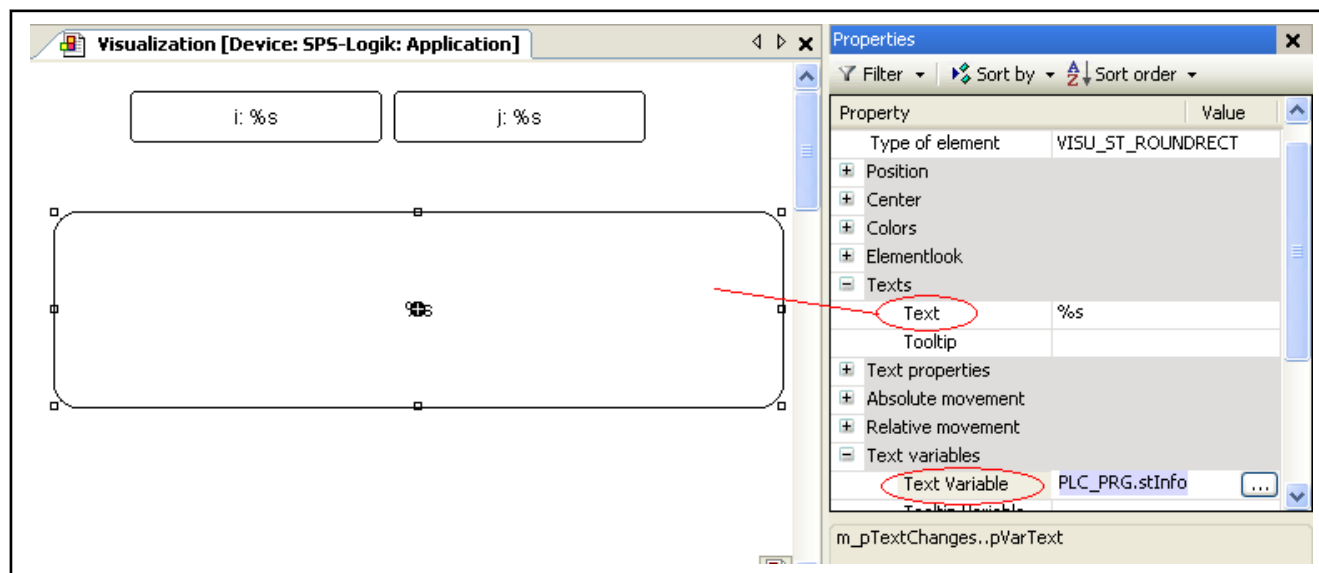


Fig. 20-106: Text output element

The application contains the following blocks:

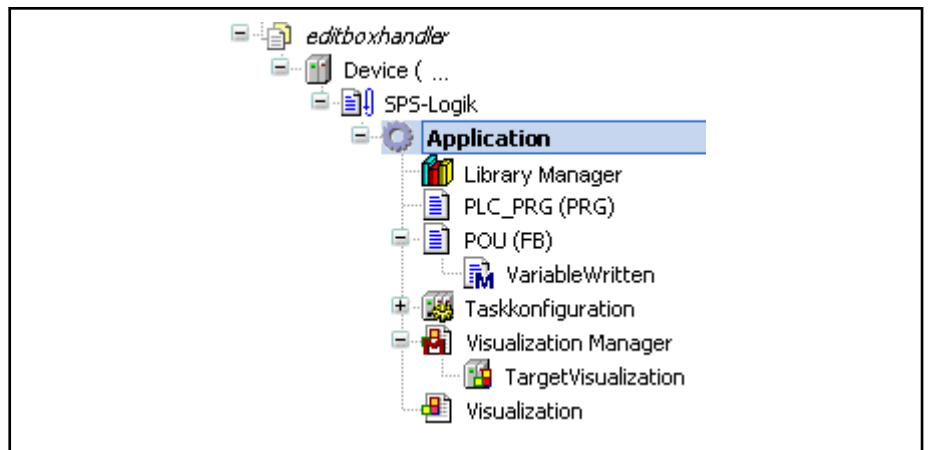


Fig. 20-107: Application with function blocks

Program PlcProg

```
PROGRAM PlcProg
VAR_INPUT
    i:INT; (* variable to be written by user input in visualization *)
    j:REAL; (* variable to be written by user input in visualization *)
    stInfo : STRING; (* information on the user input via the edit control field;
        string gets composed by method VariableWritten;
        result is displayed in the lower rectangle of the visualization *)
END_VAR
VAR
    inst : POU;
    bFirst : BOOL := TRUE;
END_VAR
IF bFirst THEN
    bFirst := FALSE;
    VisuElems.Visu_Globals.g_VisusEventManager.SetEditBoxEventHandler(inst);
    (* Call of method VariableWritten *)
END_IF
```

POU function block

FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IEditBoxInputHandler
(no further declarations, no implementation code)

Method "VariableWritten", part of POU:

```
METHOD VariableWritten : BOOL
(* provides some information always when an edit control field is closed in the visualization;
that is a variable gets written by user input in one of the upper rectangles *)
VAR_INPUT
    pVar : POINTER TO BYTE;
    varType : VisuElems.Visu_Types;
    iMaxSize : INT;
    pClient : POINTER TO VisuElems.VisuStructClientData;
END_VAR

// String stinfo, which will be displayed in the lower rectangle, is composed
PLC_PRG.stInfo := 'Variable written; type: ';
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, INT_TO_STRING(varType));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', adr: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, DWORD_TO_STRING(pVar));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', by: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, SEL(pClient^.globaldata.clienttype =
    VisuElems.Visu_ClientType.Targetvisualization,
    'other visu', 'targetvisu'));
```

Noticing keyboard events

The application can specify key actions (keyboard event) in a visualization, i.e. it notices when the user **presses and releases** a key during an active visualization.



Note the configuration possibilities for keyboard operation in visualizations ([chapter 20.11 "Keyboard operation in online mode" on page 1018](#)).

To receive information about such events, a function block (implementing the interface `VisuElems.IVisuEventManager` of the `VisuElemBase.library` implements) can be assigned to the global instance `VisuElems.Visu_Globals.g_VisuEventManager` and the method `SetKeyEventHandler` of this instance is used as shown in the following example:

Example

There is a visualization with a visualization element that displays the value `PlcProg.stInfo` as text.

That means that in online operation, the following information is output in this element regarding the activated key action: "KeyEvent up: <dwKey>, key: <key id>, modifier: <dwModifiers>, by: <pClient^.globaldata.clienttype.....>".

Example: "KeyEvent up: TRUE, key: 75, modifier: 0, by: targetvisu".

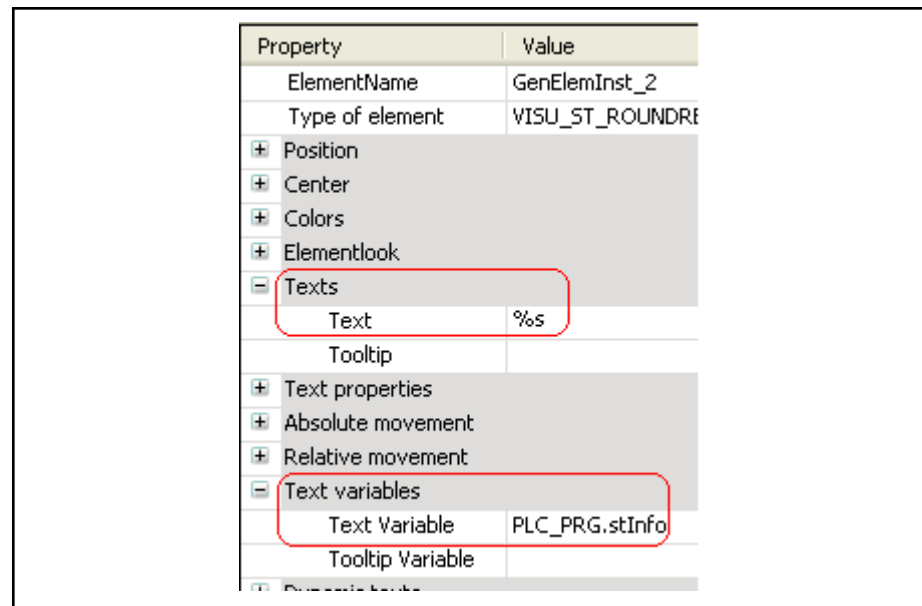


Fig. 20-108: Text configuration of the visualization element

The application contains the following blocks:

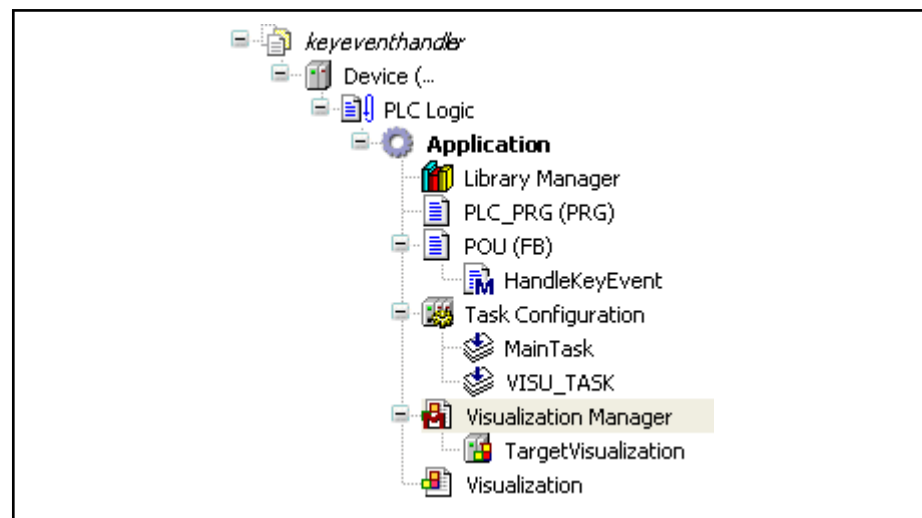


Fig. 20-109: Applications with POU

Program PlcProg

```

PROGRAM PlcProg
VAR_INPUT
    stInfo : STRING;
END_VAR
VAR
    inst : POU;
    bFirst : BOOL := TRUE;
END_VAR

IF bFirst THEN
    bFirst := FALSE;
    VisuElems.Visu_Globals.g_VisusEventManager.SetKeyEventHandler(inst);
END_IF

```

A message is displayed if a key event occurred.

POU function block

```

FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IKeyEventHandler

```

no further declarations, no implementation code

The interface `VisuElems.IVisuEventManager` requires a method with the following signature that is called with all of the available information:

Method HandleKeyEvent, part of POU

```

/// This method will be called after a key event is released.
/// RETURN:
/// TRUE - When the handler has handled this event
///        and it should not be handled by someone else
/// FALSE - When the event is not handled by this handler
METHOD HandleKeyEvent : BOOL
VAR_INPUT
    /// Event type. The value is true if a key-up event was released.
    bKeyUpEvent : BOOL;
    /// Key code
    dwKey : DWORD;
    /// Modifier. Possible values:
    /// VISU_KEYMOD_SHIFT : DWORD := 1;
    /// VISU_KEYMOD_ALT : DWORD := 2;
    /// VISU_KEYMOD_CTRL : DWORD := 4;
    dwModifiers : DWORD;
    /// Pointer to the client structure where the event was released
    pClient : POINTER TO VisuStructClientData;
END_VAR
VAR
END_VAR

PLC_PRG.stInfo := 'KeyEvent up: ';
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, BOOL_TO_STRING(bKeyUpEvent));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', key: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, DWORD_TO_STRING(dwKey));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', modifier: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, DWORD_TO_STRING(dwModifiers));
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, ', by: ');
PLC_PRG.stInfo := CONCAT(PLC_PRG.stInfo, SEL(pClient^.globaldata.clienttype =
    VisuElems.Visu_ClientType.Targetvisualization,

```

20.7 Visualization libraries

Visualization elements are implemented as function blocks in IndraLogic 2G. Analogously, they can be provided as libraries.

