SHPPLCD
13373836

L-force *Engineering*

Software Manual

PC-based Automation

PLC Designer

Lenze

Contents

1 About this documentation 21

2 Lenze Specific Functions 22

2.1 Lenze Function Blocks and Functions 23

2.1.1 Library: L_PLCLoggingAccess.lib 23

2.1.2 Library: PLCDataManagerAccess.lib 24

2.1.3 Library: L_SM3_DriveUtil.lib 26

2.1.4 Library: L_SM3_DriveUtil.lib (I/O-System 1000) 36

2.2 CANopen / PROFIBUS / EtherCAT 52

2.3 PLC program for documentation of a cycle time violation 53

2.4 Axis Error Messages on the SoftMotion register 54

2.5 Assigning the transfer of a process image to a task 55

2.6 Data consistency during access to process image 56

3 Concepts and Base Components 57

3.1 Introduction and Basic Concepts 57

3.2 Differences to »PLC Designer« V2.3 58

3.3 Profiles and Plug-Ins 61

3.4 Project 62

3.5 Supported Programming Languages 63

3.6 What is a Program Organization Unit Object "POU" 64

3.7 Program Organization Objects - "POUs" 65

3.7.1 Program 67

3.7.2 Function 69

3.7.3 Function Block 71

3.7.4 Data Type Unit (DUT) 79

3.7.5 Method 81

3.7.6 Property 83

3.7.7 Interface 84

3.7.8 Action 86

3.7.9 External function, function block, method 88

3.7.10 Global Variable List - GVL 88

3.7.11 Global Network Variables List - GNVL ... 89
3.7.12 Persistent Variables ... 91
3.7.13 External File ... 91
3.7.14 Text List ... 93
3.7.15 Image Pool ... 100
3.7.16 Visualization ... 102
3.7.17 POUs for implicit checks ... 102
3.8 Device, Device tree ... 103
3.9 Application ... 107
3.10 Task Configuration ... 109
3.11 Communication ... 109
3.11.1 PLC Configuration ... 110
3.11.2 Network topology ... 110
3.11.3 Addressing and Routing ... 111
3.11.4 Structure of addresses ... 112
3.11.5 Data Server ... 116
3.11.6 Network Variables ... 117
3.12 Code Generation and Online Change ... 124
3.13 monitoring ... 127
3.14 Debugging ... 128
3.15 Printing ... 129
3.16 Visualization ... 129
3.17 Library Management ... 129
3.18 Security ... 135
4 Quickstart ... 136
4.1 Getting Help ... 136
4.2 Uninstallation, Update, Repair ... 136
4.3 Create and run a project ... 136

5 User Interface 152

5.1 Customizing the User Interface 152

5.2 User Interface in online mode 155

5.3 User Interface Components 156

6 Standard menus and commands 160

7 User and Access Rights Management 166

8 Visualization 168

8.1 Visualization in »PLC Designer« V3 168

8.2 Preconditions 170

8.3 Options 171

8.4 Creating a visualization object 171

8.5 Editing a visualization 171

8.6 Visualization Manager 171

8.7 Start Visualization 172

8.8 Frames, References, Interface - Placeholder 172

8.9 Text and Language in Visualization 173

8.10 Images in Visualization 176

8.11 External data sources 177

8.12 Visualization Libraries 177

8.13 Visualization Profiles 178

8.14 Target-Visualization 179

8.15 Web-Visualization 180

8.16 Visualization running in programming system 181

8.17 Visualization running on external devices 183

8.18 Visualization elements of type Visual 185

8.19 Visualization elements of type Complex Controls 186

8.19.1 Configuration of the Meter element 187

8.19.2 Configuration of the Bar Display element 190

8.19.3 Configuration of the Trace element 193

8.20 Visualization elements of type Windows Controls ... 199
8.20.1 Configuration of the Table element ... 199
8.21 Keyboard Usage in Online Mode ... 205
8.22 Noticing events and input actions ... 205
8.23 Input Dialogs ... 212

9 Symbol configuration ... 217

10 Source Control ... 219
10.1 Source Control via ENI ... 219
10.2 Differences to PLC Designer V2.3 ... 221

11 Project Documentation ... 222

12 Menu Commands sorted by Categories ... 223
12.1 Commands and Menus ... 223
12.2 Bookmarks ... 223
12.2.1 Bookmarks commands ... 223
12.2.2 Toggle Bookmark ... 223
12.2.3 Next Bookmark ... 224
12.2.4 Previous Bookmark ... 224
12.2.5 Clear Bookmarks ... 224
12.3 Breakpoints ... 224
12.3.1 Breakpoints commands ... 224
12.3.2 Toggle Breakpoint ... 225
12.3.3 New Breakpoint... ... 225
12.3.4 Breakpoint positions ... 228
12.3.5 Step Into ... 229
12.3.6 Step Over ... 229
12.3.7 Step Out ... 230
12.3.8 Set Next Statement ... 230
12.3.9 Run to Cursor ... 230
12.3.10 Show Current Statement ... 230

12.4 Browse Project ... 231
12.4.1 Go To Definition ... 231
12.4.2 Cross References List ... 232
12.5 Build ... 233
12.5.1 Build Commands ... 233
12.5.2 Build ... 234
12.5.3 Rebuild ... 234
12.5.4 Generate code ... 234
12.5.5 Generate runtime system files ... 235
12.5.6 Check All Pool Objects ... 237
12.5.7 Clean ... 237
12.5.8 Clean all ... 237
12.6 CAM ... 237
12.6.1 CAM Commands ... 237
12.6.2 read CAM online file ... 238
12.6.3 read CAM data from ASCII table ... 238
12.6.4 write CAM online file ... 238
12.6.5 write CAM data into ASCII table ... 238
12.7 CFC ... 239
12.7.1 CFC Editor commands ... 239
12.7.2 Edit Working Sheet ... 239
12.7.3 Negate ... 241
12.7.4 EN/ENO ... 242
12.7.5 Set/Reset ... 243
12.7.6 Execution Order ... 244
12.7.7 Edit Parameters ... 248
12.7.8 Connect Selected Pins ... 249
12.7.9 Reset Pins ... 249
12.7.10 Remove unused pins ... 249
12.8 CNC ... 250
12.8.1 CNC Commands ... 250
12.8.2 Show preprocessed path ... 251

12.8.3 Show interpolation points ... 251
12.8.4 Step suppression ... 252
12.8.5 Show grid ... 252
12.8.6 Show end points ... 252
12.8.7 Scroll path view ... 252
12.8.8 Move program ... 253
12.8.9 Scale program ... 254
12.8.10 Rotate program ... 254
12.8.11 Reverse direction ... 255
12.8.12 Renumber CNC program ... 255
12.8.13 Load program from ASCII file ... 255
12.8.14 Write program to ASCII file ... 256
12.8.15 Import DXF file ... 256
12.8.16 Analyze dynamics ... 257
12.8.17 Set variables ... 260
12.9 Clipboard ... 261
12.9.1 Clipboard commands - Edit commands ... 261
12.9.2 Cut ... 261
12.9.3 Copy ... 262
12.9.4 Paste ... 262
12.9.5 Delete ... 263
12.9.6 Select all ... 263
12.10 Declaration ... 263
12.10.1 Tabular Declaration Editor - Declaration ... 263
12.10.2 Edit Declaration Header ... 263
12.10.3 Insert ... 263
12.10.4 Move down ... 264
12.10.5 Move up ... 264
12.11 Devices ... 264
12.11.1 Devices commands ... 264
12.11.2 Scan ... 269
12.11.3 Device Repository ... 275

12.12 Device Communication ... 277
12.12.1 Device Communication commands ... 277
12.12.2 Add Device ... 278
12.12.3 Add Gateway ... 279
12.12.4 Change Device Name... ... 280
12.12.5 Connect to local Gateway... ... 280
12.12.6 Delete selected Device ... 287
12.12.7 Edit Gateway ... 288
12.12.8 Scan for Device by Name ... 288
12.12.9 Scan for Device by Address ... 288
12.12.10 Scan Network ... 288
12.12.11 Set Active Path ... 289
12.13 FBD/LD/IL ... 289
12.13.1 FBD/LD/IL Editor commands ... 289
12.13.2 Insert Network ... 289
12.13.3 Insert Network below ... 290
12.13.4 Toggle network comment state ... 290
12.13.5 Insert Assignment ... 291
12.13.6 Insert Box ... 292
12.13.7 Insert Empty Box ... 295
12.13.8 Insert Jump ... 296
12.13.9 Insert label ... 298
12.13.10 Insert Return ... 298
12.13.11 Insert Input ... 298
12.13.12 Insert Coil ... 299
12.13.13 Insert Set Coil ... 299
12.13.14 Insert Reset Coil ... 299
12.13.15 Insert Contact ... 299
12.13.16 Insert Negated Contact ... 301
12.13.17 Insert Contact right ... 301
12.13.18 Insert Contact Parallel below ... 301
12.13.19 Insert Negated Contact Parallel below ... 301

12.13.20 Insert Contact Parallel above ... 302
12.13.21 Paste contacts: Paste below ... 302
12.13.22 Paste contacts: Paste right after ... 302
12.13.23 Paste contacts: Paste above ... 303
12.13.24 Insert IL line below ... 303
12.13.25 Delete IL line ... 303
12.13.26 Negation ... 303
12.13.27 Edge Detection ... 304
12.13.28 Set/Reset ... 305
12.13.29 Set output connection ... 305
12.13.30 Insert Branch ... 306
12.13.31 Insert Branch above ... 307
12.13.32 Insert Branch below ... 307
12.13.33 Update parameters ... 307
12.13.34 Remove Unused FB Call Parameters ... 308
12.13.35 View as function block diagram ... 308
12.13.36 View as ladder logic ... 309
12.13.37 View as instruction list ... 309
12.14 Find/Replace ... 310
12.14.1 Find/Replace commands ... 310
12.14.2 Find ... 310
12.14.3 Replace ... 313
12.14.4 Find Next ... 314
12.14.5 Find Next Selected ... 314
12.14.6 Find Previous ... 314
12.14.7 Find Previous Selected ... 314
12.15 File ... 315
12.15.1 File commands ... 315
12.15.2 Project Settings - Users and Groups ... 358
12.16 Help ... 363
12.16.1 Help commands - Online Help ... 363
12.16.2 Contents... ... 363

12.16.3 Index... 363
12.16.4 Search... 364
12.16.5 Synchronize Contents... 364
12.16.6 About... 364
12.17 ImagePool... 364
12.17.1 Image Pool commands... 364
12.17.2 Create GlobalImagePool... 365
12.17.3 Create Image File List... 365
12.18 Installation... 365
12.18.1 Library Repository... 365
12.18.2 Visual Element Repository... 370
12.19 Library Manager... 376
12.19.1 Library Manager commands... 376
12.19.2 Add Library... 376
12.19.3 Properties... 380
12.20 Message View... 382
12.20.1 Message View commands... 382
12.20.2 Go To Source Position... 382
12.20.3 Next Message... 382
12.20.4 Previous Message... 382
12.21 Objects... 383
12.21.1 Object Commands... 383
12.21.2 NavigatorControl... 390
12.21.3 PLCopenXML... 391
12.22 Online... 394
12.22.1 Online commands - Online Commands... 394
12.22.2 Login... 396
12.22.3 Logout... 399
12.22.4 Create boot application... 399
12.22.5 Download... 400
12.22.6 Online Change... 400
12.22.7 Source download... 402

12.22.8 Source upload ... 402
12.22.9 Source download to connected device ... 402
12.22.10 Start ... 403
12.22.11 Stop ... 403
12.22.12 Single Cycle ... 403
12.22.13 Multiple Download... ... 404
12.22.14 Reset warm ... 406
12.22.15 Reset cold ... 407
12.22.16 Reset origin ... 407
12.22.17 Write Values ... 407
12.22.18 Force Values, Prepare value dialog ... 408
12.22.19 Unforce values ... 411
12.22.20 Add All Forces to Watchlist ... 412
12.22.21 Release Forcelist ... 412
12.22.22 Simulation ... 412
12.22.23 Flow Control ... 413
12.22.24 Display Mode ... 416
12.22.25 Connect to <device> ... 417
12.22.26 Disconnect from Device ... 417
12.22.27 Reset origin device ... 417
12.23 Preferences ... 417
12.23.1 Preferences ... 417
12.23.2 Options, »PLC Designer« V2.3 converter ... 433
12.23.3 Options, CFC Editor ... 434
12.23.4 Options, Declaration Editor ... 435
12.23.5 Options, FBD, LD and IL editor ... 436
12.23.6 Options, Features ... 439
12.23.7 Options, Device editor ... 442
12.23.8 Options, SFC Editor ... 442
12.23.9 Options SFC ... 445

12.23.10 Options, Source Control ... 448
12.23.11 Options, Visualizations ... 450
12.23.12 Customize Dialogs ... 452
12.24 Print ... 458
12.24.1 Print commands ... 458
12.24.2 Document ... 459
12.24.3 Page Setup ... 460
12.24.4 Print ... 469
12.25 Recipe ... 469
12.25.1 Recipe Commands ... 469
12.25.2 Add a new recipe ... 470
12.25.3 Remove recipes ... 470
12.26 SmartCoding ... 471
12.26.1 Input Assistant ... 471
12.26.2 Auto Declare ... 473
12.27 SFC ... 475
12.27.1 SFC Editor commands ... 476
12.27.2 Init step ... 476
12.27.3 Insert step-transition ... 476
12.27.4 Insert step-transition after ... 478
12.27.5 Parallel ... 478
12.27.6 Alternative ... 479
12.27.7 Insert branch ... 479
12.27.8 Insert branch right ... 479
12.27.9 Insert action association ... 481
12.27.10 Insert action association after ... 481
12.27.11 Insert jump ... 482
12.27.12 Insert jump after ... 482
12.27.13 Insert macro ... 483
12.27.14 Insert macro after ... 483
12.27.15 Zoom into macro ... 484
12.27.16 Zoom out of macro ... 484

12.28 Source Control ... 485
12.28.1 Source Control commands ... 485
12.28.2 Change Source Control... ... 486
12.28.3 Open From Source Control... ... 489
12.28.4 Add Objects From Source Control ... 490
12.28.5 Get Latest Version - Entire Project ... 491
12.28.6 Get Latest Version - Selected Objects ... 491
12.28.7 Checkout ... 491
12.28.8 Checkin ... 492
12.28.9 Undo Checkout ... 493
12.28.10 Show Differences ... 493
12.28.11 History... ... 494
12.28.12 Project History... ... 496
12.28.13 Label ... 498
12.28.14 Refresh Status ... 498
12.28.15 Login... ... 499
12.29 Text Editor ... 500
12.29.1 Text Editor commands ... 500
12.29.2 Overwrite Mode ... 500
12.29.3 Go To Line... ... 500
12.29.4 Insert File As Text... ... 500
12.29.5 Make Lowercase ... 501
12.29.6 Make Uppercase ... 501
12.29.7 Go To Matching Bracket ... 501
12.29.8 Select To Matching Bracket ... 501
12.30 Textlist ... 502
12.30.1 Text list commands ... 502
12.30.2 Add language ... 502
12.30.3 Remove language ... 503
12.30.4 Create Global Text List ... 503
12.30.5 Import/Export Textlists ... 503

12.30.6 Update Visualization Text Ids ... 509
12.30.7 Check Visualization Text Ids ... 509
12.30.8 Remove Unused Text Ids ... 509
12.31 Trace ... 510
12.31.1 Trace commands ... 510
12.31.2 Download Trace ... 510
12.31.3 Start / Stop Trace ... 511
12.31.4 Reset Trigger ... 511
12.31.5 Cursor ... 511
12.31.6 Mouse scrolling ... 512
12.31.7 Mouse zooming ... 512
12.31.8 Default appearance ... 513
12.31.9 Stretch ... 513
12.31.10 Compress ... 513
12.31.11 Multi Channel ... 514
12.31.12 Online List ... 515
12.31.13 Upload Trace ... 515
12.31.14 Load trace... ... 516
12.31.15 Save Trace... ... 516
12.31.16 Shortcuts for trace ... 516
12.32 Undo/Redo ... 517
12.32.1 Undo/Redo commands - Edit Commands ... 517
12.32.2 Undo ... 517
12.32.3 Redo ... 518
12.33 User Management ... 518
12.33.1 User Management commands ... 518
12.33.2 Logon ... 519
12.33.3 Logoff ... 521
12.33.4 Permissions... ... 522

12.34 Visual Commands 524
12.34.1 Visual Commands 524
12.34.2 Add Visualelement 525
12.34.3 Select all 525
12.34.4 Select none 525
12.34.5 Group 525
12.34.6 Ungroup 526
12.34.7 Frame selection 526
12.34.8 Interface Editor 527
12.34.9 Hotkeys Configuration 527
12.34.10 Element List 528
12.34.11 Background 528
12.34.12 Order 529
12.34.13 Alignment 529
12.34.14 Activate keyboard usage 530
12.35 Views 531
12.35.1 Cross Reference View 531
12.35.2 Breakpoints 533
12.35.3 Default Navigators 536
12.35.4 PLC Designer Frame 537
12.35.5 MessageView 553
12.35.6 SFC Implementation Object 554
12.35.7 Source Control 555
12.35.8 ToolBox 556
12.35.9 Visualization Element 556
12.35.10 Watch 557
12.36 Window 558
12.36.1 Window commands -»PLC Designer« Frame 558
12.36.2 Dock 558
12.36.3 Float 559
12.36.4 Auto hide 559
12.36.5 Next Editor 559

12.36.6 Previous Editor ... 559
12.36.7 Next Pane ... 560
12.36.8 Previous Pane ... 560
12.36.9 Close Editor ... 560
12.36.10 Close All Editors ... 560
12.36.11 New Horizontal Tab Group ... 561
12.36.12 New Vertical Tab Group ... 561
12.36.13 Reset Window Layout ... 561
12.36.14 Window <n> ... 562
12.36.15 Windows ... 562

13 General Editors ... 563

13.1 CAM Editor ... 563
13.1.1 A brief survey ... 563
13.1.2 Definition of a SoftMotion CAM disk ... 564
13.1.3 Creation of CAM disk ... 565
13.1.4 Edit a CAM ... 566
13.1.5 Compilation of the CAM ... 573
13.1.6 Using CAMs - General Hints ... 573
13.1.7 CAM Data Structures ... 580
13.1.8 CAM: Element Properties ... 584

13.2 CFC Editor ... 586
13.2.1 CFC - Continuous Function Chart - Language ... 587
13.2.2 Cursor positions in CFC ... 587
13.2.3 CFC Elements / Toolbox ... 588
13.2.4 Insert and arrange elements ... 593
13.2.5 CFC Editor in online mode ... 595

13.3 CNC Editor ... 597
13.3.1 Text Editor ... 603
13.3.2 CNC Table Editor ... 643
13.3.3 Graphic Editor ... 645
13.3.4 Properties of a Path Element ... 648
13.3.5 ToolBox ... 649

13.3.6 CNC object ... 654
13.3.7 CNC Online ... 659
13.4 CNC Settings Editor ... 659
13.4.1 Parameter editor ... 662
13.5 Data Source Editor ... 663
13.5.1 Add data server ... 664
13.5.2 Add data source ... 665
13.5.3 Data Source configuration ... 666
13.5.4 Use data sources in visualizations ... 672
13.6 Declaration Editor ... 673
13.7 DUT Editor ... 678
13.8 FBD/LD/IL Editor ... 678
13.8.1 Languages ... 679
13.8.2 Modifiers and operators in IL ... 682
13.8.3 Working in the FBD and LD editor view ... 684
13.8.4 Working in the IL editor view ... 687
13.8.5 Cursor positions in FBD, LD and IL ... 693
13.8.6 FBD/LD/IL menu ... 695
13.8.7 FBD/LD/IL Elements ... 696
13.8.8 FBD/LD/IL Editor in Online Mode ... 705
13.9 Global Variables List Editor ... 710
13.10 Library Manager Editor ... 710
13.10.1 Library Manager Editor window ... 711
13.10.2 Libraries menu ... 714
13.11 Network Variables List Editor ... 714
13.12 Recipe Manager Editor ... 715
13.12.1 Recipe Definition ... 717
13.12.2 RecipeManCommands ... 720
13.12.3 Recipe Manager in online mode ... 727
13.13 SFC Editor ... 728
13.13.1 SFC - Sequential Function Chart ... 729
13.13.2 Cursor Positions in SFC ... 730

13.13.3 Working in the SFC Editor ... 731
13.13.4 SFC Element Properties ... 733
13.13.5 SFC Elements / Toolbox ... 735
13.13.6 Qualifier ... 741
13.13.7 Implicit variables - SFC flags ... 742
13.13.8 Sequence of processing in SFC ... 749
13.13.9 SFC Editor in online mode ... 751
13.14 ST Editor ... 752
13.14.1 ST-Editor in Online Mode ... 753
13.14.2 Structured Text ST / ExST ... 756
13.15 Symbol Configuration Editor ... 768
13.16 Task Editor ... 770
13.16.1 Task Configuration ... 770
13.16.2 Task Editor, Usage ... 771
13.16.3 Task Editor in online mode ... 778
13.17 Trace Editor ... 780
13.17.1 Trace Editor ... 780
13.18 Visualization Editor ... 797
13.18.1 Inserting Visualization Elements, Toolbox ... 798
13.18.2 Selecting elements ... 800
13.18.3 Position, Size, Alignment, Order ... 801
13.18.4 Grouping elements ... 801
13.18.5 Visualization Element Properties ... 801
13.18.6 Visualization Interface Editor ... 835
13.18.7 Hotkeys Confguration Editor ... 839
13.18.8 Element List Editor ... 840
13.19 Visualization Manager Editor ... 842
13.19.1 Target-Visualization Editor ... 845
13.19.2 Web-Visualization Editor ... 846
13.20 Watch List Editor ... 847
13.20.1 Create watch list ... 847
13.20.2 Watch list in online mode ... 849

14 Device Editors 850

14.1 Generic Device Editor 850

14.2 CANbus Configuration Editor 872

14.3 EtherCat Device Editor 892

14.4 Profibus DP Device Editor 918

14.5 SoftMotion Device Editor 928

14.5.1 Softmotion concept and survey on components 928

14.5.2 The Softmotion Drive Interface 930

14.5.3 Configuration within the Device Softmotion Editor 932

14.5.4 Configuration Drive Logical 935

14.5.5 Survey on special use cases 937

14.5.6 Example Applications 939

15 Command Line Options 957

15.1 Starting PLC Designer.exe 957

16 Programming Reference 958

16.1 Declaration 958

16.1.1 Variables Declaration 958

16.1.2 Recommendations on the naming of identifiers 960

16.1.3 Variables Initialization 966

16.1.4 Arbitrary expressions for variable initialization 967

16.1.5 Declaration Editor 967

16.1.6 Autodeclaration Dialog 967

16.1.7 Shortcut Mode 968

16.1.8 AT Declaration 969

16.1.9 Keywords 970

16.1.10 Local Variables - VAR 974

16.1.11 Input Variables - VAR_INPUT 974

16.1.12 Output Variables - VAR_OUTPUT 974

16.1.13 Input and Output Variables - VAR_IN_OUT 975

16.1.14 Global Variables - VAR_GLOBAL 975

16.1.15 Temporary Variables - VAR_TEMP 976

16.1.16 Static Variables - VAR_STAT ... 976
16.1.17 External variables - VAR_EXTERNAL ... 976
16.1.18 Attribute Keywords for Variable Types ... 976
16.1.19 Remanent variables - RETAIN, PERSISTENT ... 977
16.1.20 Constants - CONSTANT, Typed Literals ... 979
16.1.21 Variables configuration - VAR_CONFIG ... 981
16.1.22 Declaration and Initialization of User Defined Data Types ... 982
16.1.23 Extensible Functions - PARAMS ... 983
16.1.24 FB_init, FB_reinit Methods ... 983
16.1.25 FB_Exit Method ... 985
16.1.26 Pragma Instructions ... 986
16.1.27 List Components Functionality ... 1018
16.2 Data Types ... 1019
16.2.1 Standard Data Types ... 1020
16.2.2 Extensions to Standard IEC1131-3 ... 1022
16.2.3 User Defined Data Types ... 1028
16.3 Operators ... 1042
16.3.1 IEC Operators and norm-extending Functions ... 1042
16.3.2 Arithmetic Operators ... 1043
16.3.3 Bitstring Operators ... 1048
16.3.4 Bit-Shift Operators ... 1051
16.3.5 Selection Operators ... 1055
16.3.6 Comparison Operators ... 1059
16.3.7 Address Operators ... 1062
16.3.8 Calling Operator ... 1064
16.3.9 Type Conversion Functions ... 1064
16.3.10 Numeric Functions ... 1073
16.3.11 IEC extending Operators ... 1079

16.4 Operands 1092
16.4.1 Constants 1092
16.4.2 Variables 1097
16.4.3 Addresses 1099
16.4.4 Functions 1102

17 Devices 1103
17.1 Motion Controller 1103
17.1.1 Automatic detection of the I/O compound module topology 1107
17.1.2 Adjusting the cycle time of access to the I/O compound module 1108

18 PLC Designer Libraries 1109

19 Index 1111

1 About this documentation

This documentation contains information on the PLC programming with the »PLC Designer« and makes the introduction to the powerful language constructs of IEC 61131 easier for you.