The IndraLogic 2G visualization profile also specifies exactly which elements from these libraries are available in the visualization editor toolbox **Tools**.

A visualization library is always created as **placeholder library**. The result is that the exact version of the library to be used is not specified as long as it is not integrated into a project. The current visualization profile specifies which version is actually needed.

Please note that this library type differs from the device-specific placeholder libraries in which the placeholders of the device description are resolved.

Basic libraries are integrated by default as soon as a visualization object is inserted in a standard project. They reference other libraries not listed here. In standard cases, the visualization libraries does not have to be inserted or use the libraries in your applications:

Basic libraries, namespaces

- **VisuElems**: Basic set of visualization elements
- **VisuElemMeter**: Pointer instruments and bar diagram
- **VisuElemWinControls**: Table, text field, scrollbars
- **VisuElemTrace**: Trace
- **VisuInputs**: Handling of user inputs of a visualization

In visualization assigned to a device **Device-dependent limitations** can apply that in turn depend on the libraries the device supports. The limitations refer to configuration options, such as a list of available elements, fonts, colors, image formats as well as maximum number of visualizations in the device.

20.8 Options

By default, the **Options** dialog is opened via the menu **Tools ► Options ► IndraLogic 2G**.

20.8.1 “Options” dialog - “Visualization”

Function: The dialog is used for configuring the visualization editor and at runtime of the configuration of the integrated visualization.

Call: Menu **Tools ► Options ► IndraLogic 2G**, category **Visualization**

Also refer to:

- [chapter “Options” dialog - Visualization” on page 734](#)

20.8.2 Dialog “Options” - “Visualization styles”

Function: The subdialog of the **Options** dialog is used to define the visualization style used in **visualizations in libraries and libraries** and **global visualizations** in projects.

Call: Menu **Tools ► Options ► IndraLogic 2G**, category **Visualization styles**



The current setting in the Options dialog is not applied during on-line mode, but the setting selected in the visualization manager. Refer to the general information about visualization styles: [chapter 20.10 “Visualization style” on page 1017](#).

Also refer to:

- [chapter “Options” dialog - “Visualization style” on page 735](#)

20.9 Visualization profile

Each visualization project, i.e. a project that contains at least one visualization object, has to be based on a visualization profile. This profile defines the following:

- The names and versions of the visualization libraries that are automatically included into the project as soon as a visualization object is created ([chapter 20.7 “Visualization libraries” on page 1015](#))

- A selection of the visualization elements of the integrated libraries available in the "Tools" view ([chapter 20.3.1 "Tools" view" on page 778](#)).

The **profile configuration** is completed outside of the project in the visualization element repository. Several profiles can be defined and stored on the local system.

The **profile used in the project by default** is defined in the visu style options. Switching to another profile is possible at any time. Note that the selected profile applies to all devices and applications.

In case the profile is changed, a message is displayed that indicates that the switch could prevent logging in without an online change or a download.

Switching profiles causes an automatic update in the library manager with respect to the required libraries. These do not have to be adjusted manually.

A visualization profile has to define at least one **"main" visualization library**, i.e. a library that defines the replacements for the placeholders used in other visualization libraries. In a standard IndraLogic installation, this library is the **VisuElems.** library.

When opening an **old project** that was not created with a visualization profile, you are prompted whether you want to assign the newest profile available.

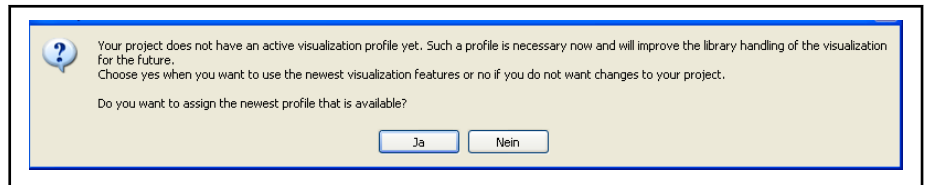


Fig. 20-110: Message box if there is no profile

If "Yes", the "Newest profile" is used ([chapter 20.8.2 "Dialog "Options" - "Visualization styles" on page 1016](#)).

If "No", the oldest available profile is entered in the project settings (it is called "Compatibility profile"), but no further changes are made in the project.

20.10 Visualization style

A style is a file in which symbolic names (style properties) are defined for values.

A style is therefore a collection of style properties that have either a fixed value for the property **Color** or **Font** or a fixed reference to an image file.

The file is in XML format with the file extension `*.visustyle.xml`.

Select one style per project and specify which style properties are available when configuring the elements.

IndraLogic also provides style properties as context-sensitive information. In this way, for example, only style properties for colors are offered for the property **Color**. A corporate identity is the result of consistent element design using style properties. When wanting to change the corporate identity, only the style has to be adjusted.

The style applied to visualizations included in an application, in the "General Module Folder" or in libraries is available in the option **Use the following visualization style** under menu **Tools ▶ Options ▶ IndraLogic 2G**, category **Visualization style**.

20.11 Keyboard operation in online mode

Some standard shortcuts are supported by every device for keyboard action in a visualization in online mode. In addition, depending on the device, other key (combinations) can be used.

Key(s)	Action
<Tab>	The next element is selected according to the chronological sequence of insertion. In this case, within a table, each individual cell is selected. If a frame element is selected, the selection is forwarded to the individual included elements
<Shift> + <Tab>	The previous element is selected; reverse order as with <Tab>
<Input>	An input action executed for the selected element.
<Arrow keys>	The next element in the direction indicated by the arrow key is selected.

Tab. 20-14: Standard key combination



Activate/deactivate the key action for diagnostic visualization via the "Activate key action" command. This can be desirable since as long as keyboard operation is activated for the visualization, other commands that are entered using shortcuts are not executed.

- [chapter 20.2.5 "Command "Activate keyboard usage"" on page 766](#)
- [chapter "Diagnostic visualization" on page 999](#)

Device-specific keyboard operation

Apart from the previously mentioned standard key combinations, each device has its additional combinations that can be used on this device during a running visualization: [chapter ""Visualization manager" - "Default keyboard configuration"" on page 998](#)



There is an option for handling keyboard events in the application code: [chapter "Noticing events and input actions" on page 1011](#).

21 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts offer you advice and assistance should you have any queries. You can contact us **24/7**.

Service Germany Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	+49 9352 40 5060
Fax:	+49 9352 18 4941
E-mail:	service.svc@boschrexroth.de
Internet:	http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

Index

Symbols

__AdrInst.....	427
__BitOffset.....	427
__Cast.....	427
__CATCH.....	331, 427
__COMPARE_AND_SWAP	
Example.....	335
Functioning.....	335
__Copy.....	427
__CRC.....	427
__CURRENTTASK	
Example.....	335
__DELETE.....	427
__ENDTRY'.....	331, 427
__FCall.....	427
__FINALLY.....	331, 427
__Init.....	427
__ISVALIDREF.....	327, 361, 427
__Lazy.....	427
__LocalOffset.....	427
__MaxOffset.....	427
__NEW.....	427
__PropertyInfo.....	427
__QUERYINTERFACE.....	329, 427
__QUERYPOINTER.....	330
__RefAdr.....	427
__RELOC.....	427
__System.ExceptionCode, data type.....	331
__SystemScope.....	427
__TRY.....	331, 427
__TypeOf.....	427
__uxint.....	359, 427
__VarInfo.....	427
__Wait.....	427
__VARINFO	
Data type SYSTEM.VAR_INFO.....	333
Example.....	332
Syntax in the declaration.....	332
Syntax upon call.....	332
__XADD	
Example.....	336
Functioning.....	336
__xword.....	359, 427
:=, ExST.....	229
"Place on top".....	771

0 ... 9

2-dimensional array	
Data access.....	364
Declaration.....	363
Initialization.....	364
2-dimensional ARRAY	
Example.....	363
3-dimensional array	
Data access.....	364

Declaration.....	364
Initialization.....	364
3-dimensional ARRAY	
Example.....	364
3-dimensional arrays of a customized structure	
Example.....	365

A

Abbreviations.....	5
About this documentation.....	1
Validity of the documentation.....	1
ABS.....	321
Access control	
Properties.....	706
Access right	
Device editor.....	508
Object.....	59
Accessor method.....	534
ACOS.....	325
Action.....	247
ACTION.....	427
Activate keyboard usage.....	766
ActiveX	
Visualization element.....	989
Add.....	605
Coil.....	664
Input connection.....	656
Output connection.....	656
ADD.....	298
Add POU	
Dialog.....	521
Adding	
EN/ENO.....	649
Adding and translating languages.....	114
Adding folder.....	605
Adding library	
Command.....	672
Adding objects.....	8
Adding text list support.....	688
Additional pointer.....	948
Address.....	348
I/O.....	79
Syntax.....	348
Addresses	
Memory position.....	349
Addressing	
Bits.....	347
Adjust size to grid.....	770
ADR.....	311, 359
Alarm color.....	831, 859, 869, 888, 916
Alarm state.....	822, 840, 844, 849, 909
Alias.....	489
ALIAS.....	361
Align bottom.....	767
Align left.....	767

Index

- Align right..... 767
 - Align top..... 767
 - Alt..... 829, 837, 846, 855, 864, 876, 895, 914, 925, 928
 - Alternative..... 643
 - Analyze code statically..... 127
 - AND..... 302
 - AND_THEN..... 303
 - AND, pragma..... 418
 - ANDN..... 259
 - Angle..... 814
 - ANSI Z535.6-2006..... 2
 - ANY..... 355
 - ANY <type>..... 355
 - Appearance..... 822, 832, 840, 849, 859, 869, 910, 992
 - Appearance variables..... 827, 844, 853, 863, 873
 - Application..... 480
 - Build..... 616
 - Clean..... 612
 - Compare..... 615
 - Compile..... 611, 616
 - Content..... 615
 - Download..... 214
 - Information..... 615
 - Name..... 70
 - Recompile..... 612
 - Remove from control..... 616
 - Reset..... 185
 - Applications, device editor..... 492
 - Applications, Device editor..... 491
 - Arcsine..... 324, 325
 - Arctangent..... 325
 - Array
 - Access..... 346
 - Declare..... 86
 - Set monitoring area..... 224
 - ARRAY..... 363
 - Limit monitoring ranges..... 194
 - of a structure..... 363
 - Array of a function block
 - Example..... 365
 - Array of arrays
 - Example..... 366
 - Syntax for data access..... 366
 - Syntax for declaration..... 366
 - ASIN..... 324
 - Assignment
 - FBD/LD/IL..... 264
 - Assignment operator 'REF'..... 230
 - Assignment operator, ST..... 227
 - Assignment, ExST..... 229
 - Assignments
 - ST assignment operator for outputs..... 227
 - Association for SFC action..... 645, 646
 - AT declaration..... 125
 - ATAN..... 325
 - Attribute
 - 'enable_dynamic_creation'..... 412
 - For pragma..... 379
 - Attribute 'call_after_init'..... 380
 - Attribute 'call_after_online_change_slot'..... 381
 - Attribute 'call_before_global_exit_slot'..... 382
 - Attribute 'estimated-stack-usage'..... 385
 - Recursive method call..... 385
 - Attribute 'hide'..... 389
 - Attribute 'initialize_on_call'..... 391
 - Attribute 'io_function_block', 'io_function_block_mapping'..... 393
 - Attribute 'monitoring'..... 394
 - Attribute 'no_assign_warning'..... 398
 - Attribute 'no_assign'..... 398
 - Attribute 'no-exit'..... 400
 - Attribute 'obsolete'..... 405
 - Attribute 'parameterstringof'..... 790
 - Attribute 'ProcessValue'..... 412
 - Attribute 'to_string'..... 409
 - Attribute "call_after_global_init_slot"..... 380
 - Attribute "call_on_type_change"..... 382
 - Attribute "const_non_replaced"..... 383
 - Attribute "const_replaced"..... 383
 - Attribute "dataflow"..... 383
 - Attribute "displaymode"..... 384
 - Attribute "ExpandFully"..... 388
 - Attribute "global_init_slot"..... 388
 - Attribute "init_namespace"..... 391
 - Attribute "init_on_onlchange"..... 392
 - Attribute "instance-path"..... 392
 - Attribute "linkalways"..... 394
 - Attribute "no_check"..... 399
 - Attribute "no_copy"..... 399
 - Attribute "no_instance_in_retain"..... 400
 - Attribute "no_virtual_actions"..... 401
 - Attribute "no-exit"..... 400
 - Attribute "noinit"..... 400
 - Attribute "pack_mode"..... 405
 - Attribute "pingroup"..... 403
 - Attribute "qualified_only"..... 408
 - Attribute "reflection"..... 409
 - Attribute "subsequent"..... 410
 - Attribute "symbol"..... 410
 - Attribute "VAR_IN_OUT_AS_POINTER"..... 787
 - Attribute "warning disable"..... 412
 - Attribute "warning restore"..... 412
 - Attribute "hide_all_locals"..... 390
 - Attribute "namespace"..... 397
 - Attribute "strict"..... 410
 - Auto-declaration..... 107
 - Avoid check function..... 399
- B**
- B..... 348
 - Background..... 773, 952
 - Background image..... 952
 - Bar diagram

Freely configurable, visualization element.....	934	Call_after_online_change_slot, Pragma attribute.....	381
Visualization element.....	934	call_before_global_exit_slot, Pragma.....	382
Begin of area.....	942, 946, 951, 955, 958	call_on_type_change, pragma.....	382
Beginning of pointer.....	948	Calling	
Bézier curve		Function.....	525
Visualization element.....	838	Function block.....	523
Binary number.....	342	CASE.....	232
BIT.....	359, 373	CATCH.....	331
Bit access.....	373	Center.....	822, 831, 840, 848, 858, 869, 888
Syntax for calls.....	347	Center horizontally.....	767
Syntax for declarations.....	347	Center vertically.....	767
To a variable with a global constant variable.....	347	Center X.....	814
To variables.....	347	Center Y.....	814
With BIT data types.....	348	CFC	
BITADR.....	313	Add input.....	656
Bitmap.....	773	Add output.....	656
Bitmap, properties.....	707	Auto data flow mode.....	711
Bookmark		Cancel grouping.....	658
Set and use.....	131	Change execution order.....	653
Bookmarks		Configure execution order.....	709
Jump to bookmarks within a POU.....	131	Connection mark.....	657
BOOL.....	352	Edit page size.....	648
BOOL constants.....	339	Edit parameters.....	653
BOOL_TO.....	314	Edit worksheet.....	648
Boot application		Explicit execution order mode.....	711
Generate.....	177	Generate control point.....	657
Properties.....	705	Grouping.....	658
Branching.....	249	One ahead.....	651
Insert.....	644	One back.....	651
Insert on the right.....	644	Order by data flow.....	652
Breakpoint		Order by topology.....	652
Properties.....	701	Parameter values.....	654
Set.....	180	Reference.....	650
Specify condition.....	181	Remove connections.....	656
Using breakpoints.....	179	Remove control point.....	657
Browse cross-references in classic view.....	594	Reset.....	649
Build information.....	612, 613	Reset connections.....	656
Button		Reset connector.....	656
Visualization element.....	887	Route connections.....	656
BY.....	231	Select linked connections.....	655
BYTE.....	352	Set.....	649
Byte addressing.....	348	Show next collision.....	655
		To the beginning.....	650
		To the end.....	651
		Unlock connection.....	655
C		CFC editor.....	270
C-stub date.....	612	Breakpoint.....	273
C-stub file for external library.....	612	Configure editors.....	270
CAL.....	313	Debugging.....	273
Call		Monitoring.....	273
Function block, ST.....	235	Online mode.....	273
Program.....	522	Option.....	721
Call list.....	189	Page-oriented.....	272
Call stack.....	189	Toolbar.....	224
Call_after_init_slot, pragma attribute.....	380	CFC editor - page-oriented	
call_after_init, Pragma attribute.....	380	Edit.....	272
		Execution order.....	273

Index

CFC editor online mode			
Breakpoint positions.....	276		
Commands in online mode.....	277		
Monitor a Boolean variable.....	273		
Monitor a scalar variable.....	274		
Monitoring.....	273		
Step into a POU.....	277		
CFC editor, online mode			
Change constant input parameters of function block instances.....	275		
cfg file.....	70		
Check syntax.....	126		
Check visualization text IDs.....	687		
CheckBounds.....	363		
Checkbox			
Visualization element.....	932		
CheckRangeSigned.....	377		
CheckRangeUnsigned.....	377		
Circle segment			
Visualization element.....	847		
Clean all.....	613		
Cleaning gaps.....	678		
Close dialog			
visualization.....	800		
Closed line branching, LD.....	268		
CmpTraceMgr			
Runtime system component.....	201		
Code			
Indentation.....	583		
Code analysis.....	127		
Coding rules.....	127		
Coil.....	267		
Insert.....	664		
Negated.....	267		
Reset.....	664		
Reset coil.....	267		
Set.....	664		
Set coil.....	267		
color.....	942, 946, 951, 955, 958		
Color.....	773, 814, 831, 859, 869, 888, 916, 948, 953		
Color gradient selection.....	822, 840, 849, 889		
Color scales.....	942, 946, 951, 955		
Color, font.....	890, 897, 937, 941, 945, 950, 954, 957		
ColorFonts.....	823, 832, 841, 850, 859, 870, 878, 901, 911, 917		
Colors.....	822, 831, 840, 844, 849, 859, 869, 888, 909, 916		
Combo box array			
Visualization element.....	882		
Combo box integer			
Visualization element.....	878		
Command			
'Download trace'.....	690		
'Use attributed member as input'.....	659		
'Cancel forcing for all values.....	633		
'Force FB input' command.....	658		
'Set active application'.....	608		
Add all forces to monitoring list.....	629		
Breakpoints.....	625		
Call list.....	627		
Call tree.....	592		
Close core dump.....	636		
Convert to multi-channel representation.....	692		
Convert to single-channel representation....	693		
Core dump.....	634		
Create core dump.....	634		
Determine next statement.....	631		
Disable breakpoint.....	627		
Display mode.....	636		
Edit breakpoint.....	627		
Enable breakpoint.....	627		
Execute to cursor.....	630		
Execute to return.....	630		
Force values.....	631		
Generate boot application.....	617		
Limit results to current declaration.....	699		
Load.....	618		
Load core dump.....	635		
Load device logbook from core dump.....	635		
Log out current device user.....	617		
Login.....	614		
Logout.....	617		
Monitoring - Monitoring list <n>.....	628		
Monitoring - Show all forces.....	628		
Multiple download.....	621		
New breakpoint.....	626		
Online change.....	619		
Operating mode.....	634		
PLC project comparison.....	603		
Prepare function block element for forc- ing.....	658		
Procedure step.....	629		
Reset cold.....	623		
Reset origin.....	624		
Reset warm.....	622		
Scale automatically.....	689		
Show next statement.....	631		
Single cycle.....	621		
Single step.....	630		
Start.....	620		
Stop.....	620		
Switch breakpoint.....	627		
Switch sequence control mode.....	633		
Toggle parallel mode.....	665		
Write values.....	632		
Comment			
CFC.....	279		
ST.....	236		
Compare PLC objects.....	17		
Comparing PLC objects and device data (online).....	51		
Comparing PLC objects online.....	51		
Comparing PLC project data and device da- ta (online)			

Detailed comparison.....	52	C0023.....	437
Options.....	54	C0026.....	438
Overview.....	51	C0027.....	438
Comparison		C0030.....	438
Application.....	20	C0031.....	439
Data server.....	39	C0032.....	439
Data source (data server).....	41	C0035.....	439
Data types.....	25	C0037.....	440
Devices (onboard I/O, Inline I/O, field buses, ...)	46	C0038.....	440
Folder.....	25	C0039.....	441
General module folder.....	18	C0040.....	441
Global variable list, persistent variables, network variables.....	27	C0041.....	442
Image pool.....	44	C0043.....	442
Interfaces, interface methods, interface properties.....	36	C0044.....	442
Library info.....	38	C0045.....	443
Library manager.....	37	C0046.....	443
Logic.....	19	C0047.....	444
Network variable lists, receiver end.....	28	C0048.....	444
Recipe definition.....	45	C0049.....	444
Recipe manager.....	44	C0050.....	445
Symbol configuration.....	42	C0061.....	445
Task.....	24	C0062.....	445
Task configuration.....	23	C0064.....	446
Text list.....	43	C0066.....	446
Visualization.....	42	C0068.....	446
Visualization manager.....	42	C0069.....	447
Compatibility		C0070.....	447
Library.....	641	C0072.....	448
Compilation		C0074.....	448
Conditional.....	413	C0075.....	448
Compile		C0076.....	449
Application.....	612	C0077.....	449
Compiled library.....	548	C0080.....	449
Compiled library in IndraWorks.....	638	C0081.....	450
Compiler		C0082.....	450
Option.....	715	C0084.....	450
Warning.....	716	C0086.....	451
Compiler errors.....	431	C0087.....	451
C0001.....	432	C0089.....	452
Compiler errors:		C0090.....	452
C0002.....	432	C0091.....	452
C0003.....	433	C0094.....	453
C0004.....	433	C0096.....	453
C0005.....	433	C0097.....	454
C0006.....	434	C0098.....	454
C0007.....	434	C0101.....	454
C0008.....	434	C0102.....	455
C0009.....	435	C0104.....	455
C0010.....	435	C0114.....	455
C0011.....	435	C0116.....	455
C0013.....	435	C0117.....	456
C0015.....	436	C0118.....	456
C0018.....	436	C0119.....	456
C0020.....	437	C0120.....	457
C0022.....	437	C0122.....	457
		C0125.....	458
		C0126.....	458
		C0130.....	458

Index

C0131.....	459	Compiler version.....	602
C0132.....	459	Compiling	
C0136.....	459	Application.....	611
C0139.....	460	Properties.....	705
C0140.....	460	Composer	
C0141.....	460	CFC.....	279
C0142.....	461	Conditional compilation.....	413
C0143.....	461	Configuration	
C0144.....	461	Device.....	73
C0145.....	462	Device editor.....	491
C0149.....	462	Configuration variable.....	291
C0161.....	462	Configure trace.....	774
C0162.....	463	Configuring the visualization style	
C0168.....	463	for library visualization styles or global	
C0169.....	463	visualization.....	1016
C0174.....	463	Connecting	
C0175.....	464	To device.....	698
C0177.....	464	Connection	
C0178.....	464	Connect.....	654
C0182.....	465	Remove.....	656
C0185.....	465	Reset.....	656
C0189.....	465	Select.....	655
C0190.....	466	Connection mark.....	657
C0195.....	466	CFC.....	279
C0196.....	466	Connector	
C0197.....	467	Reset.....	656
C0198.....	467	const_no_replaced, pragma attribute.....	383
C0199.....	467	const_replaced, pragma attribute.....	383
C0201.....	468	Constant.....	339
C0203.....	468	LTIME.....	340
C0204.....	468	REAL/LREAL.....	343
C0205.....	469	STRING.....	344
C0206.....	469	TIME.....	340
C0208.....	469	Typed.....	346
C0211.....	469	CONSTANT.....	339
C0212.....	470	Constant variables	
C0216.....	470	VAR CONSTANT.....	287
C0217.....	470	Constants.....	339
C0218.....	470	Numbers.....	342
C0219.....	471	WSTRING.....	345
C0221.....	471	Contact.....	266
C0222.....	471	Insert.....	666, 667
C0224.....	472	Content operator.....	312
C0227.....	472	CONTINUE.....	235
C0228.....	472	Continuous Function Chart (CFC)	
C0234.....	473	Page-oriented CFC.....	269
C0235.....	473	Control point	
C0236.....	474	CFC.....	277
C0237.....	474	Generate.....	657
C0238.....	474	Remove.....	657
C0239.....	474	Converting	
C0240.....	475	BOOL_TO.....	314
C0241.....	475	DATE_TO.....	318
C0242.....	476	Integer data type.....	316
C0243.....	476	LREAL_TO.....	316
C0380.....	476	REAL_TO.....	316
C0509.....	477	STRING_TO.....	319
C0511.....	477	TIME_OF_DAY_TO.....	317