Target group

This documentation addresses to qualified personnel in accordance with IEC 364.

Information on the validity

The information in this documentation is valid for the following Lenze software:

Software	From software version
L-force »PLC Designer«	3.0.0

Document history

Version			Description
1.0	06/2008	TD29	First edition for the » PLC Designer«
2.0	08/2008	TD29	Revision
3.0	05/2009	TD29	Revision
4.0	10/2010	TD29	Update on software V3.4
5.0	03/2011	TD29	Update on software V3.5

Informations and utilities on Lenze products can be found in the "Downloads" area under: http://www.Lenze.com

Liability

All information given in this documentation has been selected carefully and tested for compliance with the hardware and software described. Nevertheless, discrepancies cannot be completely ruled out. We do not accept any responsibility or liability for any damage that may occur. Required corrections will be included in updates of this documentation.

This documentation is based on the original »PLC Designer« documentation by the company 3S - Smart Software Solutions GmbH (Kempten). The screenshots shown may deviate from the »PLC Designer« representation.

Trademarks

Microsoft, Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Adobe and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the U.S.A. and/or other countries.

All other product names contained in this online documentation are trademarks of the corresponding owners.

2 Lenze Specific Functions

»PLC Designer« is based on the IEC development environment CoDeSys (Controller Development System) of the company 3S - Smart Software Solutions GmbH, Kempten.

In contrast to CoDeSys, »PLC Designer« does **not** support the following functions/hardware:

- CanDevice (CANopen slave)
- DeviceNet module
- Encoder and Sercos interfaces
- Hardware scan/status/diagnosis from the target system
- »PLC Designer« HMI
- »PLC Designer« target visualisation

2.1 Lenze Function Blocks and Functions

2.1.1 Library: L_PLCLoggingAccess.lib

Function L_AddLog - Creating logbook entry

Function library: L_PLCLoggingAccess.lib

This function makes it possible to make an entry in the logbook of the L-force controller. It supplies the return value TRUE when the entry has been made.

Inputs

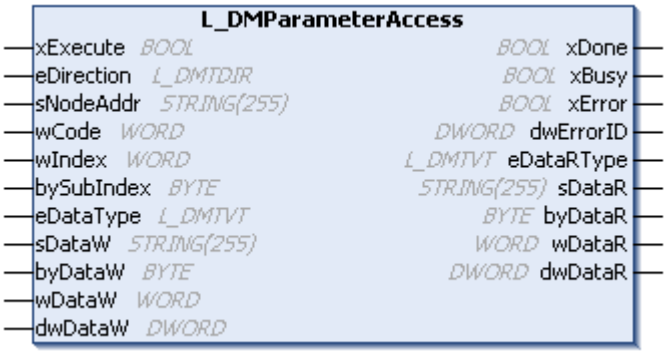
Designator / Data type	Info / Setting options
eSeverity / L_TSeverity	Logging severity - Risk of the error L_FatalError L_Error, L_Warning L_Information The Enum parameter eSeverity (IEC_INT) is sent as the severity (BYTE) to Add2Log.
eFacility / L_TFacility	Logging facility: - Area out of which the error is reported L_PLC_USER_LOG_SCOPE1 L_PLC_USER_LOG_SCOPE2 L_PLC_USER_LOG_SCOPE3 The Enum parameter eFacility (IEC_INT) is sent as the facility (WORD) to Add2Log.
sLogPayLoad / STRING(80)	Logbook entry The Ansi string from "sLogPayLoad" is sent as a 1-byte string parameter to Add2Log. ("hs", SLOGPAYLOAD)

2.1.2 Library: PLCDataManagerAccess.lib

Function block L_SMC_ParameterAccess_FB

Function library: PLCDataManagerAccess.lib

With the function block L_SMC_ParameterAccess_FB, the device parameters of the L-force controller can be read and written by the PLC.



Note: The PLC block L_SMC_ParameterAccess_FB does not support any codes that contain float/double parameters (e.g. supply voltage C00205 or rechargeable-battery voltage C00207).

If this block is nevertheless used on such codes, the error number 532 ("error if the format from the code number is not BYTE, WORD, DWORD or BSTR") is indicated.

Inputs

Designator / Data type	Info / Setting options
xExecute / BOOL	Rising edge starts access
eDirection / L_DMTDIR	Access direction L_DM_READ - Read L_DM_WRITE - Write
sNodeAddr / STRING(255)	Topology address '/' - Access to parameters of the L-force controller
wCode / WORD	Code of the parameter (Lenze) In this case, wIndex must have been set to 0
wIndex / WORD	Index of the parameter (CANopen) In this case, wCode must have been set to 0
bySubIndex / BYTE	Subindex/subcode of the parameter
eDataType / L_DMTVT	Indication of the type of parameter to be written {L_DM_VT_EMPTY, LDM_VT_I4, ...}

Designator / Data type	Info / Setting options
sDataW / STRING(255)	Value to be written (string) - Table "Relationship between the inputs and outputs and the data type"
byDataW / BYTE	Value to be written (1*byte) - Table "Relationship between the inputs and outputs and the data type"
wDataW / WORD	Value to be written (2*byte) - Table "Relationship between the inputs and outputs and the data type"
dwDataW / WORD	Value to be written (4*byte) - Table "Relationship between the inputs and outputs and the data type"

Outputs

Designator / Data type	Value / Meaning
bDone / BOOL	TRUE if operation concluded successfully
bBusy / BOOL	TRUE as long as the operation is being executed free of errors
bError / BOOL	TRUE if an error has occurred
dwErrorID / DWORD	Greater than 0 if an error has occurred
eDataRType / L_DMVT	Returned type of parameter that has been read {L_DM_VT_EMPTY, LDM_VT_I4, ...}
sDataR / STRING(255)	Value that has been read (string) - Table "Relationship between the inputs and outputs and the data type"
byDataR / BYTE	Value that has been read (1*byte) - Table "Relationship between the inputs and outputs and the data type"
wDataR / WORD	Value that has been read (2*byte) - Table "Relationship between the inputs and outputs and the data type"
dwDataR / DWORD	Value that has been read (4*byte) - Table "Relationship between the inputs and outputs and the data type"

Relationship between the inputs and outputs and the data type

Input, output	Data type
sDataR, sDataW	L_DM_VT_BSTR
byDataR, byDataW	L_DM_VT_I1, L_DM_VT_UI1
wDataR, wDataW	L_DM_VT_I2, L_DM_VT_UI2
dwDataR, dwDataW	L_DM_VT_I4, L_DM_VT_UI4, L_DM_VT_DATE

Error messages from the function block

Error number	Meaning
507	Incorrect format indicated during an instance of write access
531	Incorrect format indicated during assignment
532	Code contains float/double parameters (e.g. supply voltage C00205 or rechargeable-battery voltage C00207); This format is not supported by the block.
Other	Internal software error; Please contact your Lenze Service Centre.

2.1.3 Library: L_SM3_DriveUtil.lib

Function block L_SMC_SetTorqueLimit (for EtherCAT)

Function library: L_SM3_DriveUtil.lib

The torque limits of a drive in the positive and negative direction of rotation can be prescribed by the control system. Previously, the limits were set by means of SDO access. If the drives are controlled via an EtherCAT fieldbus, it is possible to transfer these values as part of the cyclical process data. The advantage of this is considerably faster transfer of the values to the drive.In order to use this, a corresponding mapping that contains these torque limits as an item of process data must be selected in the EtherCAT Configurator.

Access to the torque limits from the PLC takes place by means of the function block L_SMC_SetTorqueLimit:

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Execute / BOOL	Activates the function.
SetTorqueLimitPos / BOOL	Requests a positive torque limit
TorqueLimitPos_Value / LREAL	Positive torque limit in the range 0 ... 100 % of the rated torque
SetTorqueLimitNeg / BOOL	Requests a negative torque limit
TorqueLimitNeg_Value / LREAL	Negative torque limit in the range 0 ... 100 % of the rated torque
Axis / AXIS_REF	Here, the axis structure to which these torque limits apply is transferred.

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	TRUE indicates that the entry has been acknowledge.
Error / BOOL	TRUE indicates that an error has occurred in the function block.
ErrorID / INT	Error number

Error messages

The following error messages have been stored:

Error message	Description
ERROR_NO_LENZE_AXIS_REF: INT := 16#200	The function is not supported by this drive.
ERROR_TORQUE_LIMIT_NOT_VALID: INT := 16#210	Value range exceeded. Only torque limits of 0 ... 200 % are permissible.
ERROR_TORQUE_LIMIT_MAPPING_FAULT := 16#211	This function cannot be executed with the selected mapping. The torque process data are absent.

Example 1: The PLC wants to set the torque limit positively to 50 %.

SetTorqueLimitPos = TRUE
TorqueLimitPos_Value: 50.0
SetTorqueLimitNeg = FALSE
TorqueLimitNeg_Value = xxx
Execute = True

The drive sets the positive torque limit to 50 % of the rated torque; the negative torque limit valid in the drive is not altered.

Example 2: The PLC wants to set both torque limits to 100 %.

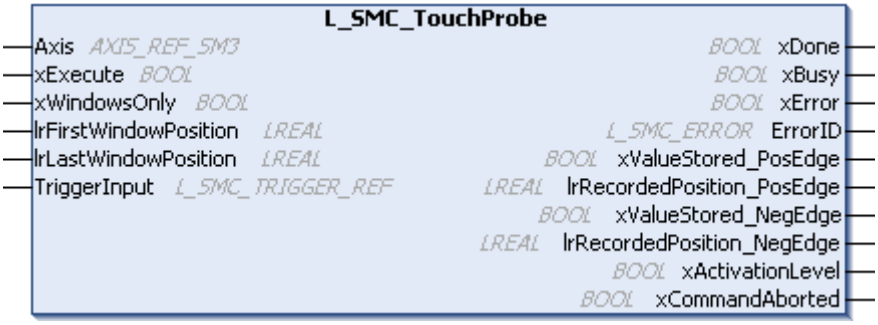
SetTorqueLimitPos = TRUE
TorqueLimitPos_Value: 100.0
SetTorqueLimitNeg = TRUE
TorqueLimitNeg_Value = 100.0
Execute = True

The drive sets both torque limits to 100 % of the rated torque.

Function block L_SMC_TouchProbe (for EtherCAT)

Function library: L_SM3_DriveUtil.lib

This purpose of this block is to detect the position of the drive with a high degree of accuracy via a fast input. Previously, the assignment and feedback of measured values took place by means of SDO access. If the drive is controlled via an EtherCAT fieldbus, it is possible to transfer these data as part of the cyclical process data. The advantage is substantially faster transfer of the data from and to the drive. In order to be able to use this, a corresponding mapping that contains these data as an item of process data must be selected.



Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Execute / BOOL	When a new positive edge is set, a new measuring is started. All outputs of the block are reset to the initial values.
WindowsOnly / BOOL	Activation of window evaluation: All measured values outside the defined window are suppressed.
FirstWindowsPosition / LREAL	Lower limit of the window in SoftMotion units [SM units]
LastWindowPosition / LREAL	Upper limit of window in SoftMotion units [SM units]
TriggerInput / L_SMC_Trigger_REF	See following description
Axis / AXIS_REF	Selection of the drive

TriggerInput / L_SMC_Trigger_REF:

Variable	Type	Initial value	Description
TouchProbeSelect	Byte	1	Selection of TP1 or TP2 1: TP_Channel1 -> 9400; DigIn 7 or zero pulse of position encoder 2: TP_Channel2 -> 9400; DigIn 8 or zero pulse of position encoder
TouchProbeMode	Byte	1	Selection of operating mode 1: Single measurement; Execute edge activates the function until Done = 1 (TRUE) or abortion by means of FB L_SMC_AbortTrigger. 2: Continuous measurement; Positive execute edge activates the function until negative execute edge (Execute = FALSE) or abortion by means of FB L_SMC_AbortTrigger.
TouchProbeSignal	Byte	1	Selection of zero pulse or digital input 1: Digital input 2: Zero pulse of position encoder
TouchProbeEdge	Byte	1	Configuration of edge evaluation 1: Triggering with positive edge 2: Triggering with negative edge 3: Triggering with positive or negative edge (logic in the FB) 4: Without function. 5: Triggering with positive and negative edge (The sequence of the edges is relevant (logic in the drive).) 6: Triggering with negative and positive edge (The sequence of the edges is relevant (logic in the drive))

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	In the "Single measurement" mode: TRUE = Measuring procedure completed. In the "Continuous measurement" mode: Without any function (see input variables from TriggerInput / L_SMC_Trigger_REF "TouchProbeMode")
Busy / BOOL	Block activated, waiting for trigger.
Error / BOOL	Error detection: An error has occurred in task assignment or in communication with the drive.
ErrorID / INT	Error number
ValueStored_PosEdge / BOOL	Measured value saved after triggering on positive flank.
RecordedPosition_PosEdge / LREAL	Measured value in the case of triggering with the positive edge in SoftMotion units [SM units], i.e. the measured value is supplied to the PLC application in the same unit in which the axis is also programmed.
ValueStored_NegEdge / BOOL	Measured value saved after triggering on negative edge.
RecordedPosition_NegEdge / LREAL	Measured value in the case of triggering with the negative edge in SoftMotion units [SM units], i.e. the measured value is supplied to the PLC application in the same unit in which the axis is programmed.
ActivationLevel / BOOL	Level of the TP input at the time of TP function activation in the drive. The application can detect the nature of the first active edge by evaluating the activation level when a "positive or negative edge" is assigned.
CommandAborted / BOOL	Block aborted by means of L_SMCAbortTrigger. The outputs "Busy" and "Done" are reset.

Error messages

The following error messages have been stored:

Error number	Description
411	TouchProbeSelect incorrect: TP channel does not exist TouchProbeSelect > 1 in the case of ECSxM TouchProbeSelect > 2 in the case of 9400
412	TouchProbeMode incorrect: TouchProbeMode > 1 in the case of ECSxM (Only 'single measurement' is supported.) TouchProbeMode > 2 in the case of 9400
413	TouchProbeEdge incorrect: When "TouchProbeSignal = Zero pulse" is selected, only evaluation of 'positive edge' is possible TouchProbeMode > 1 in the case of ECSxM (Only 'single measurement' is supported.) TouchProbeMode > 2 in the case of 9400
414	TouchProbeEdge incorrect: Requested edge evaluation does not exist TouchProbeEdge > 2 in the case of ECSxM (Only positive or negative edge possible.) TouchProbeEdge > 4 in the case of 9400 (Type of edge evaluated not supported.)
415	TP channel not free: TP channel has already been activated by means of other block.
416	Touch Probe is not possible for the currently selected process data mapping.
417	Internal error / TP channel not free: TP channel has already been activated by means of other block.
418	Internal error: Timeout during assignment of the drive

Function block L_SMC_AbortTrigger (for EtherCAT)

Function library: L_SM3_DriveUtil.lib

Note: This block only supports EtherCAT axes at the present time.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Execute / BOOL	When a new positive edge is set, all outputs of the block are reset to the initial values.
TriggerInput / L_SMC_Trigger_REF	Structure for configuration of the measurement
Axis / AXIS_REF	Selection of the drive

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	Function block L_SMC_TouchProbe has been aborted.
Error / BOOL	Error detection: An error has occurred in task assignment.
ErrorID / INT	Error number

Error messages

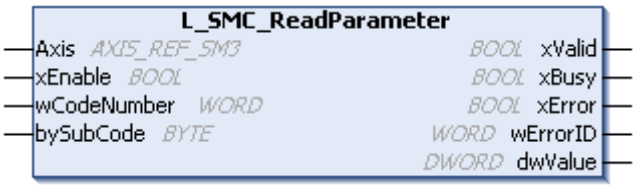
The following error messages have been stored:

Error number [hex]	Description
0x0001	Error in assignment of task to the block: Input data do not match activated TP task assignment on this channel. If no task is active on the TP channel, an error message is not generated. TouchProbeSelect > 1 in the case of ECSxM TouchProbeSelect > 2 in the case of 9400

Function block L_SMC_ReadParameter (for EtherCAT)

Function library: L_SM3_DriveUtil.lib

With this block, codes of a Motion drive are read out. The code and the subindex are indicated directly at the block. Conversion to the corresponding parameter numbers takes place internally.



Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Enable / BOOL	Activates the function.
wCode / WORD	Indication of the code that is to be read.
bySubIndex / BYTE	Indication of the subindex that is to be real.
Axis / AXIS_REF	Here, the axis structure for which the code is to be read is transferred.

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	TRUE indicates that the entry has been acknowledge.
Error / BOOL	TRUE indicates that an error has occurred in the function block
ErrorID / INT	Error number
dwValue / DWORD	Value which has been read

Error messages

The following error messages have been stored:

Error number [hex]	Description
SMC_RP_PARAM_UNKNOWN:=130	Unknown code
SMC_RP_REQUESTING_ERROR:=131	Read request has been aborted, e.g. because the drive did not respond in the prescribed time.

Function block L_SMC_WriteParameter (for EtherCAT)

Function library: L_SM3_DriveUtil.lib

With this block, codes of a Motion drive are written. The code and subindex are indicated directly at the block. Conversion to the corresponding parameter numbers takes place internally.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Execute / BOOL	Activates the function.
wCode / WORD	Indication of the code into which value to be written.
bySubIndex / BYTE	Indication of the subindex into which value is to be written.
dwValue / DWORD	Value that is to be written.
Axis / AXIS_REF	Here, the axis structure for which the code is to be written is transferred.

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	TRUE indicates that the entry has been acknowledge.
Error / BOOL	TRUE indicates that an error has occurred in the function block
ErrorID / INT	Error number

Error messages

The following error messages have been stored:

Error number [hex]	Description
SMC_WP_PARAM_UNKNOWN:=140	Unknown code
SMC_WP_SENDING_ERROR:=141	Write request has been aborted, e.g. because the drive did not respond in the prescribed time.

Function block L_SMC_ReadAxisError

Function library: L_SM3_DriveUtil.lib

This block serves to read the error code of a motion drive.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Axis / AXIS_REF	Here, the axis structure for which the code is to be read is transferred.
xEnable / BOOL	Activates the function.

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
xValid / BOOL	Request has been executed
xBusy / BOOL	Request has been started
xError / BOOL	Request executed with error
wErrorID / WORD	Error number for this command to be executed, not the error number of the drive
xAxisError / BOOL	Drive has signalled an error
dwAxisErrorID / DWORD	Read error number of the drive

Error messages

The following error messages have been stored:

Error number [hex]	Description
SMC_RP_PARAM_UNKNOWN:=130	Unknown code
SMC_RP_REQUESTING_ERROR:=131	Read request has been aborted, e.g. because the drive did not respond in the prescribed time.

For the meaning of the drive error numbers, please refer to the drive documentation.

2.1.4 Library: L_SM3_DriveUtil.lib (I/O-System 1000)

Basic functions of fast inputs/outputs

The I/O compound modules described below belong to the I/O system 1000. The system is suitable for the implementation of complex automation applications and consists of a bus coupler module and several I/O compound modules. The in-station communication of these modules as well as the exchange of process data, parameter data and diagnostic information takes place via an internal backplane bus.

2 digital inputs with time stamp function - EPM–S207

This "fast input" detects up to two binary control signals from the process level and transmits these signals to the higher-level bus system. An internal 16-bit counter (µs–ticker) can be used to record the occurrence time (latch value) of a certain event (time stamp functionality).

Fast inputs are used for:

- Alignment of discontinuous motion to a registration mark
  Flying saw / rotary knife / feed
- Alignment of continuous motion to a registration mark
  Rotary processing (stamping, embossing, printing)
- Measurement of workpieces
- Position latch on virtual or real axes

Features:

- 2 digital inputs
- Up to 15 time stamp entries can be recorded
- Suitable for switches and proximity switches

2 digital outputs with time stamp function - EPM–S310

This "fast output" detects up to two binary control signals from the higher-level bus system and transmits these signals to the process level using the time stamp functionality. Here the setting of a certain output is based on the internal 16-bit counter (µs-ticker).

Fast outputs are used for:

- Cam groups whose outputs need to be set very precisely depending on the position.

Features:

- 2 digital outputs
- FIFO buffer for up to 15 time stamp entries
- Control via process image

Note: Both I/O compound modules are provided with own 16-bit counters (µs-ticker) which must be synchronised by the RWB controller.

Function block L_SMC_TouchProbe_FastIO

Function library: L_SM3_DriveUtil.lib

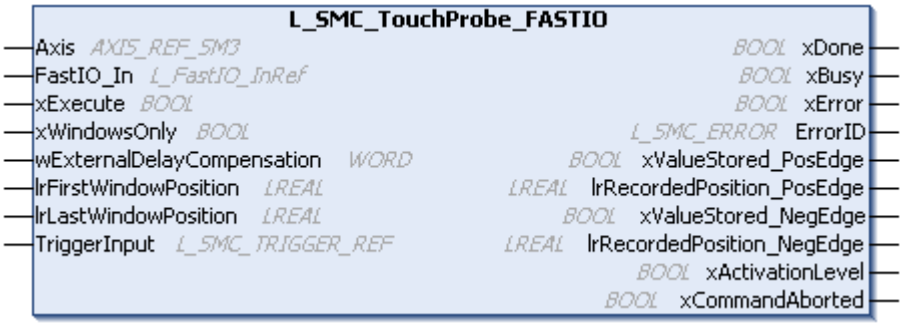
A fast input of the EPM-S207 I/O compound module provides a time stamp entry (latch of the internal µs−ticker), which is converted into a position in this function block.