TIME_TO.....	317	Connection type.....	163
To FBD.....	670	Data source type.....	162
To IL.....	671	Data transfer.....	163
To LD.....	671	Dialog “Initialize data source wizard -	
TO_<xxx>.....	315	Browse data source variables” dialog.....	482
TO_BOOL.....	315	Dialog “Initialize data source wizard -	
TRUNC.....	320	Provider settings”.....	482
TRUNC_INT.....	321	Edit communication.....	169
Core dump.....	208	Edit data source variables.....	167
Open memory view.....	595	Initially connecting devices with address	
COS.....	324	monitoring.....	164
Create		Map remote variables to local variables	
Symbol library.....	764	with a conforming data type.....	169
Text list for symbol library.....	765	Object “Data source” - Tab “General and	
CreateRecipe.....	744	diagnosis”.....	488
CreateRecipeNoSave.....	744	Object “Data source” - Tab “Type map-	
Creating and filling text list.....	114	pings”.....	485
Cross link list		Object “Data source” - Tab “Variables”.....	483
Limit results to current declaration.....	699	Select variables for data transfer.....	168
Cross reference		Tab “General and Diagnosis”.....	488
Update automatically.....	730	Tab “Type mappings”.....	485
Cross reference list		Update data interface.....	170
Place of use.....	129	Use removed data.....	171
Cross references for a specific symbol dec-		Data type.....	351
laration.....	129	_VECTOR.....	367
Cross references via text search by symbol		ANY.....	355
name.....	129	ANY <type>.....	355
Cross-reference list.....	590	BIT.....	359
View.....	590	Enumeration.....	374
Ctrl... 829, 837, 846, 855, 864, 876, 895, 914, 925,	928	Integer.....	352
CurrentVisu		LTIME.....	358
Variable to access the visualization.....	997	Message	405
Customize.....	641	REAL / LREAL.....	352
D		REFERENCE.....	361
D.....	348, 353	STRUCT.....	373
dar file.....	508	UNION.....	354
Data exchange.....	153	WSTRING.....	358
Data flow, pragma.....	383	Data type __VECTOR	
Data persistence.....	137	Determine optimal vector size.....	368
Change an existing declaration in the		Example.....	368
persistent variable list.....	141	Syntax.....	368
Double allocation of memory for instance		Syntax for index access.....	368
paths.....	141	Data type 'ARRAY'.....	363
Memory location for persistent function		Array of arrays.....	366
block instances.....	141	Array of fixed length.....	363
Recover data using the recipe manager.....	141	Array of variable length.....	366
Retain data using persistent variables.....	140	Examples.....	363, 364, 365
Data record, refer to Sample.....	200	Syntax for data access.....	363
Data recording		Data Unit Type.....	488
Trace.....	200	dataflow, pragma.....	383
Data source manager		DATE.....	353
“Communication” tab.....	486	DATE constant.....	341
“Data source manager” object.....	482	Example.....	341
Add initial data source.....	164	Syntax.....	341
Basics.....	162	DATE_AND_TIME.....	353
		DATE_AND_TIME constant.....	341
		Example.....	342
		Syntax.....	341

Index

DATE_TO.....	318	PLC shell.....	502
Debug		Status.....	511
CFC editor.....	273	Synchronized files.....	494
Core dump.....	208	Task list.....	510
Sequence control.....	186	Users and groups.....	507
Decimal number.....	342	Device repository.....	70
Declaration		Devices	
Go to.....	130	View.....	69
Declaration editor.....	85, 223	Diagnostic visualization.....	999
Edit declaration header.....	677	Restrictions.....	999
Option.....	722	Diagnostics	
Tabular/textual.....	85	Core dump.....	208
Declare		PLC shell.....	210
Short form mode.....	109	Dialog	
Declaring.....	83	'New breakpoint'.....	702
Array.....	86	Breakpoint properties.....	701
Automatic.....	730	Global visualization.....	1007
Global variable.....	88	Monitoring range.....	704
Decrease horizontal distance.....	768	Multiple download.....	621
Decrease vertical distance.....	769	Options.....	720
Default project.....	8	Options 'Device editor'.....	722
DeleteRecipe.....	745	Text list properties.....	720
DeleteRecipeFile.....	745	Trace configuration.....	737
Dereferencing.....	359	Dialog "Properties"	
Deselect all.....	774	Tab "Options Application build".....	707
devdesc.xml file.....	70	Dialog manager.....	1002, 1007
Device.....	490	Dialog visualization.....	1002, 1007
Application.....	491	Dialogs.....	700
Backup and restore.....	492	Select function block.....	700
Configuration.....	490, 491	Dialogs in the visualization.....	1007
Configure.....	73	Dialogs in visualization.....	1002
Files.....	495	DINT.....	352
I/O mapping.....	504	Dip switch	
Interactive login.....	713	Visualization element.....	966
Log.....	495	Directory for image files.....	734
Mapping I/Os.....	75	Directory for text lists.....	734
Options.....	713	Disabling	
PLC settings.....	497	Symbol library.....	763
PLC shell.....	210	Display PLC objects of an external	
Project Explorer.....	69	IndraWorks project	
Properties.....	713	Command.....	606
Simulation mode.....	179, 698	displaymode, pragma.....	384
User management.....	507, 508	DIV.....	299
Device description.....	70	DO.....	231, 232
Device editor		Documentation	
'Parameter' tab.....	491	Change record.....	1
Access rights.....	508	Download	
Applications.....	491, 492	Execute.....	214
Files.....	495	Downloading when logged in (online mode).....	215
Floating point exceptions in the PLC pro-		Downloading when logging in.....	214
gram.....	499	Dragging a variable dropping it to an ele-	
Generic.....	490	ment.....	819
I/O mapping.....	504	Dragging a variable to an element.....	819
IEC objects.....	506	DT.....	353
Information.....	511	Dual number	342
Log.....	495	dum file.....	507
Parameters.....	499	Durable color scales.....	942, 946, 951, 955
PLC settings.....	497	DUT	488

Add.....	489	Unique access to enumeration components.....	377
DWORD.....	352	EQ.....	311
Dynamic color display		Equalize height.....	770
Frame and fill color, visualization element		Equalize horizontal distance.....	767
property.....	826, 852, 911	Equalize size.....	770
Dynamic texts.....	825, 833, 843, 851, 861, 872, 891, 901, 918	Equalize vertical distance.....	768
		Equalize width.....	770
E		Error analysis.....	208
Edge detection.....	668	estimated-stack-usage, Pragma attribute.....	385
Edit control in visualization.....	1011	Event.....	829, 837, 846, 855, 864, 876, 895, 914, 925, 928
Editing mode		Exception.....	331
Graphical editor.....	224	Catch in IEC code.....	331
Editing the interface		Stop execution.....	700
of a visualization.....	782	Exception handling.....	331
Element		Visualization.....	758
Frame.....	866	Execute.....	266
Image.....	856	ST code in FBD/LF/IL.....	266
Polygon/Polyline/Bézier curve.....	838	Execute ST-code	
Element list.....	766	visualization.....	800
Element name.....	821, 831, 839, 848, 857, 868, 877, 879, 883, 885, 888, 896, 899, 909, 915, 920, 923, 927, 930, 931, 933, 936, 940, 943, 948, 952, 956, 961, 964, 967, 970, 973, 990, 992	EXIT.....	235
Element properties.....	594	Exit action.....	248
SFC.....	252	EXP.....	322
View, visualization.....	780	ExpandFully, pragma.....	388
ELSE.....	230, 232	Export/import	
ELSIF.....	230	I/O mapping.....	610
Empty function block		Text list.....	686
Insert.....	662	Expression, ST.....	226
Empty function block with EN/ENO	662	EXPT.....	323
EN/ENO		ExST.....	101
Add.....	649	Extend/reduce pin group.....	403
FBD/LD/IL.....	264	Extended Structured Text.....	101
End of pointer.....	948	External frame.....	941, 945, 949, 953
END_ACTION.....	427	External implementation.....	107
END_CASE.....	232	External variable.....	286
END_FOR.....	231	F	
END_FUNCTION_BLOCK.....	427	FALSE.....	339
END_IF.....	230	FALSE tapping.....	828, 836, 846, 854, 875, 894, 913, 928
END_PROGRAM.....	427	FB/KOP/AWL	
END_REPEAT.....	233	'Toggle parallel mode' command.....	665
END_VAR.....	280	FBD.....	90
END_WHILE.....	232	Convert.....	670
ENDTRY'.....	331	Option.....	723
Entry action.....	248	FBD/LD/IL editor.....	254
Enumeration.....	374	Line branching.....	669
Data type.....	374	Line branching LD.....	669
Declaration.....	374	Online mode.....	258
Declaration and initialization of enumeration variables.....	375	Toolbar.....	224
Default value.....	374	Unused parameters, FBD/LD.....	670
Enumeration with explicit basic data type....	375	File	
Initialization.....	374	Copy from/to control.....	217
Namespace.....	338	Link to object.....	712
Strict programming rules.....	375	Save.....	64
		Save as.....	64

Index

Files		
Device editor.....	495	
Fill color.....	822, 840, 844, 849, 909	
Fill properties.....	827, 844, 853	
Fill type.....	822, 840, 849, 910, 992	
Filling GlobalTextList.....	114	
FINALLY.....	331	
Find.....	132	
Find symbol.....	130	
Floating point exceptions.....	499	
Floating point exceptions in the PLC program.....	499	
Fond.....	890, 897	
Font.....	937, 941, 945, 950, 954, 957	
Fonts.....	823, 832, 841, 850, 859, 870, 878, 901, 911, 917	
FOR.....	231	
Forcing.....	183	
Display all forces.....	185	
Formatting		
Visualization text.....	797	
Frame		
configure.....	772	
Select visualization.....	772	
Update parameters.....	784	
Visualization element.....	866	
Frame color.....	822, 840, 844, 849, 909	
Frame element		
Program.....	783	
Frame parameters		
Declare, visualization.....	782	
Full screen mode.....	595	
Function.....	525	
Call.....	525	
Call via event.....	562	
Call with external implementation.....	107	
Transfer instance data via VAR_INPUT.....	527	
Transfer via POINTER in VAR_INPUT.....	527	
Transfer via REFERENCE TO to VAR_INPUT.....	528	
Function as operand.....	350	
Function block.....	523	
Call with external implementation.....	107	
Call, ST.....	235	
CFC.....	278	
Clean, FBD/LD.....	670	
Extend.....	146	
FBD/LD/IL.....	264	
FBD/LD/IL, insert.....	662	
Insert, FBD/LD/IL.....	662	
Repair.....	670	
Unused parameters, FBD/LD.....	670	
With EN/ENQ.....	662	
Function block diagram.....	90	
Function block input		
CFC.....	280	
Insert, FBD/LD.....	663	
Function block output		
CFC.....	280	
Function block types.....	351	
Function pointer.....	359	
Functionalities		
IEC 61131-3 Programming platform.....	1	
G		
Gateway.....	154	
GE.....	310	
General module folder		
Project settings.....	479	
Project-global objects and instances.....	479	
General settings		
Option.....	720	
Generating		
Image pool for symbol library.....	764	
Generating code.....	611	
Get.....	534	
GetLastError.....	753	
GetRecipeCount.....	750	
GetRecipeNames.....	750	
GetRecipeValues.....	751	
GetRecipeVariableNames.....	752	
Global namespace operator.....	337	
Global Network Variable List.....	158	
Global Network Variable List (receiver).....	519	
Add.....	519	
Global network variable list (sender)		
Add.....	519	
Global Network Variable List (sender).....	519	
Global text list		
Check.....	117	
Compare and export difference.....	116	
Create text.....	117	
Edit.....	114	
Edit with replacement file.....	118	
Export.....	114	
For static use.....	117	
Import file.....	115	
Remove text list entries.....	118	
Update IDs.....	117	
Global variable.....	285	
Declare.....	88	
Global variable in diagnostic visualization.....	999	
Global Variable List.....	512	
Namespace.....	337	
XML export.....	610	
XML import.....	610	
global_init_slot, Pragma.....	388	
GloballImagePool.....	121	
GlobalTextList		
Add language.....	684	
Check ID.....	687, 688	
Create.....	684	
Export/import.....	686	
Object.....	512	
Remove language.....	687	