Note: This block is similar to the already existing touch probe block for Servo Drives 9400 and ECS devices. Therefore, a detailed description is not given here.

Please observe:

- The function block L_SMC_TouchProbe_FastIO does not compensate the propagation times of the axis positions on the respective fieldbus. The compensation can, however, be specified manually via the input wExternalDelayKompensation.
- Encoders connected to a counter I/O compound module (EMP-S600 to EPM-S604) are compensated automatically. The counter modules are also provided with the time stamp functionality. The time stamps are transferred to the control together with the position values.
- For very precise touch probe measurement results, we recommend to implement the position measurement via an encoder connected to a counter I/O compound module.
- How to connect a counter I/O compound module as a SoftMotion axis is explained in the description for the function block L_SMC_FreeEncoder.

Note: If several I/O compound modules with time stamp functionality (µs−ticker) are used, they must be synchronised by the RWB controller.



Inputs/outputs (VAR_IN_OUT)

Designator / Data type	Info / Setting options
Axis	Reference to the FreeEncoder axis; here any SoftMotion axis can be connected.
FastIO_InRef	I/O compound module reference; this block is automatically added to the device tree as a subnode and instantiated when, for example, the EPM-S207 module is inserted in the device tree. The instance of this block must be transferred to the touch probe block.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
xExecute	When a new positive edge is set, a new measurement is started. All outputs of the block are reset to the initial values.
xWindowsOnly	Activation of window evaluation: All measured values outside the defined window are suppressed.
wExternalDelayCompensation	Manual compensation for the propagation times of the axis positions on the respective fieldbus. The delay time is entered in [µs].
lrFirstWindowPosition	Lower limit of the window in SoftMotion units [SM units]
lrLastWindowPosition	Upper limit of window in SoftMotion units [SM units]
TriggerInput	See below description: L_SMC_Trigger_REF

TriggerInput / L_SMC_Trigger_REF:

Variable	Type	Initial value	Description
TouchProbeSelect	BYTE	1	Selection of TP1 or TP2 1: TP_Channel1 -> DigIn 1 2: TP_Channel2 -> DigIn 2
TouchProbeMode	BYTE	1	Selection of operating mode 1: Single measurement; Execute edge activates the function until Done = 1 (TRUE) or abort by means of FB L_SMC_AbortTrigger. 2: Continuous measurement; Positive execute edge activates the function until negative execute edge (Execute = FALSE) or abort by means of FB L_SMC_AbortTrigger.
TouchProbeSignal	BYTE	1	Selection of zero pulse or digital input 1: Digital input
TouchProbeEdge	BYTE	1	Configuration of edge evaluation 1: Triggering with positive edge 2: Triggering with negative edge 3: Triggering with positive or negative edge (logic in the FB)

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
xDone	This output is set to TRUE when the parameter has been written correctly.
xBusy	This output is set to TRUE as long as the writing has not been completed.
xError	This output is set to TRUE if an error occurs during writing. eError outputs the cause of the error.
ErrorID	Error number
xValueStored_PosEdge	Measured value saved after triggering on positive edge
lrRecordedPosition_PosEdge	Measured value in the case of triggering with the positive edge in SoftMotion units [SM units], i.e. the measured value is supplied to the PLC application in the same unit in which the axis is programmed.
xValueStored_NewEdge	Measured value saved after triggering on negative edge.
lrRecordedPosition_NegEdge	Measured value in the case of triggering with the negative edge in SoftMotion units [SM units], i.e. the measured value is supplied to the PLC application in the same unit in which the axis is programmed.
xActivationLevel	Level of the TP input at the time of TP function activation in the drive. The application can detect the nature of the first active edge by evaluating the activation level when a "positive or negative edge" is assigned.
xCommandAborted	Block aborted by means of L_SMCAbortTrigger. The outputs "Busy" and "Done" are reset.

Error messages

The following error messages have been stored:

Error message	Description
411	TouchProbeSelect incorrect: TP channel does not exist TouchProbeSelect > 1 in the case of ECSxM TouchProbeSelect > 2 in the case of 9400
412	TouchProbeMode incorrect: Only the values 1 or 2 are permitted.
413	TouchProbeEdge incorrect: Only the values 1, 2 or 3 are permitted.
414	TouchProbeEdge incorrect: Requested edge evaluation does not exist TouchProbeEdge > 2 in the case of ECSxM (Only positive or negative edge possible.) TouchProbeEdge > 4 in the case of 9400 (Type of edge evaluated not supported.)
415	TP channel not free: TP channel has already been activated by means of other block.
416	Touch probe is not possible for the currently selected process data mapping.
417	Internal error / TP channel not free: TP channel has already been activated by means of other block.
418	Internal error: Timeout during assignment of the drive
419	Internal error: Communication error
421	TouchProbeSignal incorrect: Only the value 1 is permitted.
430	The FreeEncoder axis specified at the Axis input does not exist
431	The FreeEncoder axis specified at the Axis input is configured incorrectly
432	TouchProbeEdge selection incorrect
433	TouchProbeEdge selection incorrect

Function block L_SMC_AbortTrigger_FastIO

Function library: L_SM3_DriveUtil.lib

This block serves to abort a touch probe cycle running on the L_SMC_TouchProbe_FastIO block.

Inputs/outputs (VAR_IN_OUT)

Designator / Data type	Info / Setting options
Axis / AXIS_REF	Selection of the drive
FastIO_In / FastIO_InRef	I/O compound module reference; this block is automatically added to the device tree as a subnode and instantiated when, for example, the EPM-S207 module is inserted in the device tree. The instance of this block must be transferred to the touch probe block.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
Execute / BOOL	Abort of the running touch probe cycle. When a new positive edge is set, all outputs of the block are reset to the initial values.
TriggerInput / L_SMC_Trigger_REF	Structure for configuration of the measurement
Axis / AXIS_REF	Selection of the drive

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
Done / BOOL	Function block L_SMC_TouchProbe has been aborted.
Error / BOOL	Error detection: An error has occurred in task assignment.
ErrorID / INT	Error number

Error messages

The following error messages have been stored:

Error number [hex]	Description
0x0001	Error in assignment of task to the block: Input data do not match activated TP task assignment on this channel. If no task is active on the TP channel, an error message is not generated.

Function block L_SMC_FastOutput

Function library: L_SM3_DriveUtil.lib

This function block calculates the time (latch of the internal µs−ticker) for setting the outputs of the EPM-S310 I/O compound module from a position defined by the fSwitchPosition input. Since the position is transferred in SoftMotion units, the reference transferred is an AxisRef.

Please observe:

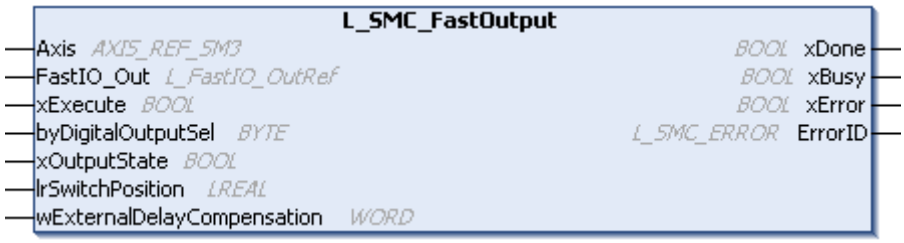
- The function block L_SMC_FastOutput does not compensate the propagation times of the axis positions on the respective fieldbus. The compensation can, however, be specified manually via the input wExternalDelayKompensationD0/D1.
- Encoders connected to a counter I/O compound module (EMP-S600 to EPM-S604) are compensated automatically. The counter modules are also provided with the time stamp functionality. The time stamps are transferred to the control together with the position values.
- To ensure that the output is set very precisely depending on the position, we recommend to implement the position measurement via an encoder connected to a counter I/O compound module.
- How to connect a counter I/O compound module as a SoftMotion axis is explained in the description for the function block L_SMC_FreeEncoder.

Please observe: If several switching operations are to be carried out in succession, the respective commands must have the same order as the switching operations.

Example: FreeEncoder axis rotates in positive direction, i.e. the position values increase.

1. Switching operation at 60 degrees
2. Switching operation at 90 degrees
3. Switching operation at 120 degrees

The order of the commands to be executed must be: "1st switching operation", "2nd switching operation", and "3rd switching operation".



Inputs/outputs (VAR_IN_OUT)

Designator / Data type	Info / Setting options
Axis	Reference to the axis. Only one L_SMC_FreeEncoder axis is supported.
FastIO_Out	Reference to the EPM S310 I/O compound module.

Inputs (VAR_IN)

Designator / Data type	Info / Setting options
xExecute	When a new positive edge is set, a new measurement is started. All outputs of the block are reset to the initial values.
byDigitalOutputSel	Selection of the output.
xOutputState	Value to which the output is to be written.
lrSwitchPosition	Position value at which the output is to be set. Since the position is transferred in SoftMotion units, the reference transferred is an AxisRef.
wExternalDelayCompensation	The propagation times of the axis positions on the respective fieldbus are not compensated automatically. This input can be used for setting the manual compensation for the D0 output. The delay time is entered in [µs].

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
xDone	This output is set to TRUE when the parameter has been written correctly.
xBusy	This output is set to TRUE as long as the writing has not been completed.
xError	This output is set to TRUE if an error occurs during writing. eError outputs the cause of the error.
ErrorID	Error number

Error messages

The following error messages have been stored:

Error message	Description
431	The configuration of the axis specified at the "Axis" input is faulty. Check the resolution set for the encoder.
416	System error. Please contact the Lenze Support.
430	The axis specified at the "Axis" input does not exist.
440	The output cannot be set for the set position due to the propagation times in the system.
441	The output cannot be set for the set position because this position will only be reached after more than 65 ms.
442	Faulty selection of the output. Only the outputs 1 and 2 are available.
443	Function cannot be executed because the speed of the FreeEncoder axis is too low.
444	FIFO overflow for EPM-S310 I/O compound module.

Function block L_SMC_FreeEncoder

Function library: L_SM3_DriveUtil.lib

The function block L_SMC_TouchProbe_FastIO must be connected to an AxisRef block providing the position and the speed value of an encoder. These data are used to determine the touch probe position.

In analogy to the "SMC_FreeEncoder device" with a lower-level AxisRef node, an own SoftMotion node can be added manually to the existing counter and SSI-I/O modules: L_SMC_FreeEncoder.

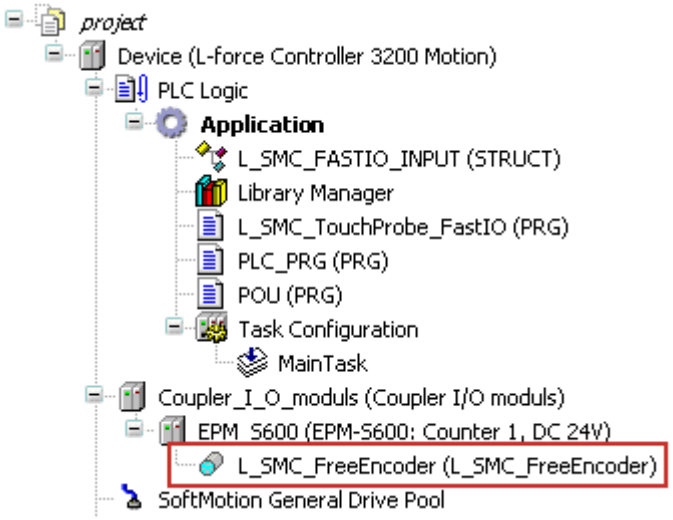


Abb. 1 Figure: Counter I/O module with AxisRef node

In addition to the position, speed and acceleration process values, this AxisRef block also provides the raw data of the counter or SSI-I/O module (i.e. the "timer tick" for the read position and the absolute position).

The position of AxisRef is compensated so that it refers to the "timer 0 interrupt" of the control, i.e. the "timer 0 interrupt" is the reference point.

The "L_SMC_FreeEncoder device" also includes a special tab that is similar to the tab of the SM_Drive_Logical. This tab can be used to set whether a linear or a rotary measuring system is to be used and to select the filtering of position and speed.

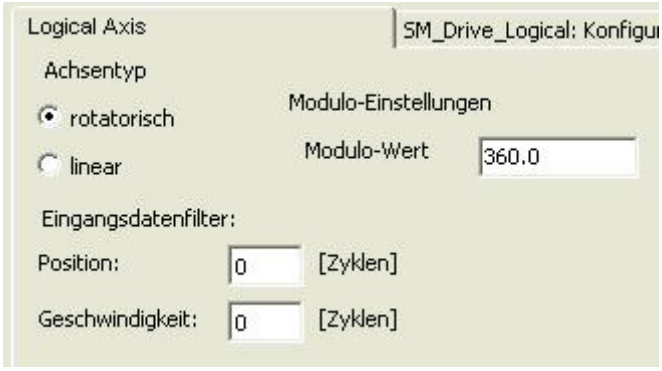
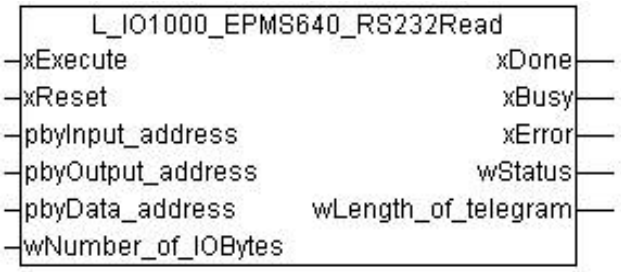


Abb. 2 Figure: Tab for L_SM_FreeEncoder

Function block L_IO1000_EPMS640_RS232Read

Function library: L_SM3_DriveUtil.lib

This function block coordinates the data reception of the EPM-S640 I/O compound module via RS232.



Inputs (VAR_IN)

Designator / Data type	Info / Setting options
xExecute / BOOL	A positive edge at the input triggers a request, i.e. if the EPM-S64l I/O compound module has received data, these are read. A reset of the input during processing does not affect the request. After executing the current request, the block is reset with xExecute =FALS“ and xReset=FALS“ and prepared for a new reset / data request.
xReset / BOOL	A positive edge at the input triggers a reset of the EPM-S640 I/O compound module which reinitialises the module. A reset of the input during processing does not affect the reset. After executing the current request, the block is reset with xReset=FALSE and xExecute=FALSE and prepared for a new reset / data request. Notes: Prior to the first data reception, the user has to execute a reset! This reset is a synchronous reset, i.e. when executed on the read block, the EPM-S640 I/O compound module will be reinitialised both for data reception and for data transmission. If a reset and a data request are triggered at the same time, the reset (having the higher priority) will be executed. A reset can also be executed during an active request. In this case, it assumes the function of a request abort.
pbyInput_address / POINTER_TO_BYTE	Start address of the of EPM-S640 I/O compound module's input area in the process image. The address must be transferred with the ADR operator. Example :"... pbyInput_address:=ADR(%IB4800) , ..."
pbyOutput_address / POINTER_TO_BYTE	Start address of the EPM-S640 I/O compound module's output area in the process image. The address must be transferred with the ADR operator. Example: "... pbyOutput_address:=ADR(%QB4800) , ..."
pbyData_address / POINTER_TO_BYTE	Start address of the data area for received data. The address must be transferred with the ADR operator. Example: "... pbyData_address:=ADR(%MB1024) , ..."
wNumber_of_IOBytes / WORD	Number of bytes available in the process image. The valid values depend on the bus system used and the module configuration: CAN: 8 bytes Profibus: 8, 16, 32, 60 bytes EtherCat: 60 bytes

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
xDone / BOOL	The bit is set if a receive request or a reset has been executed correctly.
xBusy / BOOL	The bit is set as long as a request (receive/reset) is being processed. After the processing has been completed or if an error occurs, the bit is reset and xDone or xError is set.
xError / BOOL	The bit is set if an error occurs. "wStatus" provides detailed information about the error.
wStatus / WORD	No error (xError not set): 0x0000: no request being processed / request executed correctly 0x8181: transmit request being processed Error ("xError" set): 0x0202: invalid value for wNumber_of_IOBytes 0x0301: no addresses transferred (pbyInput_address, pbyOut-put_address, pbyData_address) 0x0407: error during reception
wLength_of_telegram / WORD	Length of the last received telegram. The RS232 module can receive telegrams with lengths between 1 and 1024 bytes.

Note: Pointers to wrong addresses or to data areas with insufficient length can cause program errors or even a crash of the PLC!

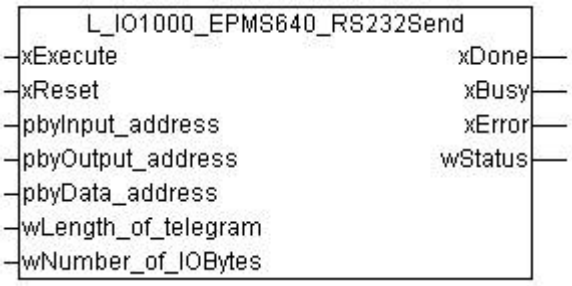
Example

```
FUNCTION_BLOCK L_IO1000_EPMS640_RS232Read
VAR_INPUT
    xExecute: BOOL; (*Trigger request on positive edge / enable reading of data from RS232 module*)
    xReset: BOOL; (*Trigger reset / reset block*)
    pbyInput_address: POINTER TO BYTE; (*Start address of input area in process image*)
    pbyOutput_address: POINTER TO BYTE; (*Start address of output area in process image*)
    pbyData_address: POINTER TO  BYTE; (*Start address of data area to be transferred*)
    wNumber_of_IOBytes: WORD; (*Number of bytes available in process image 8/ 20/ 60*)
END_VAR
VAR_OUTPUT
    xDone: BOOL; (*Request completed without error, data sent STATUS parameter = 00h*)
    xBusy: BOOL; (*Information on processing status*)
    xError: BOOL; (*Request completed with errors, STATUS parameter contains error information*)
    wStatus: WORD; (*Specification of the error if ERROR = 1 /information on processing status, request is being processed 0x8181h*)
    wLength_of_telegram: WORD;  (*Number of bytes to be transferred. 1 <= LEN <= 1024*)
END_VAR
```

Function block L_IO1000_EPMS640_RS232Write

Function library: L_SM3_DriveUtil.lib

This function block coordinates the data transmission of the EPM-S640 I/O compound module via RS232.



Inputs (VAR_IN)

Designator / Data type	Info / Setting options
xExecute / BOOL	A positive edge at the input triggers a request, i.e. the defined data are transmitted by the EPM-S640 I/O compound module. A reset of the input during processing does not affect the request. After executing the current request, the block is reset with xExecute =FALS“ and xReset=FALS“ and prepared for a new reset / data request.
xReset / BOOL	A positive edge at the input triggers a reset of the EPM-S640 I/O compound module which reinitialises the module. A reset of the input during processing does not affect the reset. After executing the current request, the block is reset with xReset=FALSE and xExecute=FALSE and prepared for a new reset / data request. Notes: Prior to the first data transmission, the user has to execute a reset! This reset is a synchronous reset, i.e. when executed on the write block, the EPM-S640 I/O compound module will be reinitialised both for data transmission and for data reception. If a reset and a data request are triggered at the same time, the reset (having the higher priority) will be executed. A reset can also be executed during an active request. In this case, it assumes the function of a request abort.
pbyInput_address / POINTER_TO_BYTE	Start address of the of EPM-S640 I/O compound module's input area in the process image. The address must be transferred with the ADR operator. Example :"... pbyInput_address:=ADR(%IB4800) , ..."
pbyOutput_address / POINTER_TO_BYTE	Start address of the EPM-S640 I/O compound module's output area in the process image. The address must be transferred with the ADR operator. Example: "... pbyOutput_address:=ADR(%QB4800) , ..."
pbyData_address / POINTER_TO_BYTE	Start address of the data area for data to be transmitted.

Designator / Data type	Info / Setting options
	The address must be transferred with the ADR operator. Example: "... pbyData_address:=ADR(%MB0) , ..."
wLength_of_telegram / WORD	Number of bytes to be transmitted starting from the start address "pbyData_address". The valid length is between 1 and 1024 bytes.
wNumber_of_IOBytes / WORD	Number of bytes available in the process image. The valid values depend on the bus system used and the module configuration: CAN: 8 bytes Profibus: 8, 16, 32, 60 bytes EtherCat: 60 bytes

Outputs (VAR_OUT)

Designator / Data type	Value / Meaning
xDone / BOOL	The bit is set if a transmit request or a reset has been executed correctly.
xBusy / BOOL	The bit is set as long as a request (transmit/reset) is being processed. After the processing has been completed or if an error occurs, the bit is reset and xDone or xError is set.
xError / BOOL	The bit is set if an error occurs. "wStatus" provides detailed information about the error.
wStatus / WORD	No error (xError not set): 0x0000: no request being processed / request executed correctly 0x8181: transmit request being processed Error ("xError" set): 0x0202: invalid value for wNumber_of_IOBytes 0x0517: invalid value for wLength_of_telegram 0x070A: transmission failed 0x0301: no addresses transferred (pbyInput_address, pbyOutput_address, pbyData_address)

Note: Pointers to wrong addresses or to data areas with insufficient length can cause program errors or even a crash of the PLC!

Example

```
FUNCTION_BLOCK L_IO1000_EPMS640_RS232Send
VAR_INPUT
    xExecute: BOOL; (*Trigger request on positive edge / enable transmission of data to RS232 module*)
    xReset: BOOL; (*Trigger reset / reset block*)
    pbyInput_address: POINTER TO BYTE; (*Start address of input area in process image*)
    pbyOutput_address: POINTER TO BYTE; (*Start address of output area in process image*)
    pbyData_address: POINTER TO  BYTE; (*Start address of data area to be transferred*)
    wLength_of_telegram: WORD;  (*Number of bytes to be transferred. 1 <= LEN <= 1024*)
    wNumber_of_IOBytes: WORD; (*Number of bytes available in process image 8/ 20/ 60*)
END_VAR
VAR_OUTPUT
    xDone: BOOL; (*Request completed without errors, data sent STATUS parameter = 00h*)
    xBusy: BOOL; (*Information on processing status*)
    xError: BOOL; (*Request completed with error, STATUS parameter contains error information*)
    wStatus: WORD; (*Specification of the error if ERROR = 1 /information on processing status, request is being processed 0x8181h*)
END_VAR
```

2.2 CANopen / PROFIBUS / EtherCAT

The "CANopen control technology" and "EtherCAT control technology" communication manuals provide detailed information about commissioning and diagnostics of "logic" and "motion" field devices.

Note: PROFIBUS can only be used as a "logic bus" in conjunction with the Lenze control technology.

Special case of "cross communication"

Cross communication is only possible with logic devices. For this purpose, the required PDO mapping must be created in »PLC Designer«. When the PLC is started, the complete configuration/PDO mapping is written into the devices. The entries from »Engineer« are overwritten when this is done.

Note: The PDO mapping of ECSxM axis modules is static and cannot be changed.

2.3 PLC program for documentation of a cycle time violation

In the logbook of the L-force controller, you can document a cycle time violation by means of the code described below.

When you are specifying the required cycle time of a PLC project, please follow the procedures described in the communication manuals "Control Technology CANopen" and "Control Technology EtherCAT" (section: "Specifying the minimum cycle time of the PLC project").

Function:

- Information about a task can be requested from the "CmplecTask" system library by means of variables of the type "Task_Info".
- The function "L_AddLog" can be used to generate a log entry in the logbook of the L-force controller from the PLC.
- If the measured jitter is greater than the cycle time, a task cycle has failed and a corresponding log entry is created in the logbook of the L-force controller.

Precondition:

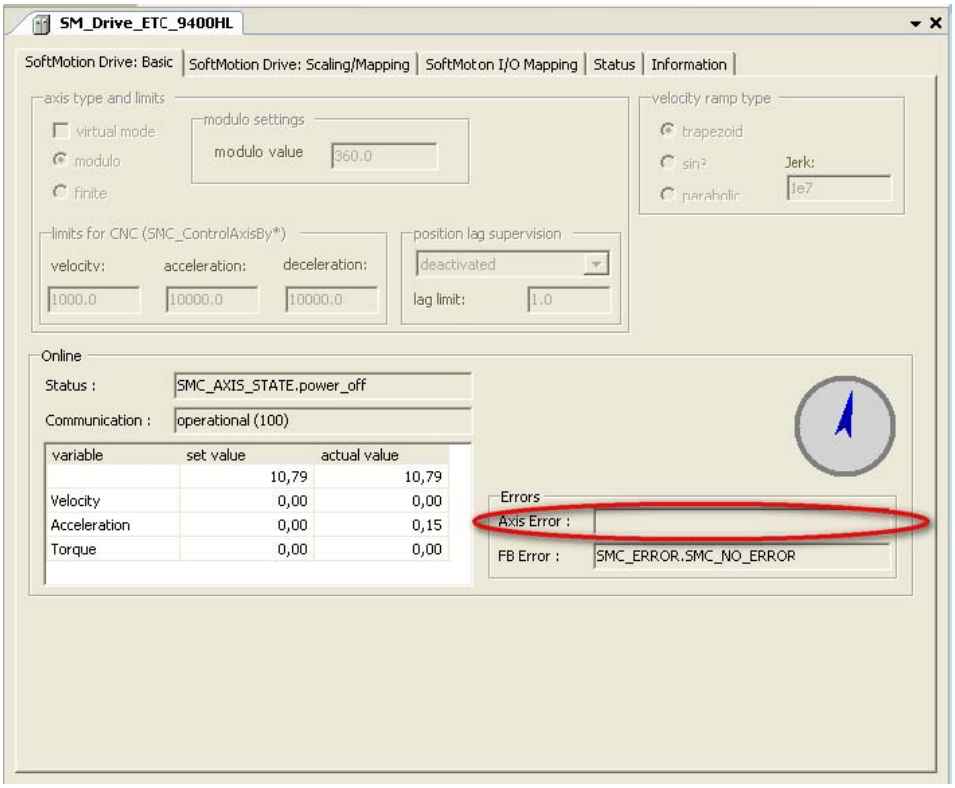
- The libraries "CmpIecTask" and "SysTypes" have been integrated in the PLC Designer project.
- The code indicated below must be called cyclically in the task to be examined.

Example of cycle time monitoring:

```
VAR
  pIecTaskResult : POINTER TO RTS_IEC_RESULT;
  tIecTaskHandle : RTS_IEC_HANDLE;
  pIecTaskInfo : POINTER TO Task_Info;
  tTaskInfo : Task_Info;
END_VAR
-------------------------------------------------------------------
tIecTaskHandle := IecTaskGetCurrent( pIecTaskResult );
pIecTaskInfo := IecTaskGetInfo2( tIecTaskHandle, pIecTaskResult );
tTaskInfo := pIecTaskInfo^;
IF ABS(tTaskInfo.iJitter) > TIME_TO_DINT(tTaskInfo.tInterval) THEN
  L_AddLog(eSeverity:= L_TSeverity.L_LogWarning,
    eFacility:= L_TFacility.L_PLC_USER_LOG_SCOPE1,
    sLogPayLoad:= 'Violation of cycle time PRG');
END_IF
```

2.4 Axis Error Messages on the SoftMotion register

In the online state of the »PLC Designer« potentially excisting axis errors are displayed. You will find this errors on the SoftMotion register "Basic":



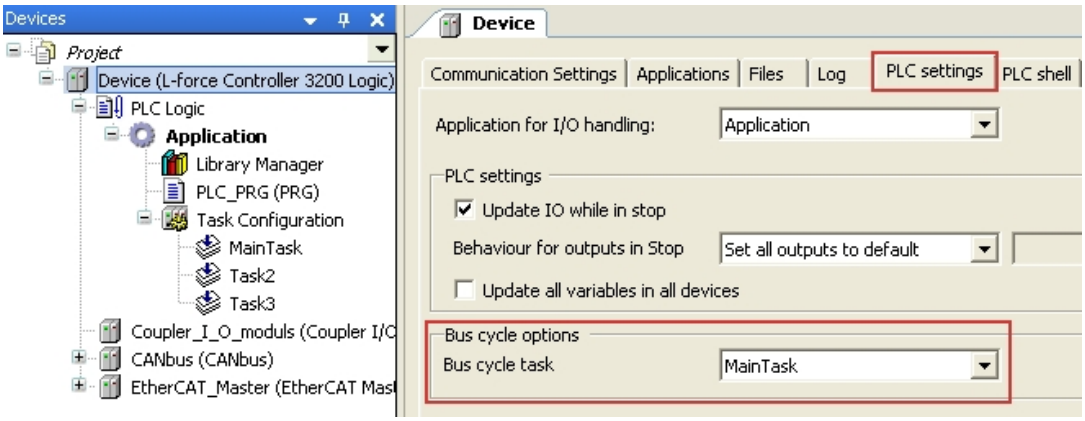
Axis Error No.	Description
2000	SDO access: Write timeout
2001	SDO access: Read timeout
2010	incorrect cycletime in the dedicated motiontask
2011	Incorrect ‚Maximum Torque' paramter in the drive
2020	Wrong application inthe drive

2.5 Assigning the transfer of a process image to a task

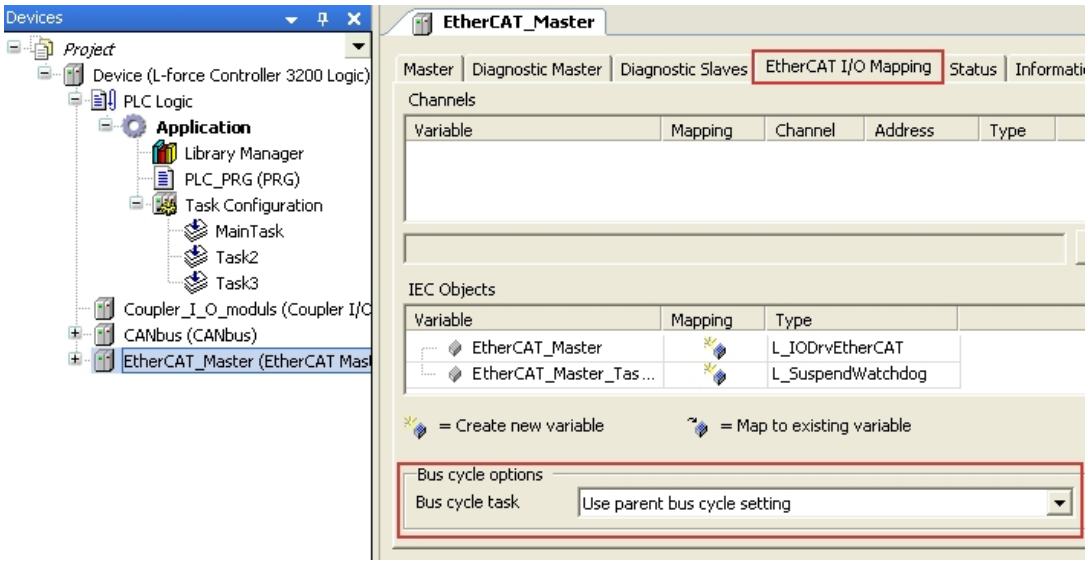
The transfer of the process image of a device can be assigned to a specific task.

For this purpose, the following input fields are available:

Control level



Bus master level (e.g. EtherCAT master)



If several tasks access the process image of a bus master within a PLC project, the fastest task always must be entered as the bus cycle task. The bus cycle tasks of the various bus systems (EtherCAT / CAN / PROFIBUS) can be set independently of each other.

2.6 Data consistency during access to process image

Within a PLC project, several tasks can access the process image of a bus master. Depending on the priority and cycle time configured, the tasks may interrupt one another during this process.

The system only ensures data consistency for the access to data types of the following length:

- Byte
- Word
- Dword

For bit accesses, data consistency is also ensured because here again uninterruptible (i.e. atomic) commands are used.

The system, however, does not provide data consistency for accesses to data structures (records). In this case, the user has to take appropriate measures.

Data consistency is achieved if only one task at a time accesses a certain data structure in the process image for writing.

3 Concepts and Base Components

3.1 Introduction and Basic Concepts

»PLC Designer« is a device-independent PLC-programming system. Matching the IEC 61131-3 standard it supports all standard programming languages but also allows including C-routines. In combination with the »PLC Designer« runtime system it allows to program multiple controller devices within one project.

Regard the following basic concepts determining programming with »PLC Designer«:

- **Object Orientation**: The mind of object orientation is not only reflected by the availability of appropriate programming elements and features but also in the structure and version handling of »PLC Designer« and in the project organization. **Multi-device** and **multi-application** usage of a »PLC Designer« project is possible basing on jointly used, instantiated programming units. So running various applications on the same device and cloning of applications is possible as well as mixing parameterizable and programmable controller devices in one project.
- **Component-based** structure of the programming system: The functionality available in the user interface (editors, menus etc.) depends on the currently used components (plug-ins) defined in a profile. There are system components, which are essential, and optional components. Additionally to the default 3S plug-ins, **customer specific plug-ins** can be created by means of the **Automation Platform** tool-kit.
- **Version handling**: A simultaneous installation of several versions of »PLC Designer« components and working with the desired combination of versions is possible. This also pertains the device-specific use of different compiler versions. Individual functions can be added without having to update the whole version.
- **Project organization** is also determined by the mind of object orientation:
  A »PLC Designer« project contains a PLC program composed of various programming objects and it contains definitions of the "resources" which are needed to run instances of the program (application) on defined target systems (devices, PLCs).
  So there are two main types of objects in a project:

  Programming objects (POUs):
  These are programs, functions, function blocks, methods, interfaces, actions, data type definitions etc.

  Programming objects which can be instantiated in the entire project, i.e. for all applications defined in the project, must be managed in the POUs window.

  The **instantiating** is done by calling a program POU by an application-assigned task.

  Programming objects which are managed (only!) in the devices window, i.e. which are directly assigned to an application, cannot only be instantiated by another application inserted BELOW.

Resource objects (devices tree):

These are device objects, applications, task configurations, recipe managers etc. and are ONLY managed in the Devices window, i.e. in the "device tree". When inserting objects in the devices tree, the hardware to be controlled must be mapped according to certain rules. The scope of objects like e.g. libraries, GVLs etc. thereby is defined by the hierarchical arrangement of application and device objects within the tree.

- **Code generation** by integrated compilers and use of machine code results in short execution times.
- **Data transfer** to the controller device: The data transfer between »PLC Designer« and the device is done via a **gateway** (component)) and a **runtime system**. Complete online functionality for controlling a program on the device is available.
- Predefined **feature sets** serve to be able to choose between a "standard" user interface with a reduced selection of features and less complexity and a "professional" environment supporting all features. When the programming system is initially started after the first installation on the system, you will be prompted to choose one of the sets. But also later you still can switch the set and also a user-defined customization of the currently used feature set is possible. For the particular differences between the standard and professional version please see the description of the Feature Options.

3.2 Differences to »PLC Designer« V2.3

Basically: Projects created with »PLC Designer« V2.3 can be opened and edited further on.

The following extensions and improvements are available with »PLC Designer« V3.x:

Object orientation in programming and project structure

- extensions to function blocks: Properties, Interfaces, Methods, Inheritance, Method invocation
- extensible functions - not yet implemented
- device bound applications are instances of independent programming objects

New concerning Datatypes

- new type: ANY_TYPE
- new type: UNION
- new type: LTIME
- new type: References
- enumerations: base data type can be specified
- di : DINT := DINT#16#FFFFFFFF; not allowed

New concerning operators and variables

- New scope operators, extended name spaces
- Function Pointers replacing the INSTANCE_OF operator
- Init method replacing the INI operator
- Exit method
- Output variables in function and method calls
- VAR_TEMP, VAR_STAT
- Arbitrary expressions for variable initialization
- Assignment as expression
- Index access with pointers and strings
- Extensible functions (variable number of parameters) - not yet implemented

New Visualization concept

- The Visualization Editor per default is working in combination with a Toolbox and a Visual Element Properties Editor.
- Parts of the visualization functionality are realized according to IEC 61131 and thus provided via libraries.
- An internal runtime system executes the main visualization functions.
- Visual elements are provided via libraries (Visual Element Repository).
- Textlists and image pools are used for managing texts and images.

New User Management and Access Rights concept

- User accounts, user groups, group-specific rights for access and actions on particular objects

New in editors

- ST-Editor: Folding, Linebreak, Autocomplete, Inline Monitoring, Inline set/reset-assignment
- FBD, LD and IL are reversible and programmable in a combined editor
- IL editor is a table editor
- LD/FBD/IL editor: main output in boxes with multiple outputs can be changed
- LD/FBD/IL editor: no automatic update of box parameters
- LD/FBD/IL editor: branches and "networks within networks"
- SFC editor: only one step type, macros, multiple selection of independent elements, no syntactic check during editing, automatic declaration of flag variables

Library versions

- Multiple versions can be used in the same project, unique access to library modules by specifying the namespace
- Installation in repositories, automatic update, debugging

and more....

- Configurable menus, toolbar and keyboard usage
- Customer-specific components might be plugged in

- PLC Configuration and Task Configuration integrated in devices tree
- UNICODE support
- Single-line comments
- Continuing in loops
- Multiple selection in the project object tree
- Online help integrated in the user interface
- Conditional compilation
- Conditional breakpoints
- Debugging: step to cursor, back to previous caller
- Field bus driver matching IEC 61131-3
- Symbolic and PLC configuration available in application
- Free memory allocation of code and data
- Each object can be specified to be "internal" or "external" (late link in the runtime system)
- Connecting external data sources (Data Server)
- Precompile notes concerning syntax errors

Compatibility concerning »PLC Designer« V2.3 projects and other file formats

- Projects of different formats, i.a. projects created with »PLC Designer« V2.3, can be opened if the appropriate converter is installed. See command 'Open Project'. The handling of included libraries and devices can be defined.
- Syntactical and semantical restrictions since »PLC Designer« V2.3:
  - FUNCTIONBLOCK is no longer a valid key word instead of FUNCTION_BLOCK.
  - TYPE (declaration of a structure) requires a following ":"
  - ARRAY initialisation requires squared brackets.
  - local declaration of an enumeration is no longer possible except within TYPE; at a V2.3 import a local enumeration declaration will be converted automatically to an explicit type definition.
  - INI is no longer supported (replaced by the Init Method)
  - In function calls, it is no longer possible to mix explicit parameter assignment with implicit, therefore it is possible to change the order of parameter input assignments:
    fun(formal1 := actual1, actual2); // -> error message
    fun(formal2 := actual2, formal1 := actual1); // same semantics as
    fun(formal1 := actual1, formal2 := actual2);
- Pragmas (»PLC Designer« V2.3 pragmas not yet implemented)
- The TRUNC operator now converts to data type DINT instead of INT; at a V2.3 import an appropriate type conversion will be added automatically.

3.3 Profiles and Plug-Ins

If you start "»PLC Designer« without Profile" from the Start menu, you first will be asked to select a "profile". (Starting directly "»PLC Designer« V3.x" from the start menu or via the desktop icon will automatically use the default profile).

A profile defines the set of currently available user interface components (**plug-in**s).

Each plug-in is defined by a name, a version and version constraints.

Besides the components provided by 3S - Smart Software Solutions GmbH also customer-defined plug-ins might be integrated.

Examples for essential components are the 'Online Manager' or the 'Runtime System'. Those must be part of the system anyway. Other components provided by 3S - Smart Software Solutions GmbH as plug-ins, like various editors, code generators, communication drivers etc., can be used optionally.

For the creation of plug-ins for customer specific components a toolkit is available (**Automation Platform Toolkit**, see separate documentation).

To look at or to modify a profile the **Installation and Profile Manager** (IPM.exe) is needed. It provides a dialog for installing plug-ins on your system and another dialog for defining profiles basing on the installed plug-ins.

Profile configurations are stored in files with extension ".profile".

Regard the following:

- simultaneous installation of several versions is possible
- different versions of the runtime system and the programming system work together
- every service employee has available all versions
- every project knows with which version it was created; when it gets opened with another version, the user will be hinted at possible resp. required updates concerning storage format version, library versions etc.
- individual functions can be added without having to update the whole version

3.4 Project

A project contains the POU objects which make up a **PLC program**, as well as the definitions of resource objects necessary to run one or several instances of the program (application) on certain target systems (PLCs, devices). POU objects are managed in the **POUs** view window, device-specific resource objects are managed in the **Devices** view window.

A project is saved in a file **<projectname>.project.**

Project specific configuration can be done in the Customize and Options dialogs.

Note: The look and the properties of the user interface are defined and stored in »PLC Designer«, not in the project. The components available in the user interface are defined by the profile.

Project Template

The basic configuration of a new project (menu structure, predefined objects) is determined by the currently used project **template**. The template is chosen when creating a new project file by command New Project.

Project Settings and Project Information

The Project Settings per default is an object in the POUs view, the Project Information will be automatically added also there as soon as it has been called once by the corresponding menu command (usually part of the Project menu).

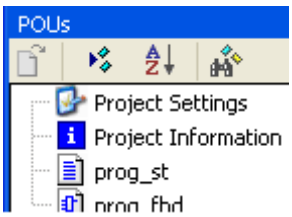


Abb. 3 POUs window

The Project Settings dialogs provide the possibility to define various settings for the current project, e.g. what should happen, if an user opens the project with a different version.

The Project Information dialogs can be used to edit or view project specific information like e.g. file data, statistics on the objects, author name etc. For a detailed description see the help page on Project Information.

Merging projects

A project resp. objects of a project can be copied to another project. See command Merge Project.

Conversion of »PLC Designer« V2.3 projects

Projects created with version 2.3 can be opened and will be converted to V3.x format. Some syntactical preparation however is recommended. If libraries are included in a V2.3 project, you can decide before the project gets converted, whether they should be used further on or should be replaced by newer ones or should be removed from the project. For details please see help page: 'Project' 'Open'.

3.5 Supported Programming Languages

All of the programming languages mentioned in the IEC standard IEC 61131 are supported via specially adapted editors:

FBD/LD/IL editor for Function Block Diagram (FBD), Ladder Logic Diagram (LD) and Instruction List (IL)

SFC editor for Sequential Function Chart

ST editor for Structured Text

Additionally »PLC Designer« provides an editor for programming in CFC that is not part of the IEC standard:

CFC editor for Continuous Function Chart

3.6 What is a Program Organization Unit Object "POU"

Do not get puzzled with the term "**POU**":

It is **basically** used for all objects which are used to create a PLC program**.**

POU = Program Organization Unit

POUs which are managed in the "**POUs**" view are not device-specific but they might be instantiated for the use on a device (application). For this purpose program POUs must be called by a task of the respective application.

POUs, which are ONLY managed in the Devices view, i.e. which are inserted in the "device tree" explicitly below an application, can only be instantiated by applications indented below this application ("child" application). For further information see the descriptions of the device tree and of the Application object.

BUT "POU" also is the name of a certain **sub-category** of these objects in the Add Object menu, at this place just comprising programs, function blocks and functions!

So, a Program Organization Unit object in general is a programming unit, an object which is managed either non-device-specifically in the "POUs" window or device-specifically in the "Devices" window and can be viewed and edited in an editor window. A POU object can be a program, function, function block as well as a method, action, interface, DUT (Data Unit Type) or an external file of any format.

Regard the possibility to set certain **properties** (like e.g. build conditions etc.) for each particular POU object.

In order to add a programming object to the project use command 'Add Object'.

The following **POU object types** can be used per default:

- Action
- DUT (Data Type Unit)
- External File
- Global Variable List
- Global Network Variables List
- Image pool
- Interface
- Library Manager
- Method
- Property
- POU
- Textlist
- Visualization

Besides the Program Organization Unit objects there are Device objects used for running the program on the target system (Resource, Application, Task Configuration etc...). Those are managed in the "Devices" view.

3.7 Program Organization Objects - "POUs"

A POU - in this context - is a Program Organization Unit of type Program, Function or Function block. Regard the help page What is a Program Organization Unit Object "POU" for information on the general usage of the term "POU" for all Program Organization Units managed in the POUs view ! Please see also there for information on the project-global resp. device-specific management of POUs.

In order to **add** a POU select an appropriate entry in the POUs or Devices tree (for example an application object), use command Add Object from the context menu and select "POU" from the appearing submenu. The **Add POU** dialog will open, where you have to configure the POU concerning name, type and implementation language. In case of a function block optionally EXTENDS and IMPLEMENTS properties can be defined, in case of a function also the return type must be specified. See the respective help pages on Program, Function, Function Block.

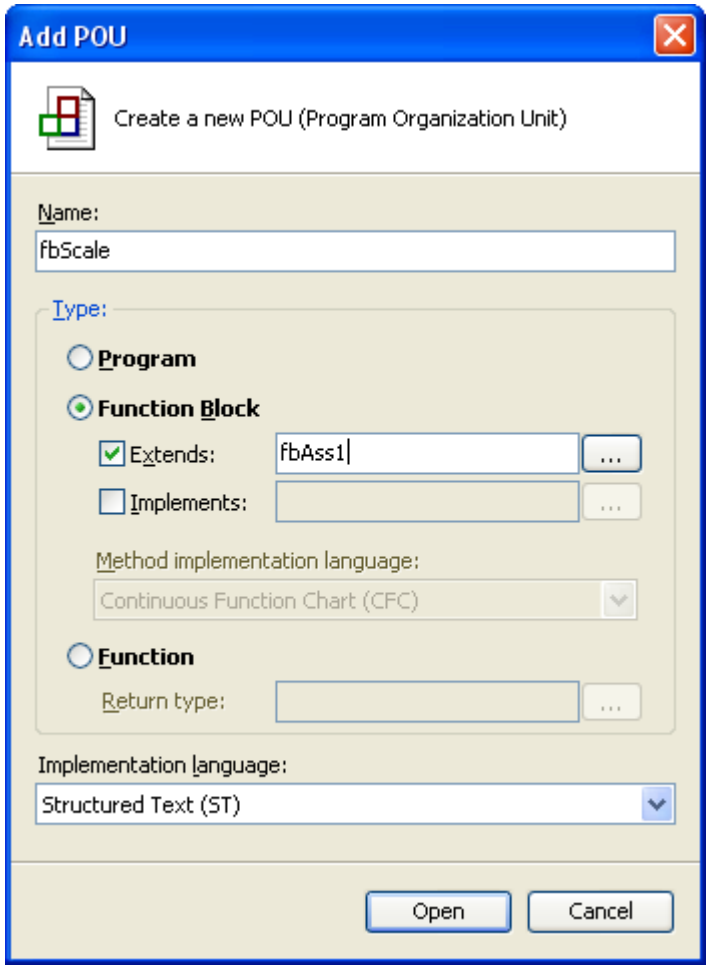


Abb. 4 Add POU dialog, example

Depending on the type the POU can be supplemented by methods, properties, actions, transitions; for this also use the "Add Object" command.