- Remove unused text list entries..... 687
- Update ID..... 688
- GlobalTextList, see global text list..... 113
- Go to
 - Declaration..... 130
 - Definition..... 130
 - Network..... 671
- Gradient editor..... 814
- Gradient type..... 814
- Graphic language
 - Convert..... 670, 671
- Graphical editor, toolbar..... 224
- Grid
 - Visualization..... 734
- Group..... 507, 771
 - User management..... 57
- Group box
 - Visualization element..... 896
- Grouping
 - Create..... 658
 - Remove..... 658
- GT..... 309
- GVL..... 512
 - Namespace..... 337
- H**
 - hasattribute..... 413
 - hastype..... 413
 - hasvalue..... 413
 - Hazard warnings..... 2
 - Height..... 822, 848, 858, 869, 877, 880, 883, 886, 888, 896, 900, 909, 916, 920, 923, 927, 930, 931, 933, 936, 940, 944, 948, 952, 957, 961, 965, 967, 970, 974, 990, 992
 - Helpdesk..... 1019
 - Hexadecimal number..... 342
 - hide_all_locals, Pragma..... 390
 - hide, pragma..... 389
 - Hierarchy, ST..... 226
 - Histogram
 - Configure..... 958
 - Visualization element..... 955
 - Horizontal alignment..... 823, 832, 841, 850, 859, 870, 878, 901, 911, 917
 - Hotkey..... 829, 837, 846, 855, 864, 876, 895, 914, 925, 928
 - Hotline..... 1019
- I**
 - I..... 348
 - I/O mapping..... 74
 - All devices..... 81
 - Change address..... 79
 - Edit..... 609
 - Export to CSV..... 610
 - Import from CSV..... 610
 - Linking a device channel to a function
 - block instance..... 78
 - Monitoring..... 80
 - Procedure..... 75
 - Task list..... 189
 - Update..... 80
 - I/O mapping:
 - Device editor..... 504
 - Identification
 - Visualization element..... 760
 - Identifier, rules..... 419
 - IDs
 - Configure in visualization tools..... 794
 - IEC 61131-3 Programming platform
 - Functionalities..... 1
 - IF..... 230
 - IL..... 91
 - Convert..... 671
 - Delete line..... 667
 - Option..... 723
 - IL editor
 - Online mode..... 258
 - Image..... 773
 - Visualization element..... 856
 - Image file
 - Insert..... 676
 - Image files for visualization..... 734
 - Image pool..... 121, 515
 - Create..... 122
 - Global..... 121
 - Properties..... 712
 - Image selection..... 515
 - Image switcher
 - Visualization element..... 960
 - Implementation of FB_Something with the method FB_Init
 - Example..... 365
 - Import
 - I/O mapping..... 610
 - Increase horizontal distance..... 767
 - Increase vertical distance..... 769
 - Indentation
 - Text editor..... 583
 - INDEXOF..... 301
 - IndraLogic
 - Project Know-how protection..... 57
 - Project Write protection..... 57
 - IndraLogic 1x converter
 - Option..... 725
 - IndraLogic 2G visualization..... 757
 - IndraLogic Gateway..... 154
 - info.cfg file 70
 - Information
 - Device editor..... 511
 - Information representation
 - Names and abbreviations..... 5
 - init step..... 642
 - init_namespace, pragma..... 391

Index

init_on_onlchange, pragma.....	392	Right.....	664
Initial step.....	642	Insert device.....	70
Initialization.....	84	Insert elements for trace control.....	774
ARRAY.....	363	Inserting	
Avoid, pragma.....	400	Function block input, FBD/LD.....	663
Input variable, pragma.....	391	Function block, FBD/LD/IL.....	662
Namespace, pragma.....	391	Jump label, FBD/LD/IL.....	663
Online change, pragma.....	392	Jump, FBD/LD/IL.....	663
initialize_on_call, Pragma.....	391	Return, FBD/LD/IL.....	663
Input		Text in text list.....	685
CFC.....	277	Installing	
device.....	75	Library.....	636
FBD/LD/IL.....	264	Instance variables.....	287
Input and output variables.....	281	instance-path, pragma.....	392
Input assistance		Instruction list.....	91
Smart tag functions.....	110	INT.....	352
Input assistant.....	107	Integer data type.....	352
Image.....	865	Convert.....	316
List components.....	108	Interface.....	529
Option.....	730	Declare, visualization.....	782
Text search.....	587	Extend.....	149
Input configuration Hotkey.....	829, 837, 846, 855, 864, 876, 895, 914, 925, 928	Implement.....	147
Input configuration tap.....	828, 836, 846, 854, 875, 894, 913, 928	INTERFACE.....	529
Input configuration Toggle.....	829, 837, 846, 855, 875, 894, 913, 928	Interface editor.....	766
Input dialogs.....	1002	Interface method.....	534
Input help		Interface property.....	534
Categories.....	587	INTERNAL.....	532
Dialog.....	587	Internal frame.....	941, 945, 949, 953
Input memory.....	348	Invisibility.....	878, 930, 990, 993
Input support		Invisible input	
Short form mode.....	109	Visualization element.....	926
Input variable.....	280	io_function_block_mapping, pragma.....	393
Insert		io_function_block, pragma.....	393
Assignment, FBD/LD/IL.....	661		
Contact.....	664, 666, 667	J	
Contact parallel above.....	665	JMP.....	234
Contact parallel below.....	665	Jump.....	251
Contact right.....	664	CFC.....	278
Empty function block.....	662	Insert.....	646
Empty function block with EN/ENO.....	662	Insert after.....	647
Image file.....	676	Insert, FBD/LD/IL.....	663
Instruction line.....	667	Jump label	
Line branching.....	668, 669	FBD/LD/IL.....	265
Negated contact.....	666	Insert, FBD/LD/IL.....	663
Network.....	661	ST.....	234
Set coil.....	664	K	
Insert action association.....	645, 646	Keyboard action	
Insert after.....	648	Visualization.....	1018
Insert assignment, FBD/LD/IL.....	661	Keyboard configuration	
Insert contact		Visualization.....	791
Left.....	664	Visualization manager.....	998
Negated.....	666	Keyfile.....	13
Negated, parallel below.....	666	Keypad	
Parallel above.....	665	Visualization.....	997
Parallel below.....	665	Keys.....	829, 837, 846, 855, 864, 876, 895, 914, 925, 928

- keyword..... 427
- Keyword
 - __AdrInst..... 427
 - __BitOffset..... 427
 - __Cast..... 427
 - __Copy..... 427
 - __CRC..... 427
 - __FCall..... 427
 - __Init..... 427
 - __Lazy..... 427
 - __LocalOffset..... 427
 - __MaxOffset..... 427
 - __PropertyInfo..... 427
 - __RefAdr..... 427
 - __RELOC..... 427
 - __SystemScope..... 427
 - __TypeOf..... 427
 - __VarInfo..... 427
 - __Wait..... 427
 - Upper case..... 730
- L**
- L..... 348
- Label..... 937, 941, 945, 950, 954, 957
 - CFC..... 278
- Label height..... 937, 941, 945, 950, 954, 957
- Label offset..... 950, 954
- Ladder diagram..... 90
- Lamp
 - Visualization element..... 963
- Language
 - Currently set for visualization..... 997
 - Language files for visualization..... 734
 - Language in a visualization..... 734
 - Language switching
 - Visualization..... 800
- LD..... 90
 - Convert..... 671
 - Option..... 723
- LD editor
 - Online mode..... 258
- LE..... 310
- Libraries
 - Option..... 726
- Library
 - Add to project..... 220
 - Check compatibility..... 641
 - Create..... 743
 - Enable/disable the library explorer..... 590
 - Install..... 636
 - Mapping definition..... 726
 - Memory location..... 636
 - Namespace..... 338
 - Profile..... 636
 - Recipe management..... 743
 - Repository..... 636
 - Uninstall..... 636
 - Use function blocks..... 112
 - Visualization..... 1015
- Library development
 - Information..... 219
- Library development summary..... 743
- Library in IndraWorks..... 638
- Library info
 - Command..... 676
 - Functions to access properties..... 12
- Library manager..... 219, 516
 - Add library..... 672
 - Placeholder..... 675
 - Properties..... 673
 - Try to reload library..... 673
- Library project
 - Category..... 545
 - Compiled library..... 548
 - Sign..... 12, 548
- Library repository..... 219, 636
 - Add library..... 221
 - Export from the library repository..... 221
- LIMIT..... 308
- Line branching..... 669
 - Closed..... 268
 - Open..... 265
- Line branching start/end..... 268
- Line branching, IL
 - Element in FBP/LD/IL..... 265
- Line type..... 822, 827, 832, 840, 844, 849, 853, 859, 863, 869, 873, 910, 992
- Line width..... 822, 827, 832, 840, 844, 849, 853, 859, 863, 869, 873, 910, 992
- linkalways, pragma..... 394
- LINT..... 127, 352
- Listing components..... 108
- LN..... 322
- LoadAndWriteRecipe..... 745
- LoadFromAndWriteRecipe..... 746
- Loading to control..... 214
- LoadRecipe..... 747
- Locale variable..... 280
- Log
 - PLC..... 208
- LOG..... 322
- Log off the PLC
 - Device editor..... 495
- Logbook
 - PLC..... 208
- Logging in user..... 62
 - Command..... 597
- Logging out user
 - Command..... 598
- Login dialog in the visualization..... 1007
- Login dialog in visualization..... 1002
- Login, user..... 62
- Look-and-feel, see corporate identity..... 1017
- LREAL..... 352
- LREAL_TO..... 316