The hierarchical order of processing the POU instances of an application depends on a device specific configuration (call stack).

Each POU consists of a **declaration part** and a **implementation part**. The body is written in one of the available programming languages, e.g. IL, ST, SFC, FBD, LD or CFC.

»PLC Designer« supports all POUs described by the standard IEC 61131-3. If you want to use these POUs in your project, you must include the library standard.library.

Calling POUs:

POUs can call other POUs. Recursions however are not allowed.

When a POU assigned to an application calls another POU **just by its name** (without any namespace added), regard the following order of browsing the project for the POU to be called:
1. Current application, 2. Library Manager of current application, 3. POUs view, 4. Library Manager in POUs view.

If a POU with the name specified in the call is available in a library of the applications library manager as well as as object in the POUs view, there is no syntax for explicitly calling the POU in the POUs view just by using its name. In this case you would have to move the concerned library from the applications library manager to the POUs view library manager. Then you could call the POU from the POUs view just by its name (and if needed that from the library by preceding the library namespace).

Note: The following fact should be described in the online help: When in a POU in an Application, you call another POU just by its name (for example, "GetVersion"), »PLC Designer« will try to find that POU first in the application, then in the Library Manager in the application, then in POUs, then in the Library Manager in POUs. So when you have two GetVersion functions, in Application/Library Manager/Conveying and in POUs, the first one has priority. There is no syntax for calling the second one - you have to move the Conveying library from the Application library manager to a POUs Library Manager, then you can call the Conveying GetVersion by "Conveying.GetVersion" and your own GetVersion by "GetVersion".

Note: Regard the special POUs for implicit checks.

3.7.1 Program

A program is a POU which returns one or several values during operation. All values are retained from the last time the program was run until the next.

A program POU can be **added** to the project via the Add Object command. To assign the program to an existing application, select the application entry in the Devices view and use the command from the context menu. Otherwise it will be added to the POUs view. In the 'Add POU' dialog choose type 'Program', enter a name for the program and set the desired implementation language. After confirming the settings with button **Open** the editor window for the new program will open and you can start editing the program:

The syntax for declaring a program:

PROGRAM <program name>

This is followed by the variable declarations of input, output and program variables, optionally also access variables.

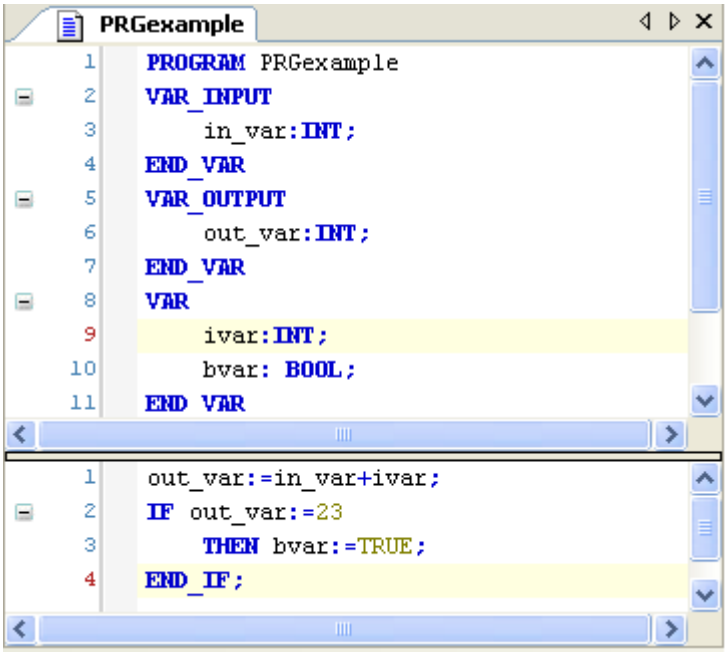


Abb. 5 Example of a program

Program calls:

A program can be called by another POU. But: A program call in a function is not allowed. There are also no instances of programs.

If a POU has called a program and if thereby the values of the program have been changed, these changes will be retained until the program gets called again, even if it will be called from within another POU. Regard that this is different from calling a function block, where only the values in the given instance of the function block are changed and so the changes only are of effect when the same instance will be called again.

If you want to set input and/or output parameters in the course of a program call, you can do this in text language editors (e.g. ST) by assigning values to the parameters after the program name in parentheses. For input parameters this assignment takes place using ":=" just as with the initialization of variables at the declaration position, for output parameters "=>" is to be used; see for an example below.

If the program is inserted via input assistant with option **Insert with arguments** in the implementation window of a text language editor, it will be displayed automatically according to this syntax with all parameters. Though you not necessarily must assign these parameters.

Examples for program calls:

In IL:

```
CAL          PRGexample      (
       in_var:= 33           )
LD           PRGexample.out_var
ST           erg
```

or with assigning the parameters (input assistant "With arguments", see above):

```
CAL          PRGexample      (
       in_var:= 33           ,
      out_var=> erg          )
```

In ST:

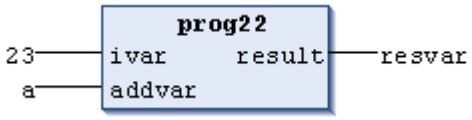
Note that in contrast to V2.3 the **brackets are obligatory**!

```
PRGexample();
erg := PRGexample.out_var;
```

or with assigning the parameters (input assistant "With arguments", see above):

```
PRGexample(in_var:=33, out_var=>erg );
```

In FBD:



3.7.2 Function

A function is a POU, which yields exactly one data element (which can consist of several elements, such as fields or structures) when it is processed, and whose call in textual languages can occur as an operator in expressions.

A function POU can be **added** to the project via command "Add Object / POU". To assign the function to an existing application, select the application entry in the Devices view and use the command from the context menu. Otherwise it will be added to the POUs view. In the **Add POU** dialog choose type 'Function', enter a name (<function name>) and a return data type (<data type>) for the new function and choose the desired implementation language. After confirming the settings with button **Open** the editor window for the new function will open and you can start editing:

Declaration:

A function declaration begins with the keyword FUNCTION. A name and a data type must be defined.

Syntax:

FUNCTION <function name> : <data type>

This is followed by the variable declarations of input and function variables.

A result must be assigned to a function. That is that the function name is used as an output variable.

Note: If a local variable is declared as RETAIN in a function, this is without any effect ! The variable will not be written to the Retain area.

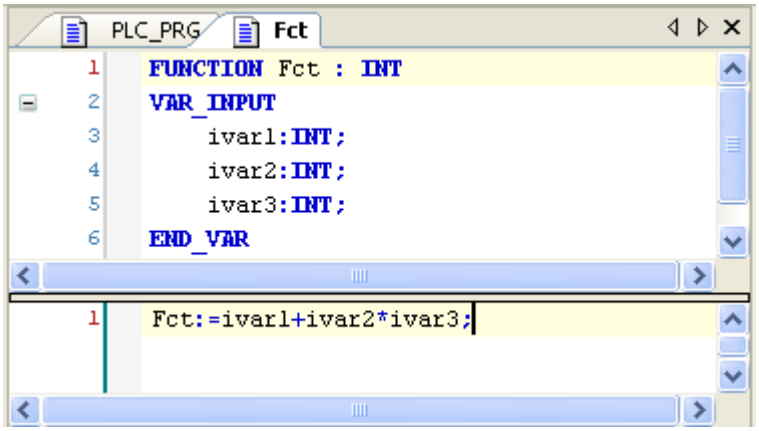


Abb. 6 Example of a function in ST (This function takes three input variables and returns the product of the second two added to the first one)

Function call:

The call of a function in ST can appear as an operand in expressions.

In IL a function call only can be positioned within actions of a step or within a transition.

Functions (in contrast to a program or function block) contain no internal state information, that is, invocation of a function with the same arguments (input parameters) always will yield the same values (output). For that reason functions may not contain global variables and addresses.

Examples of function calls:

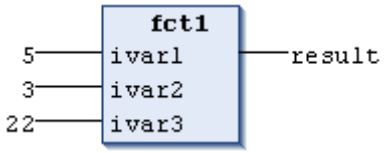
in IL:

```
LD      5
Fct     3   ,
        22
ST      result
```

in ST:

```
result := fct1(5, 3, 22);
```

in FBD:



Note: In contrast to »PLC Designer« V2.3 in function calls it is no longer possible to mix explicit parameter assignment with implicit ones.
This allows to change the order of parameter input assignments.
Example:
fun(formal1 := actual1, actual2); // -> error message
fun(formal2 := actual2, formal1 := actual1); // same semantics as the following:
fun(formal1 := actual1, formal2 := actual2);
However you have to notice the different parameter assignment when using V2.3 projects!

According to the IEC 61131-3 standard, functions can have **additional outputs**. Those must be assigned in the call of the function, for example in ST according to syntax

out1 => <output variable 1> | out2 => <output variable 2> | ...further output variables

Example: Function *fun* is defined with two input variables *in1* and *in2*. The return value of *fun* will be written to the locally declared output variables (VAR_OUTPUT) *loc1* and *loc2*.

```
fun(in1 := 1, in2 := 2, out1 => loc1, out2 => loc2);
```

Extensible functions (not yet implemented)

As extension to the IEC 61131-3 standard it is possible to declare functions or methods with a variable number of input parameters of the same type.

3.7.3 Function Block

A function block is a POU which provides one or more values during the processing of a PLC program. As opposed to a function, the values of the output variables and the necessary internal variables shall persist from one execution of the function block to the next. So invocation of a function block with the same arguments (input parameters) need not always yield the same output values.

In addition to the functionality described by standard IEC11631-3 **object oriented programming** will be supported and function blocks can be defined as extensions of other functions blocks and they can include interface definitions concerning method invocation. This means that "inheritance" can be used when programming with function blocks.

A function block **always is called via an** instance, which is a reproduction (copy) of the function block.

A function block can be **added** to the project via command "Add Object / POU". To assign the function block to an existing application, select the application entry in the Devices view and use the command from the context menu. Otherwise it will be added to the POUs view. In the **Add POU** dialog choose type 'Function',

In the 'Add object' dialog choose type 'Function Block', enter a function block name (<identifier>) and choose the desired implementation language. Additionally the following options may be set:

Extends: Enter here the name of another function block available in the project, which should be the base for the current one.

I**mplements:** Enter here the name(s) of interfaces available in the project, which should be implemented in the current function block. Several interfaces can be entered separated by commas. In the field **Method implementation language** for all method objects created via the interface implementation you can choose the desired programming language, independently from that set for the function block itself.

After having confirmed the settings with button **Open** the editor window for the new function block will open and you can start editing:

Declaration:

Syntax:

FUNCTION_BLOCK <function block name> | EXTENDS <function block name> | IMPLEMENTS <comma-separated list of interface names>

This is followed by the declaration of the variables.

Note: In contrast to »PLC Designer« V2.3 "FUNCTIONBLOCK" is no longer a valid alternative key word.

Example:

FBexample shown in the following picture has two input variables and two output variables out1 and out2. Out1 is the sum of the two inputs, out2 is the result of a comparison for equality.

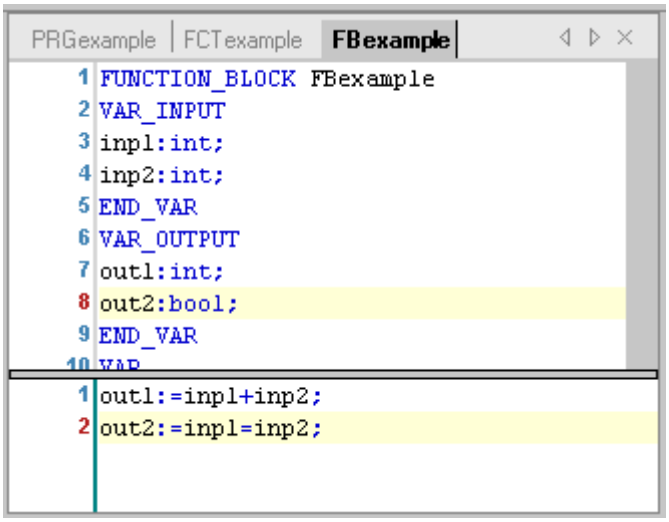


Abb. 7 Example of a function block in ST

Function Block Instance

Function blocks are always called through an instance which is a reproduction (copy) of a function block.

Each instance has its own identifier (instance name), and a data structure containing its inputs, outputs, and internal variables.

Instances like variables are declared locally or globally, whereby the name of the function block is indicated as the data type of an identifier.

Syntax for declaring a function block instance:

<identifier>:<function block name>;

Example

Declaration (e.g. in the declaration part of a program) of instance INSTANCE of function block FUB:

```
INSTANCE: FUB;
```

The declaration parts of function blocks and programs can contain instance declarations. But: Instance declarations are not permitted in functions.

Calling a function block

Function blocks are always called through a function block instance. Thus a function block instance must be declared locally or globally.

Then the desired function block variable can be accessed using the following syntax:

Syntax:

<instance name>.<variable name>

Regard the following:

- Only the input and output variables of a function block can be accessed from outside of an function block instance, not its internal variables.
- Access to a function block instance is limited to the POU in which it was declared unless it was declared globally.
- At calling the instance the desired values can be assigned to the function block parameters. See below.
- Please regard, that the InOutVariables (VAR_IN_OUT) of a function block are passed as pointers.
- In SFC function block calls can only take place in steps.
- The instance name of a function block instance can be used as an input parameter for a function or another function block.
- All values of a function block are retained until the next processing of the function block. Therefore function block calls do not always return the same output values, even if done with identic arguments!

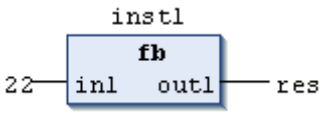
Note: If there at least one of the function block variables is a **remanent** variable, the total instance is stored in the retain data area.

Examples for accessing function block variables

Assume: Function block *fb* has an input variable *in1* of the type INT. See here the call of this variable from within program prog. See declaration and implementation in ST:

```
PROGRAM prog
VAR
inst1:fb;
 res:INT;
END_VAR

 inst1.in1:=22;  (* fb is called and input variable in1 gets assigned value 22 *)
 inst1();        (* fb is called, this is needed for the following access on the output variable *)
res:=fbinst.outl; (* output variable of fb is read *)
```

Example in FBD:



Assigning parameters at call:

In the text languages **IL and ST y**ou can set input and/or output parameters immediately when calling the function block. The values can be assigned to the parameters in parentheses after the instance name of the function block. For input parameters this assignment takes place using ":=" just as with the initialization of variables at the declaration position, for output parameters "=>" is to be used.

Example, call with assignments:

In this example a timer function block (instance CMD_TMR) is called with assignments for the parameters IN and PT. Then the result variable Q is assigned to the variable A. The result variable is addressed with the name of the function block instance, a following point, and the name of the variable:

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

If the instance is inserted via **input assistant** (<F2>) with option **With arguments** in the implementation window of a ST or IL POU, it will be displayed automatically according to the syntax showed in the following example with all of its parameters. But you not necessarily must assign these parameters. For the above mentioned example the call would be displayed as follows:

Example, insert via input assistant with arguments:

```
CMD_TMR(in:=, pt:=, q=>)
-> fill in, e.g.:
CMD_TMR(in:=bvar, pt:=t#200ms, q=>bres);
```

Extension of a function block

Supporting **object orientated programming** a function block can be derived from another function block. This means a function block can extend another, thus automatically getting the properties of the basing function block in addition to its own.

The extension is done by using the keyword EXTENDS in the declaration of a function block. You can choose the "extends" option already during adding a function block to the project via the Add Object dialog.

The syntax:

FUNCTION_BLOCK <function block name> EXTENDS <function block name>

This is followed by the declaration of the variables.

Example:

Definition of function block fbA:

```
FUNCTION_BLOCK fbA
VAR_INPUT
```

x:int;

```
....
```

Definition of function block fbB:

```
FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
```

ivar:int;

```
....
```

Extension by EXTENDS means:

- fbB does contain all data and methods which are defined by fbA. An instance of fbB can now be used in any context were a function block of type fbA is expected.
- fbB is allowed to override the methods defined in fbA. That means: fbB can declare a method with the same name and the same inputs and output as declared by A.
- fbB is not allowed to use function block variables with the same name as used in fbA. In this case the compiler will prompt an error.
- fbA variables and methods can be accessed directly within a fbB-Scope by using the keyword "**SUPER**" (SUPER^.<method>).

Note: Multiple inheritance is not allowed.

Implementing interfaces

In order to support **object oriented programming** a function block can implement several interfaces which provide the possibility to use methods:

Syntax:

FUNCTION_BLOCK <function block name> IMPLEMENTS <interface_1 name>|,<interface_2 name>, ..., <interface_n name>

A function block that implements an interface must contain all methods of this interface. That is name, inputs and the output of the particular method must be exactly the same. For this purpose - when creating a new function block implementing an interface - automatically all methods defined in this interface will be inserted below the new function block in the object tree. Currently however any modifications in the interface definition will not be updated automatically in the concerned function blocks. This must be done explicitly by command 'Implement interfaces...'.

Example:

INTERFACE I1 includes Method GetName:

```
METHOD GetName : STRING
```

Function blocks A and B each implement interface I1:

```
FUNCTION_BLOCK A IMPLEMENTS I1
FUNCTION_BLOCK B IMPLEMENTS I1
```

Thus in both function blocks the method "GetName" must be available and will be inserted automatically below each when the function blocks get inserted in the object tree.

Now consider a declaration of a variable of type I1:

```
FUNCTION DeliverName : STRING
VAR_INPUT
  I_i : I1;
END_VAR
```

This input can receive all function blocks that implement interface I1.

Example for function calls:

```
DeliverName(I_i := A_instance); // call with instance of type A
DeliverName(I_i := B_instance); // call with instance of type B
```

Note that a variable of an interface-type...

... must get assigned an instance of a function block before a method can be called on it.

... always is a reference to the assigned function block instance. Thus a call to the interface method does result in a call to the function block implementation. As soon as the reference is assigned, the corresponding address will be monitored in online mode. Otherwise, if no reference has been assigned yet, the value 0 will be shown within monitoring in online mode.

Example:

in the implementation part of the above described function "DeliverName":

```
DeliverName := I_i.GetName();  // in this case it depends on the "real" type of I_i whether A.GetName or B.GetName is called.
```

Note: See also the possibility to extend a function block by using keyword EXTENDS in the declaration.

Method invocation

Object oriented programming with function blocks is - besides of the possibility of extension via EXTENDS - supported by the possible use of interfaces and inheritance. This requires dynamically resolved method invocations, also called "virtual function calls".

See Method for further information on methods.

Virtual function calls need some more time than normal function calls and are used when:

- a call is performed via a pointer to a function block (pfub^.method)
- a method of an interface variable is called (interface1.method)
- a method calls another method of the same function block

Virtual function calls make possible that the same call in a program source code will invoke different methods during runtime.

According to the IEC 61131-3 standard, methods like "normal" functions can have **additional outputs**. Those must be assigned in the method call according to syntax
<method>(in1:=<value> |, further input assignments, out1 => <output variable 1> | out2 => <output variable 2> | ...further output variables)
which effects that the output of the method is written to the locally declared output variables as given within the call.

Example:
Assume that function blocks *fub1* and *fub2* EXTEND function block *fubbase* and IMPLEMENT *interface1*. Method *method1* is contained.
Possible use of the interfaces and method calls:

```
VAR_INPUT
  b : BOOL;
END_VAR
VAR
  pInst : POINTER TO fubbase;
  instBase : fubbase;
  inst1 : fub1;
  inst2 : fub2;
  interfaceRef : interface1;
END_VAR

IF b THEN
  interfaceRef := inst1;    (* Interface1 for fub1 *)
  pInst := ADR(instBase);
ELSE
  interfaceRef := inst2;    (* Interface1 for fub2 *)
  pInst := ADR(inst1);
END_IF

pInst^.method1();      (* If b is true, fubbase.method1 is called, else fub1.method1 is called *)
interfaceRef.method1(); (* If b is true, fub1.method1 is called, else fub2.method1 is called *)
```

Now assume that *fubbase* of the upper example contains two methods *method1* and *method2*. *fub1* overrides *method2* but not *method1*.

method1 is called like shown in the upper example:

```
pInst^.method1(); (*If b is true fubbase.method1 is called, else fub1.method1 is called*)
```

Call via THIS-pointers:

Let's have a look at the body of *method1*:

```
METHOD method1 : BOOL
VAR_INPUT
END_VAR
method1 := method2();
```

For calling *fubbase.method1* via THIS pointers the following is true:

If THIS is of type *fubbase*, *fubbase.method2* will be called. If THIS is of type *fub1*, *fub1.method2* will be called

3.7.4 Data Type Unit (DUT)

Along with the standard data types the user can define own data types. Structures, enumeration types and references can be created as Data Type Units (**DUT**s) in a DUT editor.

For a description of the particular standard and the user defined data types see 'Data Types'.

A -DUT object can be **added** to the project via the Add Object command. To assign it to an existing application, first select the application entry in the Devices view. Otherwise it will be added to the POUs view. In the 'Add DUT' dialog enter a name for the new data type unit and choose the desired type **Structure**, **Enumeration**, **Alias** or **Union**.

In case of type "structure" you might utilize the principle of inheritance, thus supporting **object oriented programming**. Optionally you can specify that the DUT should **extend another DUT**, which is already defined within the project. This means that the definitions of the extended DUT will be automatically valid within the current one. For this purpose activate option **Extends:** and enter the name of the other DUT.

After confirming the settings with button **Open** the editor window for the new DUT will open and you can start editing.

Syntax for declaration of a DUT:

TYPE <identifier> : <DUT components declaration>

END_TYPE

The DUT components declaration depends on the type of DUT, e.g. a structure or an enumeration.

Example:

See in the following two DUTS, defining structures struct1 and struct2; struct2 extends struct1, which means that you can use "`struct2.a`" in your implementation to access variable a.

```
TYPE struct1 :
STRUCT
 a:INT;
 b:BOOL;
END_STRUCT
END_TYPE
```

```
TYPE struct2 EXTENDS struct1 :
STRUCT
 c:DWORD;
 d:STRING;
END_STRUCT
END_TYPE
```

3.7.5 Method

Note: This functionality is only available if supported by the currently used feature set.

Supporting **object oriented programming**, methods can be used to describe a sequence of instructions. Like a function a method is not an independent POU, but **must be assigned to a** function block. It can be regarded as a function which contains an instance of the respective function block.

Also as a means of **object oriented programming** Interfaces can be used to organize the methods available in a project. An Interface is a collection of method-"prototypes", that means a method assigned to an interface just contains a declaration part, but no implementation. The implementation of the method is done in the function block which implements the interface and uses the method.

The advantage: You can use one and the same method call in all function blocks which are implementing the same interface, that means the call can be used for different purposes. Calling a method is like telling roughly what purpose should be served and it depends on the currently concerned function block which instructions (implementation) will be carried out in detail to serve this purpose.

Note: When copying or moving a method from a POU to an Interface the contained implementations will be deleted automatically. When copying or moving from an Interface to a POU the user will be asked to specify the desired implementation language.

Inserting:

To assign a method to a function block or interface select the appropriate function block or interface entry in the POUs or Device tree and in the context menu use Add Object / Method. In the **Add Method** dialog enter a **Name**, the desired **Return type**. and **Implementation language**. For choosing the return data type you can use the ... button to get the Input Assistant dialog. After having confirmed the settings via **Open** the method editor view will be opened:

Declaration:

Syntax:

```
METHOD <method name> : <data type>
VAR_INPUT
x: INT;
END_VAR
```

For a description on how to define interfaces handling methods, see: Interface.

Method call:

Method calls are also named "virtual function calls". Please see: Method invocation.

Note:

All data of a method is temporary and only valid during the execution of the method (stack variables).

In the body of a method access to the function block instance variables is allowed.

THIS Pointer: You can use identifier "THIS" to point directly to the implicit function block instance, which is available automatically. Notice that a locally declared variable might hide a function block variable. So in case of the example shown above in the syntax description "THIS^.x" would not refer to the methods input x, but to the function block variable x.

VAR_IN_OUT or VAR_TEMP-variables of the function block can not be accessed in a method!

Methods defined in an Interface are only allowed to have input-, output and inout-variables, but no body (implementation part).

Methods, like functions, can have additional outputs. Those must be assigned during method invocation

Special methods for a function block:

Init method: A method named "FB_init" always is declared implicitly, but also can be declared explicitly. It contains initialization code for the function block as declared in the declaration part of the function block. See: FB_Init method.

Reinit method: If a method named "FB_reinit" is declared for a function block instance, it will be called after the instance has been copied and will reinitialize the new instance module. See: FB_reinit method.

Exit method: If an exit method named "FB_exit" is desired (for example for deallocating any , it must be declared explicitly. There is no implicit declaration. The exit method will be called for each instance of the function block before a new download, a reset or during online change for all moved or deleted instances. See: FB_exit method.

Properties: Set and Get methods; see: Property.

Method call also when application is stopped

In the device description file it can be defined that a certain method should always be called task-cyclically by a certain function block instance (of a library module). If this method has the following input parameters, it will be processed also when the active application currently is not running.

```
VAR_INPUT
pTaskInfo : POINTER TO DWORD;
pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO;
END_VAR
```

The programmer now can check the application status via pApplicationInfo, and define what should happen. Example: IF pApplicationInfo^.state=RUNNING THEN <instructions> END_IF.

3.7.6 Property

Note: This functionality is only available if supported by the currently used feature set.

A "**property**" is an object type, which can be inserted below a program or function block via command "Add object / Property" from the context menu. In the **Add Property** dialog the **name**, **return type** and desired **implementation language** have to be specified.

A "property" contains two special methods: These will be inserted automatically in the objects tree below the property object:

- the "Set" method is called when the property is written, that is the name of the property is used as input.

- the "Get" method is called when the property is read, that is the name of the property is used as output.

Example:

Function block FB1 uses a local variable "milli". This variable is determined by the properties Get and Set:

Get:

```
seconds := milli / 1000;
```

Set:

```
milli := seconds * 1000;
```

You can write the property of the function block (Set method) for example by "fbinst.seconds := 22;" (fbinst is the instance of FB1)

You can read the property of the function block (Get method) for example by "testvar := fbinst.seconds;"

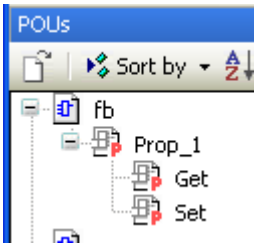


Abb. 8 Example: Property "seconds" added to function block FB1

A property can have additional local variables but no additional inputs and - in contrast to a function or method - no additional outputs.

Note: When copying or moving a Property from a POU to an Interface the contained implementations will be deleted automatically. When copying or moving from an Interface to a POU the user will be asked to specify the desired implementation language.

Monitoring a property

A property may be monitored in online mode either with help of Inline Monitoring or with help of a Watch List. The condition precedent to monitoring a property is the addition of the pragma '{attribute 'monitoring':='variable'}' on top of its definition.

3.7.7 Interface

Note: This functionality is only available if supported by the currently used feature set.

The use of interfaces is a means of **object oriented programming**.

Interfaces can be used to organize methods which are used by function blocks. An "Interface" is a POU describing a collection of method-prototypes. "Prototype" means that just declarations but no implementation is contained. A function block can implement one or several interfaces which means that basically all methods of the interface(s) are available in the function block.

The advantage: The method calls are defined and thus unique for all function blocks implementing the same interface. Within a function block the methods have to be filled with implementation code.

Regard the following:

- It is not allowed to declare variables within an interface. An interface has no body (implementation part) and no actions. Just a collection of methods is defined within an Interface and those methods are only allowed to have input-, output and inout-variables.
- Variables declared with the type of an interface are always treated as references.
- A function block implementing an Interface must have assigned methods which are named exactly as they are in the interface and which contain identically named inputs, outputs and inouts.

Note: When copying or moving a method or property from a POU to an Interface the contained implementations will be deleted automatically. When copying or moving from an Interface to a POU the user will be asked to specify the desired implementation language.

Inserting:

An Interface can be **added** via command "Add object / Interface". If it should be assigned to an application, select the application in the device tree and use the command from the context menu. Otherwise the new Interface will be added to the POUs tree.

In the '**Add Interface**' dialog enter a name for the new interface (<interface name>). Optionally you can activate option **Extends:** if you want the current interface to be an extension of another interface.

If "ITF1" extends "ITF_base", all methods described by ITF_base will be automatically available in ITF1.

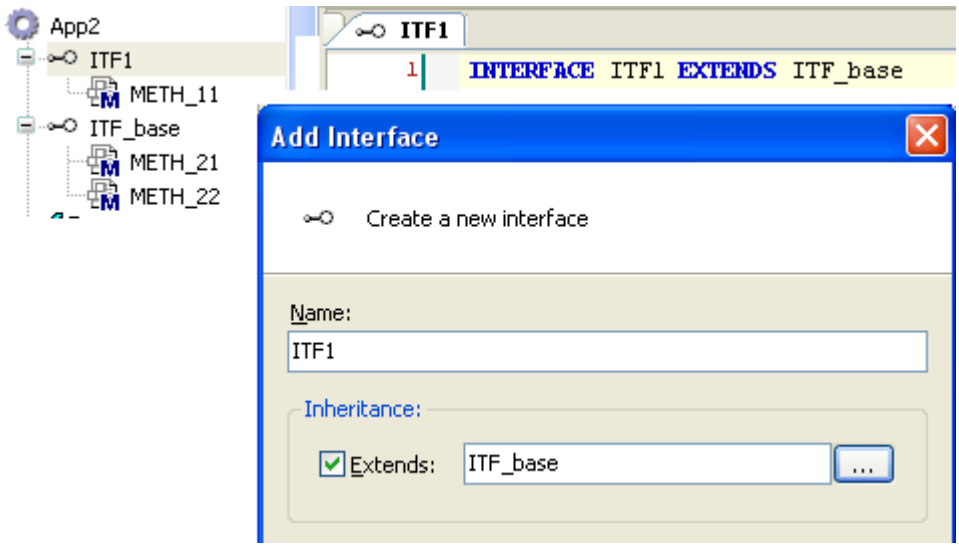


Abb. 9 Example for extending an interface

After having confirmed the settings with button **Open**, the editor window for the new interface will open.

Syntax:

INTERFACE <interface name>

resp. for an interface extending another one:

INTERFACE <interface name> EXTENDS <base interface name>

Example:

INTERFACE itf1 EXTENDS itf_base

Adding the desired collection of methods

To complete the definition of the interface, you have to add the desired collection of methods. For this purpose select the interface entry in the POUs view and choose command 'Add Object' to get the "Add Object' dialog for defining a method to be part of the interface. Add as many methods as desired and regard that those methods are only allowed to have input-, output and inout-variables, but no body (implementation part).

3.7.8 Action

Actions can be defined and assigned to function blocks and programs using the command Add Object. An action is an additional implementation which can be created in a different language than the "basic" implementation is. Each action is given a name.

An action works with the data of the function block or program which it belongs to. It uses the input/output variables and local variables defined there and does not contain own declarations.

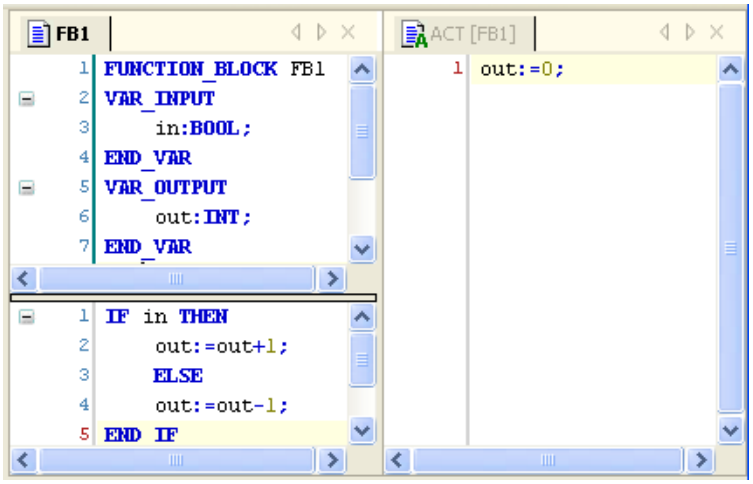


Abb. 10 Example of an action of a function block

In this example each call of the function block "FB1" will increase or decrease the output variable "out", depending on the value of the input variable "in". Calling action 'Reset' of the function block will set the output variable "out" to zero. The same variable "out" is written in both cases.

An action can be **added** via command "Add Object / Action" when the respective program or function block object is selected in the Devices or POUs tree. In the **Add Action** dialog define the action name and desired implementation language.

Calling an action:

An action is called with

<Program_name>.<Action_name>

or

<Instance_name>.<Action_name>

Regard the notation in FBD (see example below)!

If it is required to call the action within its own block, i.e. in the program or function block it belongs to, it is sufficient to just use the action name.

Examples for the call of the above described action from another POU:

Declaration for all examples:

```
PROGRAM PLC_PRG
VAR
 Inst : Counter;
END_VAR
```

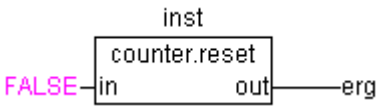
Call of action 'Reset' in another POU, which is programmed in **IL**:

```
CAL Inst.Reset(In := FALSE)
LD Inst.out
ST  ERG
```

Call of action 'Reset' in another POU, which is programmed in **ST**:

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

Call of action 'Reset' in another POU, which is programmed in **FBD**:



Note: The IEC standard does not recognize actions other than actions of the sequential function chart (SFC). There actions are an essential part, containing the instructions to be processed at the particular steps of the chart.

3.7.9 External function, function block, method

For an external function, function block or method no code will be generated by the programming system.

Perform the following steps to create an external POU:

1. Add the desired POU object in the POUs view of your project like any internal object and define the respective input and output variables.

Note: Local variables must be defined in "external" function blocks, but may not be defined in "external" functions or methods! Also notice that VAR_STAT variables can not be used in the runtime system!

2. Define the POU to be external: For this purpose select the POU object in the POUs view and use command 'Properties' from the context menu. Open the 'Build' tab and activate option 'External Implementation (Late link in the runtime system)'

In the runtime system an equivalent function, function block or method must be implemented. At a program download for each "external POU" the equivalent will be searched in the runtime system and - if found - will be linked.

3.7.10 Global Variable List - GVL

A Global Variables List ("GVL") is used to declare global variables. If a GVL is placed in the POUs view, the variables will be available all over the project, if a GVL is assigned to a certain application, the variables will be valid within this application.

A GVL can be added via command "Add Object / Add Global Variable List...". To assign it to an existing application, choose the command from the context menu while the application is selected in the device tree. Otherwise the new GVL object will be added to the POUs view.

The GVL Editor is used to edit a Global Variable List.

If the target system supports the network functionality, the variables contained in a GVL can be defined to be available as network variables, i.e. for a broadcast data exchange with other devices in the network. For this purpose appropriate Network Properties must be configured for the GVL.

Regard that as from compiler version 3.2.0.0 variables declared in GVLs always get initialized before local variables of POUs.

3.7.11 Global Network Variables List - GNVL

Note: This functionality is only available if supported by the currently used feature set.

A global network variables list (GNVL) is only used in the device tree. It defines variables, which are specified as network variables in another device within the network.

Thus a GNVL object can only be added to an application, if a GVL with special network properties (network variables list) is available in one of the other network devices, no matter whether defined in the same or in different projects. If several of appropriate GVLs are found within the current project for the current network, you can choose the desired one from a selection list when adding a GNVL via the Add Object dialog. GVLs from other projects must be imported, see below.

This means that each GNVL in the current device (**receiver**) corresponds exactly to one GVL in another device (**sender**).

Besides a **Name**, also a **Task**, responsible for the handling of the network variables, must be defined when adding the GNVL. Alternatively to directly choosing a **Sender**GVL from another device, an **export file *.gvl** previously generated from that GVL via the "Link to file" Properties can be specified. In any case this will be necessary if the desired GVL is defined within another project. For this purpose select option "Import from file" in the Sender selection list and then enter the file path in the "**Import from file:**" edit field (use the standard dialog for browsing in the file system via the button). A special case is, if you want to set up a network variables communication with a V2.3 project. Please see details in: Network Variables.

The settings can be modified at a later time via the Properties dialog (Network Settings).

The basic GVL as well as the corresponding NVL list object get marked as "network variables lists" by the icon in the devices tree.

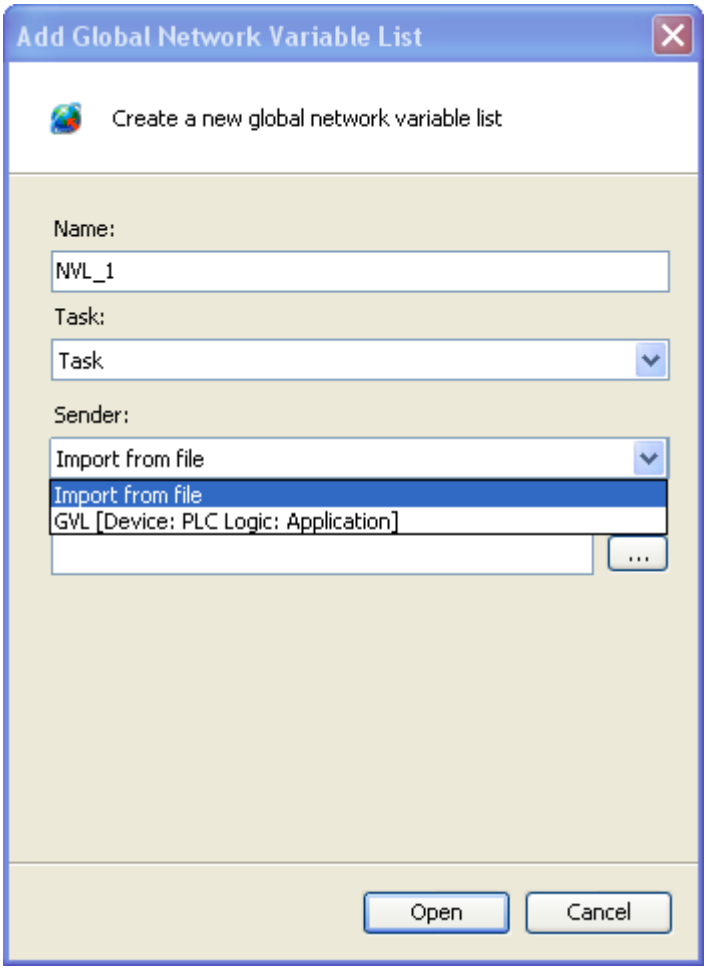


Abb. 11 Add Object Dialog, Global NVL

A GNVL is displayed in an editor view (NVL Editor), however cannot be modified by the user. It just shows the current content of the corresponding GVL, i.e. if you change the basic GVL, the GNVL will be updated accordingly.

Automatically a comment is added at top of the declaration part of a GNVL, providing information on the sender (device path), the GVL name and the protocol type.

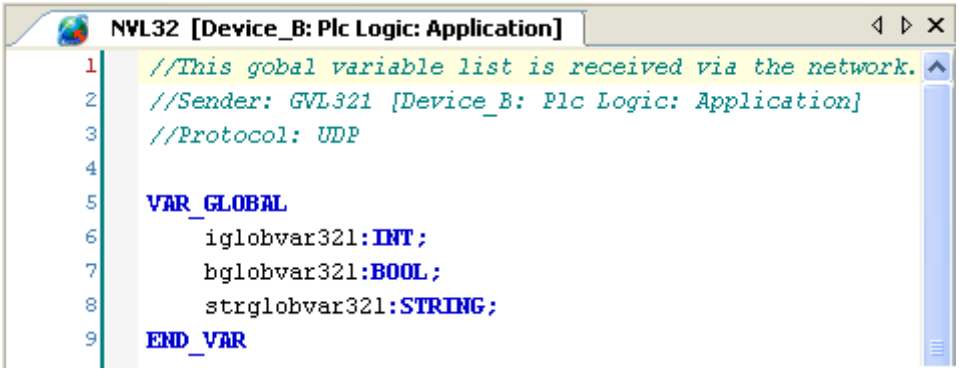


Abb. 12 Global Network Variable List, Example

Please see the general information on using Network Variables**.**

3.7.12 Persistent Variables

This object is a global variables list, which however only contains persistent variables of an application. Thus it must be assigned to an application and for this purpose be inserted in the device tree via the 'Add Object' dialog below this application.

A persistent variables list is edited in the GVL Editor, whereby "VAR_GLOBAL PERSISTENT RETAIN" already is preset in the first line.

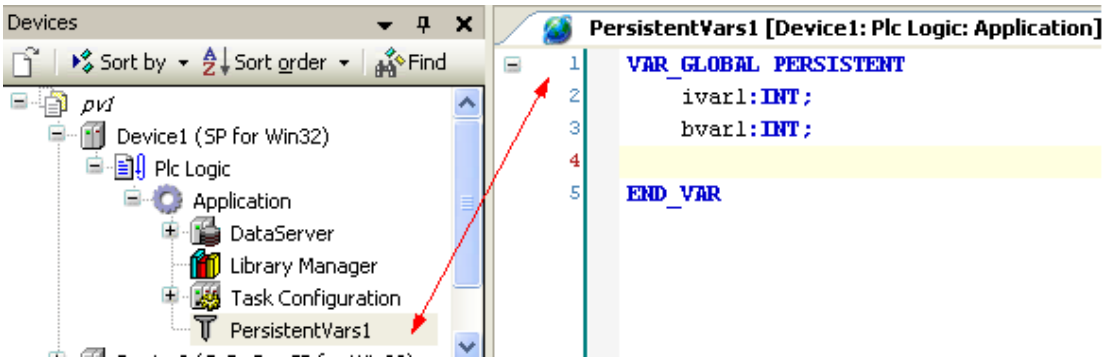


Abb. 13 Example, Persistent Variables list

Persistent variables only get re-initialized at a 'Reset (origin) <application>'. For further information please see the help page on remanent variables.

3.7.13 External File

Note: This functionality is only available if supported by the currently used feature set.

Any external file can be added to the POUs view of a project via the Add Object command: In the 'Add object' dialog choose object type 'External File'. Press button ... for getting the dialog for browsing for a file, the path of which will be entered to the field below **File path:**. In the field below **Name:** automatically the name of the chosen file will be entered without extension. You can edit this field to define another name for the file under which it should be handled within the project.

What do you want to do with the external file ?

Select one of the options:

Remember the link. : The file will be available in the project only if it is available in the defined link path.

Remember the link and embed into project. : A copy of the file will be stored internally in the project but also the link to the external file will be remembered. As long as the external file is available as defined, the defined update options will effect accordingly. Otherwise just the file version stored in the project will be available.

Embed into project. : Just a copy of the file will be stored in the project. There will be no further connection to the external file.

When the external file changes, then ...

If the external file is linked to the project, you can additionally select one of the options:

reload the file automatically : The file will be updated within the project as soon as it has been changed externally

prompt whether to reload the file : A dialog will pop up as soon as the file has been changed externally. You can decide whether the file should be updated also within the project.

do nothing : The file will remain unchanged within the project, even when it is changed externally.

Display file properties... This button opens the standard dialog for the properties of a file, which also appears when you select the file object in the POUs window and use command Properties. The dialog i.a. contains a tab 'External file' where the properties which have been set in the 'Add Object dialog, can be viewed and modified.

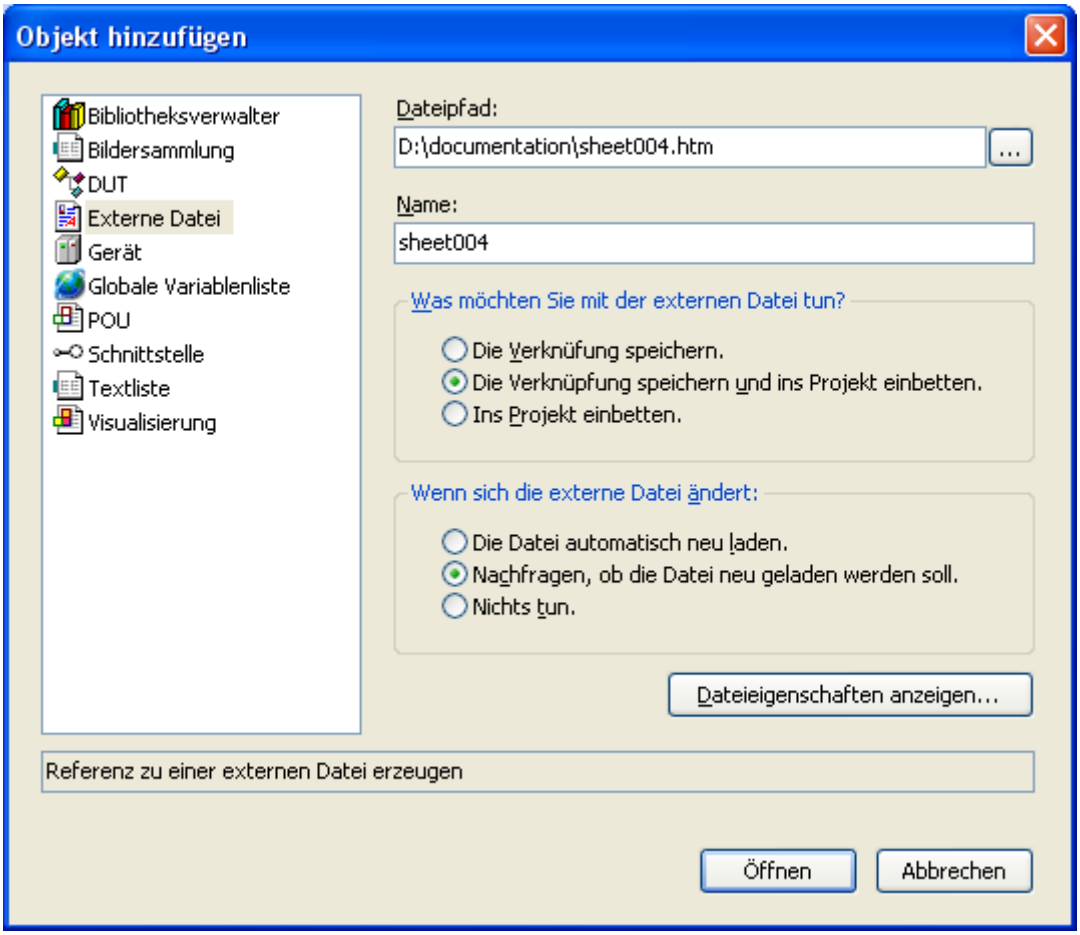


Abb. 14 Add Object dialog for adding an external file

After having done the settings via button **Open** the file will be added to the POUs window and will be opened in that tool which is defined as default for the given file format.