Index

LT.....	310	Show next statement.....	631
LTIME.....	358	Start application.....	620
Example.....	341	Stop application.....	620
Syntax.....	340	Switch breakpoint.....	627
LWORD.....	352	Switch display mode (binary, decimal, hexadecimal).....	636
M		Switch operating mode.....	634
M.....	348	View 'Open breakpoints'.....	625
M4-interface file for external library.....	612	Write values.....	632
Macro.....	251	Menu 'View'	
Display.....	647	'Memory' command.....	595
Exit.....	648	Library.....	590
Insert.....	647	Project Explorer.....	590
Insert after.....	647	Menu 'Commands'	
Magnification tool.....	224	Command 'Go to reference'.....	578
Main action.....	248	Menu 'Commands'	
Main scale.....	941, 945, 949, 953	'Previous bookmark' command.....	577
Mapping, refer to I/O image.....	75	Menu 'Create'.....	611
Marker memory.....	348	Menu 'Debug'.....	614
MAX.....	308	Execute reset cold.....	623
Max. label width.....	937, 941, 945, 950, 954, 957	Execute reset origin.....	624
Memory		Execute reset warm.....	622
Display core dump.....	595	Login to application.....	614
Memory consumption		Menu 'Debug'	
Recipe.....	200	Dialog 'Multiple download'.....	621
Memory range.....	348	Menu 'Edit'	
Memory view.....	595	Command 'Declare variable'.....	588
Menu 'Commands'		Command 'Input assistance'.....	586
Command 'Go to instance'.....	578	Command 'Insert file as text'.....	577
Menu 'Debug'	633	Command group 'Bookmark'.....	576
'Cancel forcing for all values'.....	633	Command group 'Refactoring'.....	579
Add all forces to monitoring list.....	629	Command group 'Search and replace'.....	573
Close core dump.....	636	Command group 'Browse'.....	577
Core dump commands.....	634	Command group 'Extended'.....	583
Create core dump.....	634	Command group 'Standard commands'.....	573
Determine next statement.....	631	Standard commands.....	573
Dialog 'Breakpoint property'.....	626	Menu 'Edit'	
Disable breakpoint.....	627	'Expand all' command.....	584
Edit breakpoint.....	627	'Find' command.....	573
Enable breakpoint.....	627	'Input wizard' command.....	587
Execute multiple download.....	621	'Replace' command.....	574
Execute online change.....	619	'Browse cross references' command.....	578
Execute single cycle.....	621	'Delete bookmark' command.....	577
Execute single step.....	630	'Go to definition' command.....	577
Execute to cursor.....	630	'Next bookmark' command.....	576
Execute to return.....	630	'Rename refactoring' command.....	579
Force values.....	631	'Show space' command.....	583
Generate boot application.....	617	'Toggle bookmark' command.....	576
Load application.....	618	'Dump call tree' command.....	578
Load core dump.....	635	Command 'Declare variable'.....	588
Load device logbook from core dump.....	635	Command 'Expand all'.....	584
Log out current device user.....	617	Command 'Add variable'.....	580
Log out of an application.....	617	Command 'Corresponding parenthesis'.....	586
Open call list.....	627	Command 'Go to'.....	585
Open monitoring list.....	628	Command 'Highlight up to corresponding parenthesis'.....	586
Procedure step.....	629	Command 'Insert file as text'.....	577
Show all forces.....	628	Command 'Make lower case'.....	586

Command 'Make upper case'.....	586	Function call.....	394
Command 'Rearrange variables...'.....	582	Limit monitoring ranges.....	194
Command 'Refresh affected connections'...	583	Monitor function.....	196
Command 'Remove variable'.....	582	Monitor function blocks.....	194
Command 'Search down'.....	575	Monitor in the declaration editor.....	192
Command 'Search down' (selection).....	576	Monitor methods.....	195
Command 'Search up'.....	575	Monitor properties.....	194
Command 'Search up' (selection).....	576	Monitor property accesses in the parent programming object.....	195
Menu 'Tools'.....	636	Monitor the return value of a function call....	196
Menu 'Tools'		Monitoring in the implementation (inline monitoring).....	192
Libraries.....	636	Monitoring interval.....	713
Menu 'View'.....	590	Partial monitoring of an array.....	194
Command group 'Other windows'.....	590	Pragma.....	394
Menu commands.....	572	Property.....	394
Advanced.....	583	SFC.....	236
Bookmark.....	576	Monitoring function.....	145
Browse.....	577	Monitoring the PLC program of the applica- tion at runtime.....	542
Declare variable.....	588	POU templates.....	543
Input assistance.....	586	MOVE.....	301
Insert file as text.....	577	MUL.....	298
Menu "Project".....	596	Multilingual PLC comments.....	65
Menu 'AS'.....	642	Multiplying a visualization element.....	817
Menu 'CFC'.....	648	Multitouch	
Menu 'FBD/LD/IL'.....	661	Setting.....	998
Menu 'Trace'.....	688	MUX.....	309
Menu 'Declarations' (persistence).....	678		
Menu 'Edit'.....	573	N	
Menu 'Image pool'.....	676	namespace.....	397
Menu 'Library'.....	671	Namespace	
Menu 'Recipes'.....	680	Automatic.....	730
Menu 'Text list'.....	684	Enumeration.....	338
Menu 'View'.....	590	GVL.....	337
Menu "Declarations".....	677	Library.....	338
Miscellaneous.....	698	NE.....	311
Refactoring.....	579	Negate.....	649
Message pragma.....	378	Negated contact.....	266
Meta information		Negation.....	667
Enter for project.....	12	Network.....	153
Method.....	287, 532	Comment.....	661
'FB_Exit'.....	427	FBD/LD/IL.....	263
'FB_Init'.....	427	Insert.....	661
'FB_Reinit'.....	427	Insert below.....	661
Add.....	532	Jump.....	671
Call.....	150	Topology.....	153
Call methods recursively.....	152	Network settings	
Call with external implementation.....	107	Properties.....	709
Interface.....	534	Network variables.....	158
Memory location for persistent function block instances.....	150	Properties.....	708
Method call.....	150	NEW.....	327
MIN.....	308	no_assign, pragma.....	398
MOD.....	300	no_check, pragma.....	399
Modifier in IL.....	259	no_copy, pragma attribute.....	399
Modifying PLC compiler version.....	9	no_instance_in_retain, pragma.....	400
Monitoring.....	191	no_virtual_actions, pragma.....	401
AS.....	236		
CFC editor.....	273		
Display all forces.....	185		

Index

- no-exit, pragma..... 400
- noinit, pragma..... 400
- Normal state..... 822, 840, 844, 849, 909
- NOT..... 302
- NOT, operator in pragma..... 418
- Noticing event in the visualization..... 1011
- Noticing keyboard events..... 1011
- Number constants..... 342
- Numpad
 - Visualization..... 997
- NVL (receiver)..... 519
- NVL (sender)..... 519
- O**
- Object
 - Access right..... 59
 - Action..... 540
 - Add, command..... 605
 - Edit object (offline)..... 608
 - General module folder..... 479
 - Library info..... 545
 - Link to file..... 712
 - Properties..... 704
 - Text list..... 553
 - Transition..... 541
- Object 'Property'..... 537
- obsolete, pragma..... 405
- Octal number..... 342
- OF..... 232
- On level up..... 771
- One level down..... 771
- One-dimensional array of ten integer elements
 - Example..... 363
- Online change
 - Execute..... 213
- Online change in the logged-in state (online mode)..... 214
- Online change when logging in..... 213
- Online mode
 - Force and write variables..... 274
 - Task monitoring..... 564
- Open dialog
 - Visualization..... 800
- Operand..... 352
- Operator
 - __COMPARE_AND_SWAP..... 335
 - __CURRENTTASK..... 334
 - __DELETE..... 326
 - __NEW..... 327
 - __VARINFO..... 332
 - __VCADD..... 368
 - __VCDIV..... 369
 - __VCDOT..... 370
 - __VCLOAD_LREAL..... 372
 - __VCLOAD_REAL..... 371
 - __VCMAX..... 370
 - __VCMUL..... 369
 - __VCSET_LREAL..... 371
 - __VCSET_REAL..... 371
 - __VCSQRT..... 370
 - __VCSTORE..... 372
 - __VCSUB..... 369
 - __XADD..... 336
 - Binding strength..... 226
 - Hierarchy..... 226
 - IL..... 259
 - INI..... 338
 - SEL..... 307
 - TEST_AND_SET..... 336
 - Operator __NEW
 - Use cases..... 328
 - Operator '__NEW'
 - Syntax..... 327
 - Operator 'SEL'..... 307
 - Operators..... 294
 - Overflow/underflow in the data type..... 295
 - Options
 - CFC editor..... 721
 - Declaration editor..... 722
 - FBS, LD and IL..... 723
 - General settings..... 720
 - IndraLogic 1x converter..... 725
 - Libraries..... 726
 - Safe FBD options..... 727
 - SFC editor..... 728
 - Smart coding..... 730
 - Syntax highlighting..... 731
 - Text editor, ST editor..... 731
 - Visualization..... 1016
 - Visualization styles..... 735
 - Visualizations..... 734
 - OR..... 302
 - OR_ELSE..... 304
 - OR, pragma..... 418
 - Order
 - One ahead..... 651
 - One back..... 651
 - Origin offset..... 950, 954
 - ORN..... 259
 - Output
 - CFC..... 278
 - Device..... 75
 - Reverse..... 649
 - Output memory..... 348
 - Output variable..... 281
- P**
- pack_mode, pragma..... 405
- Page
 - CFC..... 277
- Page size
 - Edit..... 648
- Page-oriented..... 272
- Panning tool..... 224

Parallel.....	643	PLC shell.....	210
Parameter list		Device editor.....	502
Configurable constants in libraries.....	513	PlcOperationControl.....	212
Parameters		Pointer.....	359, 948, 953
Edit.....	653	SUPER.....	291
Update.....	669	THIS.....	293
parameterstringof, pragma.....	790	POINTER.....	359
PARAMS.....	427	Pointer distance.....	950, 954
Password		Pointer end.....	953
Password manager.....	62	POINTER in VAR_INPUT.....	527
Password manager.....	57, 62	Pointer instrument 90°	
Path for files for text lists.....	734	Visualization element.....	939
Path for image files.....	734	Pointer instrument 180°	
Persistence.....	137	Visualization element.....	943
Clean gaps.....	678	Pointer instrument 360°	
Compare mechanisms.....	138	Visualization element.....	947
Lifespan of variables when calling online		Pointer start.....	953
commands.....	139	Pointer type.....	948, 953
Lifespan of variables when downloading		Polygon	
a boot project.....	139	Visualization element.....	838
Mechanisms for data retention.....	137	Polyline	
Rearrange gaps.....	678	Visualization element.....	838
Recipe.....	678	Position....	822, 848, 858, 869, 877, 880, 883, 886,
Variable configuration.....	679	888, 896, 900, 909, 916, 920, 923, 927, 930, 931,	
Persistence editor.....	514	933, 936, 940, 944, 948, 952, 957, 961, 965, 967,	
PERSISTENT		970, 974, 990, 992	
Declaration in the function block FB_A.....	289	Positioning.....	950, 954
Declaration in the persistent variable list		Potentiometer	
PersistentVars.....	288	Visualization element.....	951
Declaration in the program PLC_PRG.....	289	POU.....	520
Possible declaration locations.....	289	Add.....	521
Syntax of the declaration.....	288	Change type.....	677
Syntax of the declaration in POUs.....	288	POU for implicit check.....	145
Persistent variable.....	137	Power switch	
Create Persistent Variable List and de-		Visualization element.....	966
clare a persistent variable.....	143	Pragma	
Declare.....	143	Attribute 'pin_presentation_order_inputs/	
PERSISTENT.....	288	outputs'.....	404
Preserve data with recipes.....	142	Attributes.....	379
Save in recipe.....	144	Conditional.....	413
Persistent variable list.....	514	define.....	413
pingroup, pragma.....	403	hasattribute.....	413
Place of use		hastype.....	413
Variable.....	129	hasvalue.....	413
Placeholder		if.....	413
Library manager.....	675	Message.....	378
PLC communication		Region pragma.....	418
Gateway.....	154	Static analysis.....	127
PLC compiler version		undefine.....	413
Command.....	602	Use.....	110
PLC localization data.....	65	Use in the visualization interface editor.....	782
Export translation file.....	66	Prepare value.....	701
Import translation file.....	66	Preview display on a library visualization.....	759
Manage project languages.....	65	PRIVATE.....	532
Select project language.....	67	Processing order	
PLC log.....	208	in ST.....	226
PLC settings		Processing sequence	
Device editor.....	497	In SFC.....	237