3.7.14 Text List

Note: This functionality is only available if supported by the currently used feature set.

A text list is an object managed globally in the POUs window or assigned to an application in the Devices window. It is the base of

1. **multi-language support** for static and dynamic texts and tooltips in visualizations and in the alarm handling and of

2. **dynamic text** exchange.

Text lists can be exported and (re-) imported. **Export** is necessary, if a language file in xml-format must be provided for a Target-Visualization, but also might be useful for translation purposes (see below).

Possible **formats** of text lists are text and XML. Support of **Unicode** can be activated.

Each text list is uniquely defined by its **namespace**. It contains text strings which are uniquely referenced within the list by an **identifier** (ID, consisting of any sequence of characters) and a **language** identifier. The text list to be used is specified when configuring the text for a visualization element.

Depending on the language which is currently set in the visualization the corresponding text string will be displayed in online mode. The language used in a visualization can be changed by a "Change the language" input, i.e. by an appropriately configured mouse action on any visualization element. Each text is available in the list at least in the "default" language (the default language depends on the currently used »PLC Designer« configuration) and optionally in further languages. If no entry is found which matches the language currently set, the **default language** entry will be used. Each text might contain formatting definitions.

Basic structure of a text list:

Identifier (Index)	Default	<language 1>	<language 2>	.... <language n>
<unique string of characters>	<text abc in default language>	<text abc in language 1>	<text abc in language 2>	...
<unique string of characters>	<text xyz in default language>	<text xyz in language 1>	<text xyz in language 2>	...

Text List types

There are two types of text usable in visualization elements and correspondingly there are two types of lists:

1. GlobalTextList **for static texts:**

Static texts in a visualization in contrast to dynamic texts **cannot be exchanged by a variable in online mode**. Just the language version will be adapted in case it gets changed by a "Change the language" input on any visualization element. A static text is assigned to a visualization element via property 'Text' or 'Tooltip' in category 'Texts'. As soon as the first static text is defined in a project, a text list object named "GlobalTextList" gets **added automatically** to the POUs window. It primarily will contain the defined text string in column "Default" and an automatically assigned integer number as text identifier. Further static texts will be added as soon as they are defined in the Properties of a visualization element. The identifier number each will be counted up.

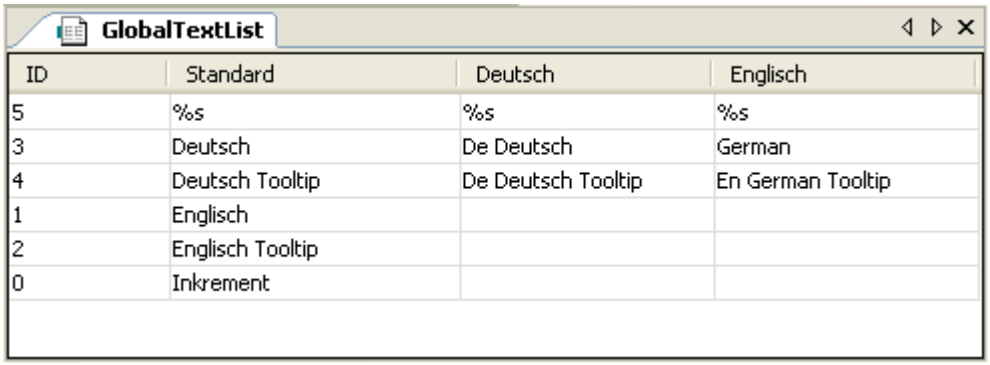
If a static text is inserted into a visualization element (for example if in a rectangle in property category „Texts“, property „Text“ the string „Example“ is specified), this text will be looked up for in the GlobalTextList.

- If the text is found, (for example ID „4711“, Text „Example“), in the element the value 4711 of „TextId“ will be assigned to an internal variable. This is how a relation between the element and the corresponding line in the GlobalTextList is created.
- Otherwise, a new line will be inserted in the GlobalTextList (for example ID „4712“, Text „Example!“). In the element now the value 4712 will be assigned to the internal variable.

This means that each modification of a static text within the visualization may provoke a modification in the GlobalTextList as well.

Notice that - if not yet existing - a global text list also can be **created explicitly** by command Create Global Text List.

GlobalTextList is a special text list where the identifiers for the particular text entries are handled implicitly and are not editable in »PLC Designer« by the user. Also the list cannot be deleted. However the list can be exported, edited externally and then re-imported. In this case at the re-import it will be checked whether the identifiers are still matching those which are used in the configuration of the respective visualization elements. If necessary an implicit update of the identifiers used in the configuration will be done.



ID	Standard	Deutsch	Englisch
5	%s	%s	%s
3	Deutsch	De Deutsch	German
4	Deutsch Tooltip	De Deutsch Tooltip	En German Tooltip
1	Englisch		
2	Englisch Tooltip		
0	Inkrement		

Abb. 15 Example of GlobalTextList

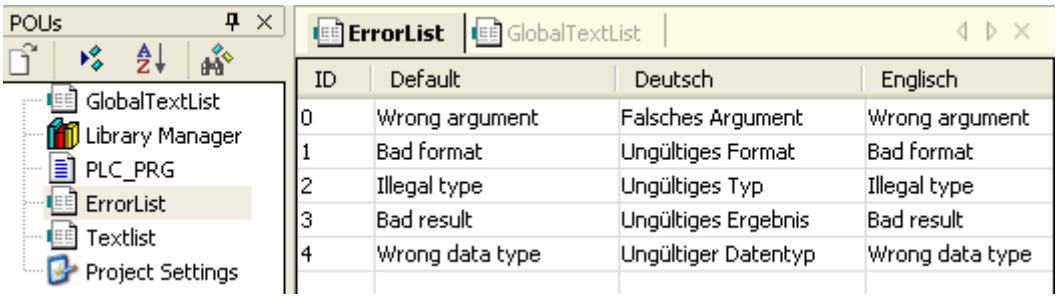
2. Textlist **for dynamic texts:**

Dynamic texts (see above) can be **changed dynamically in online mode**. The text index (**ID)**, which is a string of characters, must be unique within the used text list and in contrast to GlobalTextLists must be defined by the user. Also in contrast to the GlobalTextList, text lists for dynamic texts **must be explicitly created** via command 'Add Object'.

All currently available dynamic text lists will be offered when configuring a visualization element via property 'Dynamic texts' / 'Textlist'. If you there specify a text list name combined with the text index (ID) - which can be entered directly or by entering a project variable which defines the ID string - the current text can be changed in online mode.

A dynamic text list **must be exported if** it is needed as a language file for language switching in a Target-Visualization. The file path is to be specified in the Visualization Options. Like GlobalTextList a dynamic text lists also can be exported for the purpose of external editing and re-import.

Note: In contrast to GlobalTextList at a re-import of dynamic text lists there is **no automatic check and update of the identifiers**. So make sure that the identifiers get not modified when editing the exported list.



ID	Default	Deutsch	Englisch
0	Wrong argument	Falsches Argument	Wrong argument
1	Bad format	Ungültiges Format	Bad format
2	Illegal type	Ungültiges Typ	Illegal type
3	Bad result	Ungültiges Ergebnis	Bad result
4	Wrong data type	Ungültiger Datentyp	Wrong data type

Abb. 16 Example of a dynamic text list named 'ErrorList'

Imagine you want to configure a visualization element, which later always when an error is detected should display the corresponding error message.

Your application is processing detected error events which are identified via numeric IDs assigned to an integer variable *ivar_err* .

Provide a dynamic textlist named ' ErrorList' where the error message texts for error IDs "0" to "4" are defined in languages 'Deutsch' and 'Englisch' and 'Default':

Example 'ErrorList'

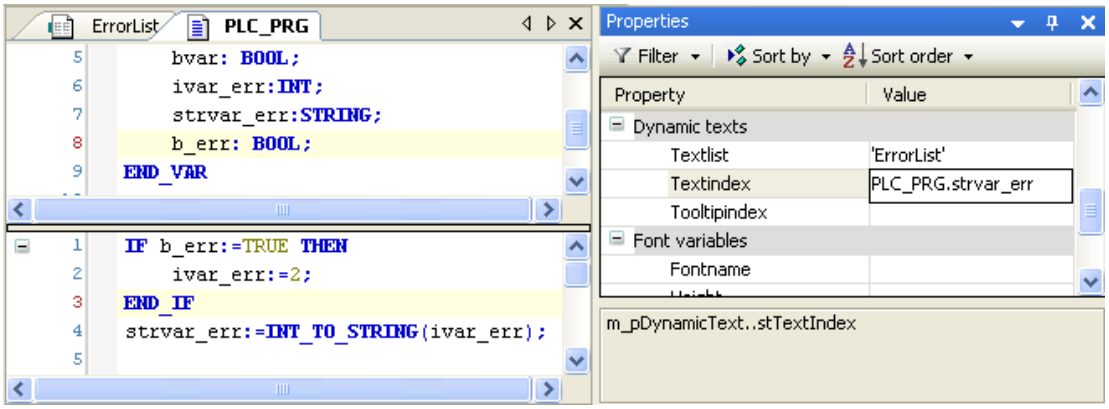
ErrorList

ID	Default	Deutsch	Englisch
0	This is Error 0. Do the following....	Fehler 0. Führen Sie folge...	Error 0. Do the ....
1	This is Error 1. Close....	Fehler 1. Schließen Sie...	Error 1. Close the...
2	This is Error 2. Perform a...	Fehler 2. Führen Sie einen...	Error 2. Perform....
3	This is Error 3. Try to...	Fehler 3. Versuchen Sie....	
4	This is Error 4. Start.....	Fehler 4. Starten Sie	

For using the error IDs in the visualization configuration define a STRING variable, for example strvar_err. To assign the integer value of ivar_err to strvar_err use strvar_err:=INT_TO_STRING(ivar_err);)

strvar_err now can be entered as 'Textindex' parameter in the configuration of the 'Dynamic texts' properties of a visualization element. So this element will display the appropriate error message in online mode.

Example, Project variables processing the error ID, Configuration of a visualization element (Properties dialog), which should display the error message.

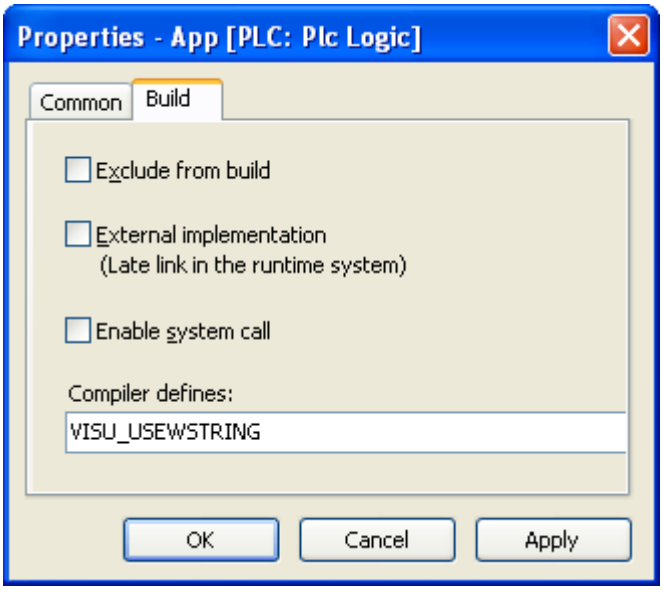


Creating a text list

- To create a text list for dynamic texts, add a 'Text List' object to the project via command "Add Object' / Textlist" . When you have specified a list name and confirmed the 'Add Textlist' dialog, the new list will be inserted in the POUs window or - if currently an application is selected - in the Devices window below the application object - and automatically be opened in the text list editor window.
- To create text list for static texts, either assign a text in Property 'Text' in category 'Texts' of a visualization object to get the text ist 'GlobalTextList' created automatically, or add it explicitly by command Add Global Text List.
- To open an existing text list for editing, use command 'Edit Object' or perform a double-click on the object entry. See ABOVE for how a text list is structured.
- To edit a field in the list, first select the field by a mouse-click and then perform a further mouse-click or press <Space> to get an edit frame. Enter the desired characters and close the entry by <Return>. You can use the arrow keys to get to the next/previous field.

Support of Unicode format

If the Unicode format should be supported, the respective option must be activated in the Visualization Manager. Further on a special compilation directive for the application must be set: Select the application in the Devices tree, open the 'Properties' dialog, category 'Build': In the Compiler defines field enter "VISU_USEWSTRING". Also notice the necessary export of any language files, described in the following.



Export and Import of Textlists

Static and dynamic text lists can be exported in text- or xml-format. If a language file is needed for a Target-Visualization, this must be provided by an explicit export of the respective text list(s) in text format. Exported files also can be used for adding texts externally, for example by an external translator. However notice that **only files available in text-format (*.txt, *.csv) can be re-imported**.

See the description of the respective commands for details.

The folder in which the export files should be saved is to be specified in the Visualization Project Settings.

Formatting of texts

The texts can contain formatting definitions (%s,%d,...), which i.a. allow to include the current values of variables in a text. For the possible formatting strings see the help page on text in visualization.

When later using a text, the replacement will be done in the following order of steps:

1. The actual text string to be used is searched via list name and ID.

2. If the text contains formatting definitions, those will be replaced by the value of the respective variable.

Subsequent delivery of translated texts

By inserting GlobalTextList.csv (subsequently delivered file) in that directory which is used for loading textfiles, a subsequent integration of translated texts is possible. At startup of the bootproject it will be detected that an additional file is available and the translations will be checked against those in the textlist files. New and modified translations will be applied to the textlist files.

Afterwards the file GlobalTextList.csv will be marked as "loaded". Due to this fact a subsequent supply of texts always only will effect once the startup time of the bootproject.

"List Components" for text input

Via the Visualization Options a text template file can be specified. All texts of column „Default“ of this file will be copied to a list, which will be used for the "List Components“ functionality. As template file a file might be used which has been created before via the Export-command.

Multiple user operations

By use of the source control in »PLC Designer« it is possible that multiple users work simultaneously on the same project. Thereby the following items have to be kept in mind:

- If a static text is modified in a visualization element, the visualization and evetually the GlobalTextList as well must have write permission (see GlobalTextList). If the GlobalTextList must not be written to, the texts in the visualizations should not be modified. Otherwise it may happen that the Text-Ids do no longer fit the texts within the visualization elements.
- By use of the command „Check Visualization-Text-Ids“ such errors may be detected in the visualizations.
- By use of the command „Update Visualization-Text-Ids“ these errors may be resolved automatically. Therefore all affected visualizations as well as the GlobalTextList must have write permission.

Detected error cases in a dispatch may provoke misplaced texts in visualizations. If no error cases are reported for a project, the textfile may also be translated later on.

Use of textlists for changing language in visualizations

If an appropriate textlist is available, that is a textlist defining several language versions for a text, then the language used for the texts in a visualization can be switched in online mode by an input on a visualization element. For this purpose the "Dynamic Texts" properties of the element must specify the textlist to be used, and an "OnMouse.." input action "Change the language" must be configured specifying the language which should be used after the mouse action has been performed. The language must be specified with exactly that string which is shown in the column header of the respective textlist.

3.7.15 Image Pool

Note: This functionality is only available if supported by the currently used feature set.

Image pools are tables defining the file path, a preview and a string ID for each image. By specifying the ID and (for unique accessing) additionally the image file name the image can be referenced e.g. when being inserted in a visualization (configuration of the properties of an image element, see below).

Note: It is recommended to reduce the size of an image file as far as possible before adding it to an image pool. Otherwise the project size and the loading and storing efforts of visualization applications, including images, might get unnecessarily big.

Structure of an image pool:

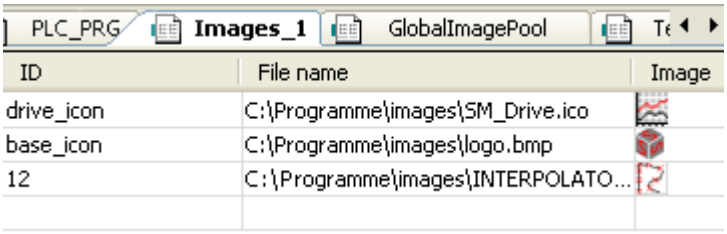


Abb. 17 Example of an image pool

ID: String ID, e.g. "logo", "y_icon", "2"); an unique referencing of an image is achieved by the combination of image list name and ID (e.g. "List1.basic_logo")

File name: Path of the image file (e.g. "C:\programs\images\logo.bmp")

Image: Preview of the image

Creating and editing an image pool

A project can contain several image pools.

If no pool is yet available in a project, then - as soon as you add the first image element and enter an ID ("static ID") for the respective image in the visual element properties - an image pool with the default name "**GlobalImagePool**" will be created automatically and an entry for the image will be inserted. GlobalImagePool is a global pool which always will be searched first when an image file is to be used. Besides this also other, individually named pools can be used.

Manual creation of image pools is done like follows: GlobalImagePool via the Create GlobalImagePool command, the others via the Add Object command. The pool objects can be inserted in the POUs view or directly assigned to an Application object in the Devices view. In the 'Add object' dialog choose object type 'Image pool' and define a name for the pool.

To add an image manually to a pool, select the **ID** field of the first empty line in the pool table, use the <Space>-key to open an edit frame and enter an ID (string). If the entered ID is already used within the table, automatically a digit will be appended, starting with 0 and being counted up for each further "copy" of the ID.

Then set the cursor to the **File name** field and via the <Space>-key and then by the ... button open the **Select image** dialog, where you can specify the path of the desired image file.

Using images which are managed in image pools

Regard the following if the ID of the image to be used is specified in multiple image pools:

- **Search Order**: If you choose an image managed in the GlobalImagePool you do not need to specify the pool name. The search order for images corresponds to that for global variables: 1. GlobalImagePool, 2. Image pools assigned to the currently active application, 3. Image pools in POUs window besides GlobalImagePool, 4. Image pools in libraries
- **Unique accessing**: You can directly call the desired image by adding the image pool name before the ID according to syntax "<pool name>.<image ID>" (example, see figure above: "imagepool1.drive_icon")

1. Using an image in a Visualization element of type "image":

When inserting an image element in a visualization, you can define it to be a static image or a dynamic image, which can be changed in online mode according to the value of a project variable:

Static images:

In the configuration of the element (property "**Static ID**") enter the image ID resp. image pool name + image ID. Regard in this context the remarks on Search Order and Unique accessing)

Dynamic images:

In the configuration of the element (property "**Bitmap ID variable**") enter the variable which defines the ID, e.g. "PLC_PRG.imagevar".

2. Using an image for the Visualization background:

In the Background definition of a visualization you can define an image to be displayed as visualization background. The image file can be specified as described above for a visualization element by the name of the image pool and the image file name.

3.7.16 Visualization

For information on visualizations in »PLC Designer« see Visualization Editor.

3.7.17 POUs for implicit checks

Below an application you might add special POUs, which must be available there, if the implicitly provided check functionality for array and range boundaries, divisions by zero and pointers during runtime should be used.

For this purpose the "Add Object" menu in category "POUs for implicit checks" the following functions:

- CheckBounds
- CheckDivInt
- CheckDivLInt
- CheckDivReal
- CheckDivLreal
- CheckRange
- CheckRangeUnsigned
- CheckPointer

After having inserted a check POU, it will be opened in the editor corresponding to the implementation language selected. A default implementation that might be adapted to your requirements is available in the ST editor.

After having inserted a certain check POU, the option will not be available in the dialog any longer, thus avoiding a double insertion. If all types of check POUs have already been added below the application, the 'Add Object' dialog at all will not provide the 'POUs for implicit checks' category any longer.

Note: Do not modify the declaration part of an implicit check function, in order to maintain the check functionality.

3.8 Device, Device tree

In the Devices window ("**device tree**") the hardware can be mapped on which the application is to run.

Each "**device**" object represents a specific (target) hardware object. Examples: controller, field bus node, bus coupler, drive, I/O-module, monitor.

Each device is defined by a device description and must be installed on the local system in order to be available for inserting in the devices tree (see below). The device description file defines the properties of a device concerning configurability, programmability and possible connections to other devices.

In the device tree however not only device objects are managed, but all objects which are needed to run an application on a device (controller, PLC); thus also the "Application" objects as well as "Task Configuration" and "Task" objects. But also pure programming objects like particular POUs, Global Variable lists and Library Manager can - instead of being managed as project-globally instantiable units in the POUs window - be managed ONLY in the device tree and in this case are only available for exactly "their" application or "child applications" of their application.

Regard the possibility to run the active application on a "**simulation device**" which is per default automatically available within the programming system. So no real target device is needed to test the online behaviour of an application. When you switch to simulation mode, the »PLC Designer« SP V3 entry in the device tree will be displayed in italic letters and you can login with the application.

See in the following some Generals on the device tree and information on the installation and the arranging of the objects.

For information on the conversion of device references when opening projects of another format please see Open Project.

Generals:

- The **root node** of the tree always is a symbolic node entry: *<projectname>*.
- The trees of the "**PLC Configuration**" and "**Task Configuration**", which in »PLC Designer« V2.3 are completely handled in separate windows, are integrated in the devices tree. The PLC configuration is defined by the topological arrangement of the devices in the device tree. Just the configuration of the particular device resp. task parameters is done in corresponding editor dialogs.
- Thus the hardware structure will be mapped and represented within the devices tree by the corresponding arrangement of "device" objects, allowing to set up a complex heterogeneous system of multiple networked controllers and underlying field busses.
- See below for the "rules" for inserting objects in the device tree.

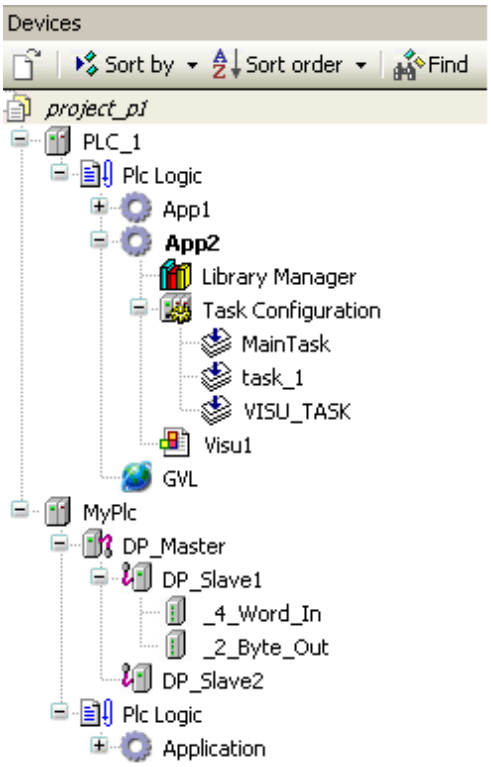


Abb. 18 Device tree, example

- Each entry in the device tree shows the symbol, the **symbolic name** (editable) and the **device type** (=device name as provided by the device description)
- A device is programmable or just parameterizable. The type of the device determines the possible position within the resources tree and also which further resources can be inserted below the device. **Programmable devices** are indicated by an additional "**Plc Logic**" node inserted automatically below the device entry. Below this node the objects needed for programming the device (applications, text lists etc.), as well as functional objects like e.g. a Parameter Manager, can be inserted. Pure **parameterizable devices** can not get assigned such programming objects, however the values of the device parameters might be edited in the parameter dialog of the device editor. Regard that the programmability of a device is a property which can change (device description) without the need of reinserting the device.
- Within a single project one or several programmable devices - regardless of manufacturer or type - can be configured ("**Multi-Resource**", "**Multi-Device**", Networking).
- The **configuration** of a device concerning communication, parameters, I/O Mapping is done in the **Device dialog (Device Editor)**, which can be opened by a double-click on the device entry (see Device Editor for a description).
- **In online mode** an icon at the beginning of a device entry indicates whether the device currently is connected or not connected . Additional diagnostics information is provided in the device dialog in category "status".