Index

Profile		
Visualization.....	1016	
PROGRAM.....	522	
Programming		
Frame element.....	783	
Programming in the CFC editor.....	97	
Programming in the CFC Editor		
Create connection marks.....	97	
Insert element and wire with connecting lines.....	97	
Reduce the display of a function block.....	100	
Resolve collisions and fixedly set connecting lines using control points.....	98	
Progress bar		
Visualization element.....	929	
Project.....	7	
Add folder.....	605	
Create.....	8	
Create property with key.....	12	
Export.....	15	
File information.....	545	
Information.....	545	
Key for meta information.....	546	
Key VisuSymbolLibrary.....	546	
Meta information.....	545	
POUs for key.....	546	
Project settings.....	13	
Protection.....	57	
Query information.....	11	
Released.....	59	
Rights management.....	59	
Save.....	64	
Save as.....	64	
Select symbol library.....	763	
Statistics of objects.....	548	
Storage.....	59	
Transfer.....	15	
User management.....	60	
Project Explorer.....	69	
Enable/disable Project Explorer.....	590	
Rules.....	70	
Project file		
Access data.....	62	
Project setting		
SFC.....	714	
Visualization.....	719	
Project settings.....	545	
Command.....	596	
Make.....	13	
User management.....	60	
Properties		
Access control.....	706	
Bitmap.....	707	
Boot application.....	705	
CFC.....	709	
Compile.....	705	
Device.....	713	
General.....	705	
Image pool.....	712	
Library manager.....	673	
Link to file.....	712	
Monitoring.....	713	
Network settings.....	709	
Network variables.....	708	
Of an object.....	704	
SFC.....	712	
Task configuration.....	562	
Visualization object.....	1001	
PROPERTY.....	537	
PROTECTED.....	532	
Protection		
Project.....	57	
PUBLIC.....	532	
Push switch		
LED, visualization element.....	966	
Visualization element.....	966	
Q		
Q.....	348	
qualified_only, pragma.....	408	
Qualifiers for SFC actions.....	239	
R		
Radio button		
Visualization element.....	931	
READ_ONLY.....	427	
READ_WRITE.....	427	
ReadAndSaveRecipe.....	747	
ReadAndSaveRecipeAs.....	748	
ReadRecipe.....	749	
REAL.....	352	
REAL_TO.....	316	
REAL/LREAL constant.....	343	
Example.....	344	
REAL/LREAL constants		
Syntax of exponential notation.....	343	
Rearranging list.....	678	
Recipe.....	552	
Change values with recipes.....	198	
Create.....	198	
CreateRecipe.....	744	
CreateRecipeNoSave.....	744	
DeleteRecipe.....	745	
DeleteRecipeFile.....	745	
GetLastError.....	753	
GetRecipeCount.....	750	
GetRecipeNames.....	750	
GetRecipeValues.....	751	
GetRecipeVariableNames.....	752	
Insert.....	680	
Insert variable.....	680	
Load.....	681	
Load and write.....	682	
Load from file.....	199	
LoadAndWriteRecipe.....	745	

LoadFromAndWriteRecipe.....	746	Output.....	649
LoadRecipe.....	747	Remove.....	649
Memory consumption.....	200	Reset assignment	
Read.....	681	Multiple assignments.....	229
Read and write.....	683	R=, ExST.....	228
ReadAndSaveRecipe.....	747	Reset coil.....	267, 664
ReadAndSaveRecipeAs.....	748	ResetLastError.....	753
ReadRecipe.....	749	Restore values from recipe.....	679
RecipeManCommands.....	743	RETAIN	
ReloadRecipes.....	749	In a GVL.....	290
Remove.....	680	In a POU.....	290
Remove variable.....	683	Possible declaration locations.....	290
ResetLastError.....	753	Syntax in the declaration.....	290
Save.....	681	Retain variable	
Save values.....	678	RETAIN.....	290
SaveRecipe.....	748	Retain variables	
SetRecipeValues.....	752	Memory location of persistent function	
SetStoragePath.....	754	block instances.....	142
Write.....	682	Preserve data with retain variables.....	142
WriteRecipe.....	749	Retaining data with persistent variables	
Recipe definition.....	552	Download mechanism.....	140
Recipe file		Return	
Load.....	199	CFC.....	278
Recipe manager.....	549	FBD/LD/IL.....	265
RecipeManCommands.....	743	Insert, FBD/LD/IL.....	663
Recording, see data recording.....	738	RETURN.....	234
Rectangle		Rights	
Ellipse, visualization element.....	821	Command.....	601
rounded, visualization element.....	821	Rights management.....	57
Visualization element.....	821	Project.....	59
Refactoring.....	132	Rocker switch	
Add variable.....	133	Visualization element.....	966
Option.....	726	ROL.....	305
Remove variable.....	133	ROR.....	306
Rename.....	132	Rotary switch	
REFERENCE.....	361	Visualization element.....	969
Reference "Programming".....	223	Runtime system files	
REFERENCE TO to VAR_INPUT.....	528	Generate	612
Reference user interface.....	479	Runtime system service	
reflection, pragma attribute.....	409	Disable.....	211
Register files for text lists without path.....	734		
Register images without path.....	734	S	
Reloading library		Safe FBD options	
Command.....	673	Option.....	727
ReloadRecipes.....	749	Safety instructions.....	2
Remove		Sample	
Reset.....	649	Save in trace file.....	202
Remove horizontal distance.....	768	Show.....	200
Remove vertical distance.....	769	Save	
Removing		Project.....	59, 64
Set.....	649	Save access data.....	62
Removing text list support.....	688	SaveRecipe.....	748
Repairing		Scale.....	941, 945, 949, 953
Function block.....	670	Scale end.....	941, 945, 949, 953
REPEAT.....	233	Scale format (C-syntax)....	937, 941, 945, 950, 954, 957
Replace.....	132	Scale in 3D.....	941, 945, 949, 953
Reset			
Application.....	185		

Index

- Scale line width..... 941, 945, 949, 953
- Scale start..... 941, 945, 949, 953
- Scale width..... 941, 945, 949, 953
- Scan hardware..... 70
- SCE
 - LD, bypass function block..... 268
- Scopes of objects..... 70
- Scroll bar
 - Visualization element..... 914
- Select all..... 774
- Select insertion location..... 70
- Select menu 'Edit'
 - Command 'Remove uncomment selected lines"..... 585
 - Command 'Uncomment selected lines"..... 584
- Select the visualization element
 - for operation..... 1018
- Selector
 - CFC..... 279
- Semitransparency..... 815
 - Setting..... 998
 - Use in visualization..... 815
- Send to back..... 771
- Sequence
 - To the beginning..... 650
 - To the end..... 651
- Sequence control..... 186
- Service hotline..... 1019
- Set..... 534
 - Output..... 649
 - Remove..... 649
- Set assignment
 - Multiple assignments..... 228
 - S=, ExST..... 228
- Set coil..... 267
 - Insert..... 664
- Set, ExST..... 228
- Set/reset, FBD/LD/IL..... 668
- SetRecipeValues..... 752
- SetStoragePath..... 754
- Setting output connection, FBD/LD..... 668
- SFC..... 102
 - Action..... 247
 - Action qualifiers..... 239
 - Branching..... 249
 - Code generation..... 712, 714
 - Compile..... 712
 - Element properties..... 252
 - Implicit variables..... 240
 - Jog mode..... 241
 - Jump..... 251
 - Library..... 714
 - Macro..... 251
 - Online mode..... 236
 - Processing sequence..... 237
 - Program..... 102
 - Project setting..... 714
 - Prompt dialog to select the duplication mode..... 106
 - Properties..... 712
 - Step..... 245
 - Step time..... 240
 - Transition..... 245
- SFC editor..... 236
 - Option..... 728
 - Toolbar..... 224
- SFC flag..... 241
- Shadowing rules..... 424
 - Ambiguous access and qualified access..... 426
 - Search order in the application..... 425
 - Search order in the library..... 425
 - Searching in instance paths..... 426
- Shift..... 829, 837, 846, 855, 864, 876, 895, 914, 925, 928
- Shift downwards..... 677
- Shift upwards..... 677
- SHL..... 304, 305
- Short form mode..... 109
- Shortcut
 - Configure for visualization..... 791
- Show source position..... 592
- Sign
 - Library project..... 548
- Signal alert symbol..... 2
- Signal words..... 2
- Simulation..... 179
 - Command..... 698
 - For testing..... 179
- SIN..... 323
- SINT..... 352
- SIZEOF..... 301
- Slider
 - Visualization element..... 919
- Smart coding..... 107
 - Option..... 730
- SpinControl
 - Visualization element..... 922
- SQRT..... 321
- ST..... 101
 - Expression..... 226
 - Extended..... 101
- ST assignment operator, ... 227
- ST code in FBD, LD..... 266
- ST editor..... 225
 - Online mode..... 226
 - Option..... 731
- ST statement 'RETURN'..... 234
- Stack checking of recursive methods..... 385
- Standard axial..... 814
- Standard linear..... 814
- Standard radial..... 814
- Start screen..... 596
- Start visualization..... 773
- State variables..... 878, 930, 990, 993
- Static analysis

Example: Ignore several rules.....	128	System variable.....	211
Exclude implementation code.....	127	Operation control.....	211
Exclude programming objects with pragmas from the static analysis.....	128	T	
Pragma.....	127	Tab	
Static analysis light		CFC execution order.....	709
Project settings.....	716	Task groups.....	565
Static variable.....	286	Variable use.....	565
Status		Tab control element	
Device editor.....	511	Visualization element.....	884
SFC actions.....	240	Table	
SFC steps.....	240	Assigning data to the table, example.....	903
Step.....	245	Configure properties, example.....	904
Step status.....	240	Visualization element.....	897
Step transition		TAN.....	324
Insert.....	642	Tap upon entering if mouse freezes.....	828, 836, 846, 854, 875, 894, 913, 928
Insert after.....	643	Tapping.....	828, 836, 846, 854, 875, 894, 913, 928
Storage		Task.....	565
Project.....	59	Configuration tab.....	566
strict, pragma.....	410	Cycle times.....	208
STRING.....	353	Monitor.....	208
Data type.....	353	Monitoring.....	564
String constant.....	344	Processing sequence.....	136
STRING_TO.....	319	Statistics.....	208
STRUCT.....	373	Task configuration.....	136
Structure.....	373	Basic settings.....	562
Access.....	346	Basics.....	136
Data type.....	373	Create.....	137
Style property.....	1017	Object.....	562
SUB.....	299	Properties.....	562
Subordinate safety control.....	172	Task list	
Subrange types.....	377	Check.....	189
Subscale.....	941, 945, 949, 953	Device editor.....	510
Subscale position.....	941, 945, 949, 953	Task monitoring, online.....	564
subsequent, pragma.....	410	Temporary variable.....	286
SUPER.....	291	TEST_AND_SET	
Support.....	1019	Example.....	337
Switch frame		Functioning.....	336
visualization.....	800	Testing and debugging	
Switch variable		Step a program.....	181
visualization.....	800	Switch to "Debug" mode.....	181
Switching variable.....	874, 887	Text	
Symbol configuration.....	156	Alignment, visualization element property... ..	994
Add.....	553	Color.....	731
Configure synchronization with IEC tasks....	557	Control, visualization element property.....	994
Support for the current configuration and appropriate corrective actions.....	559	Font.....	731
Symbol file.....	553	Formatting, visualization element property.....	994
Symbol library.....	763	Text and language	
use in visualization.....	763	Use in the visualization.....	796
symbol, pragma.....	410	Text editor	
Symbols.....	4	Indentation code.....	583
Synchronized files		Option.....	731
Device editor.....	494	Tab 'Editing'.....	731
Syntax highlighting		Tab "Margin area".....	733
Option.....	731	Tab "Monitoring".....	733
System event.....	562		
Function call.....	562		

Index

Tab "Text area".....	732	Advanced setting.....	736
Text field		Advanced settings, visualization element....	985
Visualization element.....	908	Configure.....	689
Text index..... 825, 833, 843, 851, 861, 872, 891,	901, 918	Configure a visualization trace with trig-	
Text list..... 825, 833, 843, 851, 861, 872, 891, 901,	918	ger.....	979
Add language.....	684	Configure data recording.....	738
Compare and export difference.....	116	Configure the recording using the visuali-	
Edit.....	114	zation element.....	981
Enter text dynamically.....	120	Configure variable for visualization ele-	
Export.....	114	ment.....	983
Export all as text.....	685	Configure visualization element, example...	975
Export as Unicode text.....	685	Configured variable, visualization ele-	
Export/import.....	686	ment.....	984
For dynamic use.....	120	Configuring the variable.....	740
Import file.....	115	Data recording.....	207
Insert text.....	685	Display setting, visualization element.....	986
Remove language.....	687	Export configuration.....	988
Set in the visualization manager.....	999	File formats.....	202
Text list or global text list.....	113	Getting started.....	203
Text properties..... 823, 832, 841, 850, 859, 870,	878, 890, 897, 901, 911, 917	List.....	694
THEN.....	230	Load device trace.....	694
THIS.....	293	Load file.....	692
TIME.....	353	Load to trace editor.....	697
Example.....	340	Manage as file.....	207
Syntax.....	340	Online list.....	694
TIME constants.....	339	Open statistics.....	697
Time data types.....	353	Record data of IEC variable.....	203
TIME function.....	350	Remove from runtime system.....	694
TIME_OF_DAY.....	353	Retrieve y-value.....	689
TIME_OF_DAY constant.....	342	Runtime system buffer	736
Example.....	342	Runtime system component CmpTra-	
Syntax.....	342	ceMgr.....	201
TIME_OF_DAY_TO.....	317	Save samples in file.....	202
TIME_TO.....	317	Save to file.....	695
TIME().....	350	Show statistics.....	208
TO.....	231	Start data recording.....	696
TO_<xxx>.....	315	Stop data recording.....	696
TO_BOOL.....	315	Trace cursor.....	689
TOD.....	353	Trigger.....	202
Toggle variable/key variable		Upload.....	697
Visualization element button.....	893	Visualization element.....	972
Toggle/key variable		Visualization element with cursor, exam-	
Visualization element property..... 826, 852, 911		ple.....	978
Toggling.... 829, 837, 846, 855, 875, 894, 913, 928		Wizard of the visualization element.....	988
Toggling upon release if mouse frozen.... 829, 837,	846, 855, 875, 894, 913, 928	Trace configuration.....	204
Tool.....	590	'Download trace' command.....	690
Tools		Add an IEC variable.....	204
View in the visualization.....	778	Add variable.....	688
Tooltip		Assign task.....	204
Display, Visualization element property.....	994	Configuring a trace variable.....	204
Tooltip index..... 825, 833, 843, 851, 861, 872, 891,	901, 918	Configuring a trigger.....	205
Trace.....	200	Configuring the buffer for data on the run-	
Access all traces of the control.....	207	time system.....	206
		Configuring the display of the time axis.....	205
		Configuring the display of the trace varia-	
		ble.....	206
		Delete variable.....	204
		Deleting a trace variable.....	204

Editing the trace configuration at runtime....	206	Updating visualization text IDs.....	688
Export symbolically.....	691	Upper case	
Tracing a parameter.....	205	Keyword.....	730
Trace diagram		Use color gradient.....	822, 840, 849, 889
At runtime.....	569	Use one color.....	814
Convert to multi-channel representation.....	692	Use two colors.....	814
Convert to single-channel representation....	693	User account.....	62
Navigate with keyboard.....	570	User group.....	57
Navigate with mouse.....	569	User input	
Navigating in the data.....	207	Edit element.....	795
Scale automatically.....	689	User management.....	57
Zoom with mouse.....	692	“Access control” object.....	571
Trace editor.....	568	Command.....	596
Trace graph		Configure.....	597
Compress.....	689	Default setting of the dialogs, visualiza-	
Reset view to default.....	695	tion.....	998
Stretch.....	696	Log in user.....	597
Trace manager, refer to CmpTraceMgr.....	201	Log out user.....	598
Transfer variable		Project.....	60
VAR_IN_OUT CONSTANT.....	283	Rights.....	601
Transition.....	245	Users und groups.....	599
Transparency		User-defined attributes.....	379
Fonts, visualization element property.....	994	Users and groups	
Transparency color.....	952	Device editor.....	507
Trigger		Users and groups:	
Reset trace configuration.....	695	Command.....	599
TRUE.....	339	Using the visualization element.....	1018
TRUNC.....	320	USINT.....	352
TRUNC_INT.....	321		
TRY.....	331	V	
Typed constants.....	346	VAR.....	280
Typed literals.....	346	VAR CONSTANT	
U		Syntax.....	287
UDINT.....	352	VAR_ACCESS.....	427
UINT.....	352	VAR_CONFIG.....	291
ULINT.....	352	VAR_EXTERNAL.....	286
undefine.....	413	VAR_GLOBAL.....	285
Ungroup.....	772	VAR_IN_OUT.....	281
Unicode		VAR_IN_OUT_AS_POINTER, pragma.....	787
Text list.....	685	VAR_INPUT.....	280
Use for character string in visualization.....	997	VAR_INST.....	287
Uninstalling		VAR_OUTPUT.....	281
Library.....	636	VAR_PERSISTENT	
UNION.....	354	Add all instance paths.....	679
Unit.....	937, 941, 945, 950, 954, 957	VAR_STAT.....	286
Unit offset.....	950, 954	VAR_TEMP.....	286
Unlocking connection.....	655	Variable....	339, 828, 829, 836, 837, 846, 854, 855,
UNTIL.....	233		874, 875, 887, 894, 913, 928
Update		Access.....	346
Structured variables	683	Address.....	125
Update frame parameters in open visualiza-		Allocate memory space.....	410
tion.....	774	Bit access.....	347
Updating		Declare.....	83, 86
Parameters.....	669	Declare, tabular.....	85
Symbol library.....	764	Declare, textual.....	85
Updating structured variables.....	683	External.....	286
		Global.....	285

Index

Hide, pragma.....	389	Displaying the available.....	998
Initialize.....	84	Edit control.....	1011
Input.....	280	Element Button.....	887
local.....	280	Element list.....	793
Monitoring.....	191	Folder with image pool.....	719
Output.....	281	Folder with text list.....	719
Place of use.....	129	Generate ID.....	760
Representation format, pragma.....	384	Input configuration.....	800
Short form mode.....	109	Input dialog.....	1007
Static.....	286	Line element.....	830
Temporary.....	286	Menu commands.....	765
Value in online mode.....	191	Multiply element.....	817
Variable configuration.....	123	Notice keyboard events.....	1011
Variable declaration.....	83	Option.....	1016
shift downwards.....	677	Options.....	1016
Shift upwards.....	677	Preview on the library visualization.....	759
Variable name, rules.....	419	Profile.....	1016
VCADD.....	368	Project setting.....	719
Example: Adding.....	369	Properties.....	1001
VCDIV.....		selected as frame reference.....	772
Example: Dividing.....	370	Semitransparency.....	815
VCDIV.....	369	Show or enable grid lines.....	734
VCDOT.....	370	Storage location for file using the "List	
Example: Dot product.....	370	components" function.....	735
VCLOAD_LREAL.....	372	Storage location for image files.....	735
VCLOAD_LREAL.....		Storage location for text lists.....	735
Example: Vectorization.....	372	Tools view.....	778
VCLOAD_REAL.....	371	Trace wizard.....	988
Example: Vectorization.....	371	Using the symbol library	763
VCMAX.....	370	Visualization change.....	800
Example: Maximum vector.....	370	Visualization editor.....	777
VC MUL.....	369	Visualization element.....	
Example: Multiplying.....	369	Add.....	766
VCSET_LREAL.....	371	Configuration with interface property.....	719
VCSET_LREAL.....		Drag element to the visualization.....	760
Example.....	371	Drag the variable to an element.....	819
VCSET_REAL.....	371	Identification.....	760
Example.....	371	label.....	759
VCSQRT.....	370	List of all.....	793
Example: Square root.....	370	multiply.....	775
VCSTORE.....		select.....	759
Example: Saving.....	372	Show assignment on tools.....	794
VCSTORE.....	372	Show available visualization elements.....	759
VCSUB.....	369	With static text.....	117
Example: Subtracting.....	369	Visualization elements.....	795
VECTOR.....		Basis.....	821
Data type.....	367	Configuring elements - General informa-	
VendorException.....	497	tion.....	795
Vertical alignment.... 823, 832, 841, 850, 859, 870,		General control elements.....	876
878, 901, 911, 917		Lamps/switches/images.....	960
View.....		Measuring devices.....	934
Cross-reference list.....	590	Special control elements.....	972
Memory view.....	595	Visualization interface.....	999
Visualization.....		Visualization library.....	1015
activate for display variant.....	998	Visualization manager.....	996, 1007
Add object for the first time.....	794	Activate multitouch.....	998
available as frame reference.....	772	Keyboard configuration.....	998
Compatibility profile.....	1016	Setting.....	996

Test list setting.....	999
User management dialog.....	998
Visualization property element.....	993
Visualization style.....	1017
Option.....	1016
Workaround for deriving the visualization style.....	1016
Visualization styles	
Option.....	735
Visualizations	
Option.....	734
VisuDialogs.library.....	1002, 1007
VisuElems	
Library.....	1016
VUM_ChangePassword	
User management dialog, visualization.....	998
VUM_Login	
User management dialog, visualization.....	998
VUM_UserManagement	
User management dialog, visualization.....	998

W

W.....	348
Waiting symbol	
Flower, visualization element.....	991
warning disable, pragma.....	412
warning restore, pragma.....	412
Warnings.....	2
Watch list.....	196
Add variable.....	699
WHILE.....	232
Width.....	822, 848, 858, 869, 877, 880, 883, 886, 888, 896, 900, 909, 916, 920, 923, 927, 930, 931, 933, 936, 940, 944, 948, 952, 957, 961, 965, 967, 970, 974, 990, 992
WORD.....	352
Word addressing.....	348
Working in the CFC editor	
Commands when editing.....	272
Connect.....	271
Edit.....	270
Worksheet.....	648
Write protection.....	60
Write variable	
visualization.....	800
WriteRecipe.....	749
Writing.....	183
Writing variables to visualizations.....	1011
WSTRING.....	358
WSTRING constants.....	345

X

X.....	348, 822, 848, 858, 869, 877, 880, 883, 886, 888, 896, 900, 909, 916, 920, 923, 927, 930, 931, 933, 936, 940, 944, 948, 952, 957, 961, 965, 967, 970, 974, 990, 992
XOR.....	303

XORN.....	259

Y

Y.....	822, 848, 858, 869, 877, 880, 883, 886, 888, 896, 900, 909, 916, 920, 923, 927, 930, 931, 933, 936, 940, 944, 948, 952, 957, 961, 965, 967, 970, 974, 990, 992

Z

Zoom	
Factor.....	224
Graphical editor.....	224
Zoom in/zoom out	
Graphical editor.....	224

Notes

Notes

Bosch Rexroth AG

P.O. Box 13 57

97803 Lohr a.Main, Germany

Bgm.-Dr.-Nebel-Str. 2

97816 Lohr a.Main, Germany

Phone +49 9352 18 0

Fax +49 9352 18 8400

www.boschrexroth.com/electrics



R911396137