Installing of devices on the local system

- **Installation** and uninstalling of devices can be done in the Device Repository dialog. The installation bases on **device description files** in xml-format. The default extension for a valid device description file is ***.devdesc.xml**. However also bus-specific description files like *.gsd-files (Profibus), *.eds-files (CANbus, ASi) etc. can be installed via the Device Repository dialog. The specific »PLC Designer« V2.3 PLC configuration files ***.cfg** can be used if an appropriate additional info-file ***.info.cfg**, describing the respective set of configuration files, is provided.

Arranging and configuring objects in the devices tree, How to do, Rules:

Currently:

- To **add an object** use command Add Device. The device types which can be inserted depend on the currently selected object within the device tree. For example: modules for a DP Profibus slave cannot be inserted without having inserted an appropriate slave device before; no applications can be inserted below non-programmable devices. Further on only devices correctly installed on the local system and matching the current position in the tree will be available for insertion.
- **Re-positioning** of objects is possible via the standard commands of category "Clipboard" (Cut, Copy, Paste, Delete) or by drawing the selected object with the mouse while the mouse-button (plus <Ctrl> for copying) is pressed. Regard for the Paste command: In case the object to be pasted can be inserted below or above the currently selected entry, the **Select Paste Position** dialog will open where you have to define the insert position.
- A device, already inserted in the device tree, can be replaced by another version of the same device type or by a device of another type (**device update**). In doing so, a configuration tree indented below the respective device will be maintained as far as possible.
- Only "device" objects can be positioned on the level directly below the **root node** *<projectname>*. If you choose another object type from the 'Add Object' dialog, like e.g. a "Text list" object, this will be added to the POUs window! If currently no entry is selected, e.g. if you click with the mouse in the empty space of the devices view, this equals to having selected the root node entry.
- A device will be inserted as a node in the tree. If defined in the device description file, **automatically sub-nodes** might be inserted. A sub-node again might be a programmable device.
- **Below a "device" object** further devices might be inserted, if those are installed on the local system and thus available in the 'Add Object' resp. 'Add Device' dialog. The **sorting** of device objects within the tree from up to down: On a particular tree level first always the programmable devices (Plc Logic) are arranged, followed by any further devices, each sorted alphabetically.

- An "**application**" object only can be inserted below a "Plc Logic" node (symbolic node for programmable devices). If several applications resp. an application tree are inserted, the application **names must be unique** within the scope of the device. The applications oBelow each application further programming objects can be inserted like e.g. DUT, GVL or visualization objects. A task configuration **must** be inserted below each application. In this task configuration the respective program calls must be defined (instances of POUs from the POUs window or application specific POUs). If several applications are inserted directly below a device, it is to be defined in the I/O-Mapping of the device, which of these applications should be actually regarded.
- **Hierarchical arrangement of applications, Scopes**:
  An application can be inserted below another application: Let's name the resulting construct "Father application" - "Child application". The child application can use the objects of the father application, not however vice versa! Background: A child application always should be removable or replaceable without concerning the father application. See the following figure: 'Application' is a father application for 'App1', that again is a father application for 'App2':

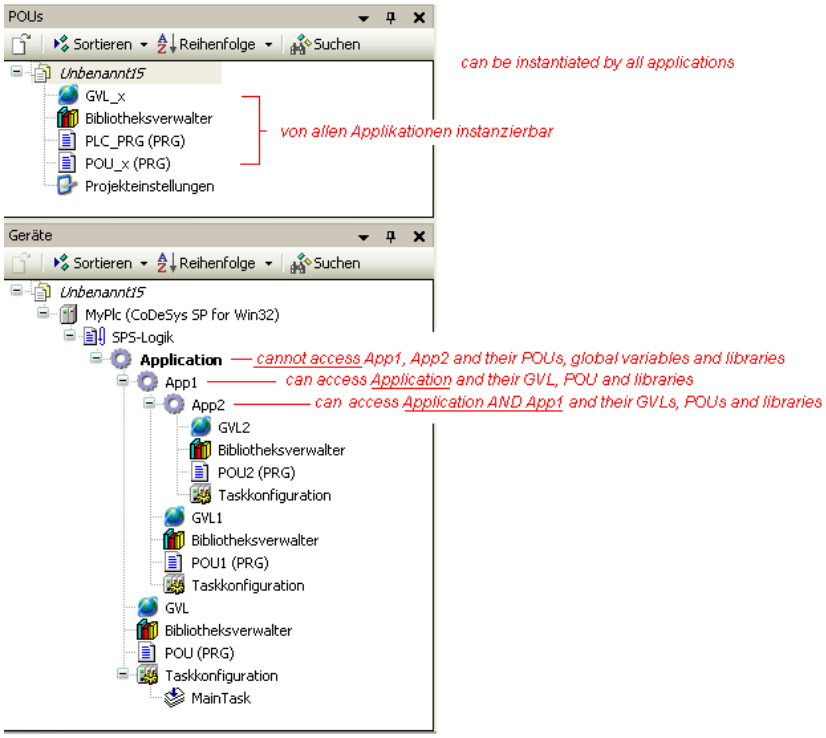


Abb. 19 "Father"- and "child applications"

- **Network scan of the current hardware**
  As an assistance for setting up the PLC configuration in the device tree a scan functionality is provided by the standard device editors. The current hardware structure can be read and displayed in a dialog, allowing the user to direct take over the desired modules to the projects' device tree in the project. To this see command 'Scan For Devices'.

3.9 Application

- An "application" is a set of objects which are needed for running a particular **instance of the PLC program** on a certain hardware device (PLC, controller). For this purpose i.a. "independent" objects, managed in the POUs view, are instantiated and assigned to a device in the Devices view. This meets the mind of object orientated programming. However also purely application-specific POUs can be used.
- An application is represented by an " **Application" object** in the Devices tree , insertable below a Plc Logic (programmable) device node. Below an application entry the objects defining the applications "resource set" are to be inserted.
- All applications below the same device must have **unique names**.
- Before **inserting** an application object via the Add Object command select the respective programmable device object entry in the Devices view, otherwise you will not get offered object type "Application" in the 'Add Object' menu. In the 'Add Application' dialog enter a desired name for the application and press button 'Open'. The application will be added to the Devices tree below the device entry.
- An essential part of each application is the **Task Configuration** controlling the run of a program (POU instances or application-specific POUs). Additionally it might have assigned resource objects like Global Variables Lists, Libraries etc., which - in contrast to those managed in the POUs window - only can be used by the particular application and its "childs", see Device, Device tree for the rules.
- Within a project **several applications can be defined for the same** device, i.e. can be assigned to and run on the same controller. Thereby **nested** arrangement of the applications is possible ("father"- and "child applications"). In this case the objects assigned to the "father" can be used by the "child", not however vice versa, see Device, Device tree for the rules.

- If several applications are available below a device, for the **I/O-Mapping** of that device you can define which of the applications should be regarded.

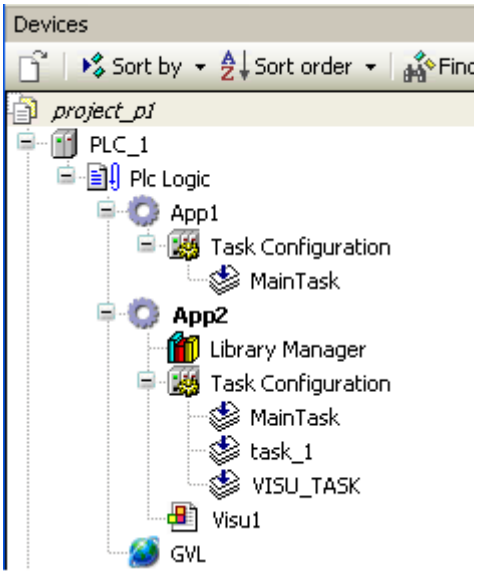


Abb. 20 Example: Two applications for one device

- Please note that the application you want to work on, must be set as "**active application**", see "App2" in the picture shown above.
- An application can get assigned special **properties** in the (object) properties dialog; for example dynamic memory allocation can be activated.
- When going to log in with an application on a target device (PLC or simulation target), it will be **checked** which applications currently are on the PLC and whether the application parameters on the PLC are matching those in the project configuration. Appropriate messages will indicate mismatches resp. you will get the possibility to **delete** applications on the PLC. See the description of the Login command for more details.
- Regard the **'Application' subdialog** of the Device Editor, where you can scan a device for the currently loaded application projects and where you can remove those from the target system. Here also applications might be listed, which are not represented by a separate object in the device tree, like e.g. "<application>__symbols.app", which contains a symbol list possibly generated for application <applicaton>.

3.10 Task Configuration

The Task Configuration defines one or several tasks for controlling the processing of an application program.

It is an essential resource object for an application and must be inserted in the Devices tree below an application object. A task can call an application-specific program POU, which is only available in the device tree below the application, as well as a program which is managed in the POUs window. In the latter case the project-globally available program will be instantiated by the application.

A task configuration can be edited in the Task Editor, the available options being target-specific.

In online mode the Task Editor provides a monitoring view giving information on cycles, cycle times and task status.

Important notes for multitasking systems:

On some systems real preemptive multitasking is realized (e.g. »PLC Designer« SP). In this case the following must be regarded:

Like in »PLC Designer« V2.3 all tasks share one process map. Reason: An own process map for each task would charge the performance. So however the process map always can only be consistent to one task. Hence the user, when creating a project, explicitly must take care that in case of conflicts the input data will be copied to a save area, the same problems have to be regarded for the outputs. For example, modules of the SysSem.library could be used to solve the synchronization problems.

Also when accessing other global objects (global variables, modules), consistency problems might occur as soon as the size of the objects exceeds the data width of the processor (structures or arrays forming a logical unit). Also here the modules of the SysSem.library might be used to solve the problems.

3.11 Communication

Within this chapter you will find information on the following topics:

- Configuration of a PLC
- Network topology
- Addressing and Routing
- Structure of addresses
- Data Server
- Network Variables

3.11.1 PLC Configuration

The "PLC Configuration" in »PLC Designer« maps the target system, i.e. the controller hardware in order to make the I/Os and parameters of the controller and field bus devices accessible by the application. Further on it allows to monitor the available device parameters.

The tree of the "**PLC Configuration**", which in »PLC Designer« V2.3 is handled in own editor windows, in V3.x is integrated in the device tree, where also the other objects necessary for running an application on a target system, are arranged. The standard device editors support the mapping of the current hardware structure into the device tree by a scan functionality.

The assignment of the I/Os of the PLC to project variables is done either via the "AT declaration" of a variable in the declaration editor or in the mapping dialog of the device editor. The device editor provides dialogs for the configuration of a device.

If below a device, which is positioned top-level in the device tree, multiple applications are inserted, then you can define in the I/O Mapping dialog of the device, which of those should be regarded for the mapping. When a new PLC gets inserted in the device tree, primarily per default the application coming with the PLC device will be defined as "mapping application".

3.11.2 Network topology

The aim is to create a system able to largely configure itself (address assignment), to support transparently any communication media and to route packets between different networks. The routing mechanism should be simple enough that any node in the network, i.e. even nodes with low resources, are able to reroute packets. In consequence, large routing tables, complex calculations or requests during runtime should be avoided.

A control network should be configured hierarchically, i.e. each node has one parent node and an arbitrary number of children. A node without parent is referred to as top-level node. Cycles are not permitted, i.e. a control network has a tree structure.

Parent-child relationships arise from the specification of network segments. A network segment corresponds e.g. to a local Ethernet or a serial point-to-point connection. We distinct between the main network (mainnet) and the sub-networks (subnet). Each node has at most one main network, wherein it expects its parent, as far as a parent exists. For each node an arbitrary number of subnets can be configured. The node acts as parent for all of them.

If a network segment had been defined simultaneously as subnet of several nodes, the network would have several parents. Though, the resulting configuration will be invalid, as each network segment is allowed to have one single parent only.

3.11.3 Addressing and Routing

Addressing maps the topology of the control network to unique addresses. A **node address** is built up hierarchically:

For each network connection a **local address** identifying the node uniquely within its respective local network is allocated by the relevant block driver. For the entire node address this local address is preceded by the **subnet index** the local network is assigned to by the parent. Furthermore it must be preceded by the node address of its parent.

Thereby, the length of the subnet index (in bit) is determined by the device, whereas the length of the local address is determined by the network type.

A node without main network is a top-level node with address 0. A node with a main network that does not contain a parent is also a top-level node and will be assigned to its local address in the main network.

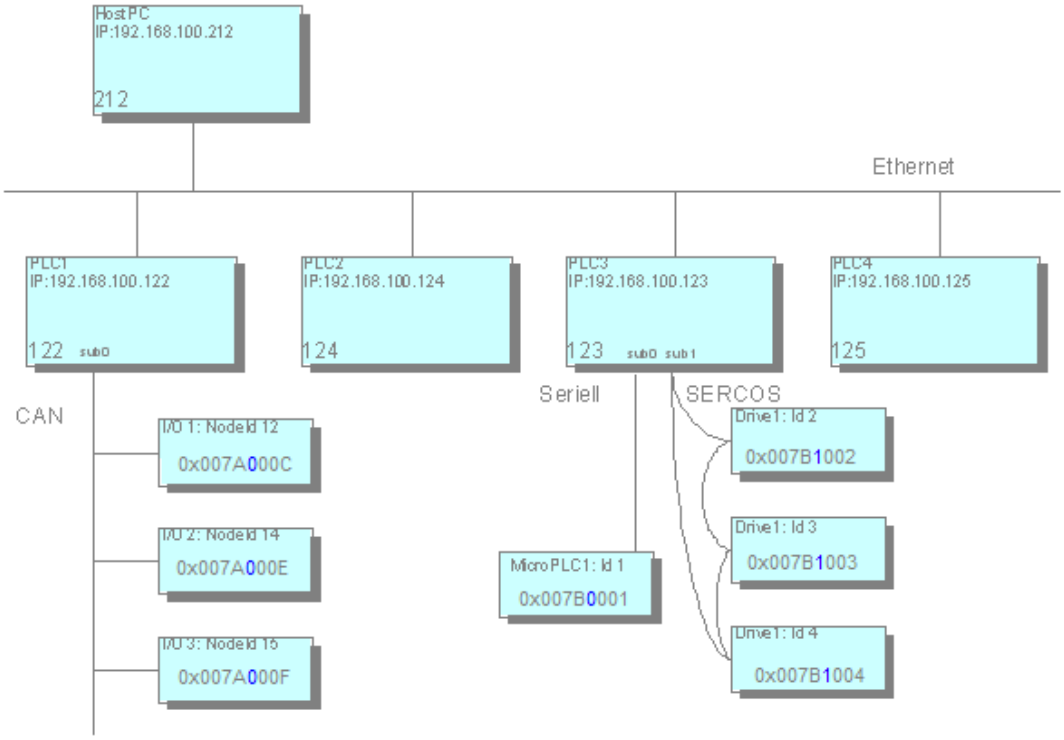


Abb. 21 Example - Main net and sub nets

In the example the addresses of the child nodes are given in hexadecimal representation. The first 4 digits represent the address of the particular parent within the main net, e.g. 0x007A=122 for PLC1. The next byte (displayed in blue) is reserved for the subnet index and this is followed by the local address, e.g. C=12 for NodeId 12.

Due to the structuring of the address the routing algorithm can be kept relatively lean. For example, no routing tables are necessary. Information is required locally: on the own address and on the address of the parent node.

Thereon a node may handle data packets in a correct way:

- If the target address equals the address of the current node, it is determined as receiver.
- If the target address starts with the address of the current node, the packet is intended for a child or descendant of the node and has to be forwarded.
- Else, the receiver is no descendant of the current node. The packet has to be forwarded to the own parent.

Relative addressing is a special feature. Relative addresses do not contain the node number of the receiver node, but directly describe the path from the sender to the receiver. The principle is similar to a relative path in the file system: The address is comprised of the number of steps the packet has to move “up”, i.e. to the next respective parent, and the subsequent path down to the target node.

The advantage of relative addressing is, that two nodes within the same subtree are able to continue the communication when the entire subtree is moved to another position within the overall control network. While the absolute node addresses will change due to such a relocation, the relative addresses are still valid.

Determination of addresses

The knowledge about the own address demands from a node the knowledge either on the address of the parent or on being a top-level node. For this purpose a node will send an address determination message as broadcast to its main network during boot-up. As long as this message is not responded, the node considers itself to be a top-level node, although it will continue to try and detect a parent node. A parent node will respond by an address notification. Thereon the node will complete its own address and pass it to the subnets.

Address determination can be executed at bootup or on request of the programming PC. Not yet implemented: Once the address of a node is frozen, address determination can no longer be executed.

3.11.4 Structure of addresses

Below you find a detail description on the structure of the following address types:

- Network adresses
- Node addresses
- Absolute and relative addresses
- Broadcast addresses

Network addresses

Network addresses represent a mapping of addresses of a network type (e.g. IP addresses) to logical addresses within a control network. This mapping is handled by the respective block driver. Within an EtherNet with Class C IP addresses, the first 3 bytes of the IP address are the same for all network devices. Therefore, the last 8 bits of the IP address suffice as network address, since they allow unambiguous mapping between the two addresses at the block driver.

A node has separate network addresses for each network connection. Different network connections may have the same network address, since this address has to be unique only locally for each network connection.

Terminology: In general, the network address of a node without statement of the network connection refers to the network address in the main network.

The length of a network address is specified in bits and can be chosen by the block driver as required. Within a network segment the same length must be used for all nodes. A network address is represented as an array of bytes with the following coding:

• Length of the network address: n bits

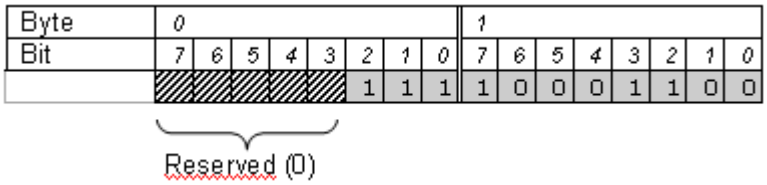
• Required bytes: b = (n + 7) DIV 8

• The (n MOD 8) lowest-order bits of the first byte and all remaining (n DIV 8) bytes are used for the network address.

Example - Network address:

Length: 11 Bit

Address 111 1000 1100



Node addresses

The node address indicates the absolute address of a node within a control network and therefore is unique within the whole “tree”. The address consists of up to 15 address components, each consisting of two bytes. The lower a node is located within the network hierarchy, the longer is its address.

The node address is comprised of the partial addresses of all predecessors of the node and the node itself. Each partial address consists of one or several address components. The length is therefore always a multiple of two. The partial address of a node is formed from the network address of the node in its main network and the subnet index of the main network in the parent node. The bits required for the subnet index are determined by the router of the parent node. Filler bits are inserted between the subnet index and the network address in order to ensure that the length of the partial address is a multiple of 2 bytes.

Special cases:

- Node has no main network: This means there is no subnet index nor a network address in the main network. In this case the address is set to 0x0000.
- Node with main network but without parent: In this case a subnet index with 0 bit length is assumed. The partial address corresponds to the network address, supplemented by filler bits if required.

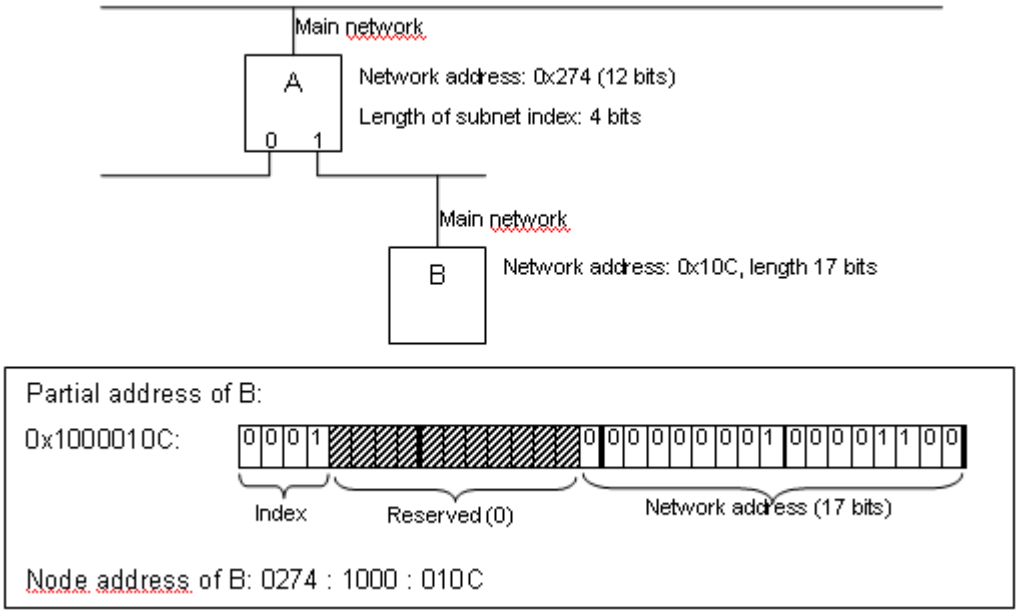


Abb. 22 Example - Node address

The node address representation is always hexadecimal. The individual address components (two bytes in each case) are separated by a ":" (colon). The bytes within a component appear sequentially without separator (see example above). Since this represents a byte array and not a 16-bit value, the components are not displayed in little-endian format. For manually entered addresses missing digits in an address component are filled with leading zeros from the left: "274" = "0274". To improve readability the output should always include the leading zeros.

Absolute and relative addresses

Communication between two nodes can be based on relative or absolute addresses. Absolute addresses are identical to node addresses. Relative addresses specify a path from the sender to the receiver. They consist of an address offset and a descending path to the receiver.

The (negative) address offset describes the number of address components that a packet has to be handed upwards in the tree before it can be handed down again from a common parent. Since nodes can use partial addresses consisting of more than one address component, the number of parent nodes to be passed is always = the address offset. This means that the demarcation between parent nodes is no longer unambiguous, which is why the common initial part of the addresses of the communication partners is used as parent address. Each address component is counted as an upward step, irrespective of the actual parent nodes. Any errors introduced by these assumptions can be detected by the respective parent node and must be handled correctly by the node.

On arrival at the common parent the relative path (an array of address components) is then followed downwards in the normal way.

Formal: The node address of the receiver is formed by removing the last AdressOffset components from the node address of the sender and appending the relative path to the remaining address.

Example:

Within the example a letter will represent an address component, whereas a point will separate the particular nodes. Since a node is allowed to have multiple address components, it is allowed to have multiples letters within the example.

Node A: a.bc.d.ef.g

Node B: a.bc.i.j.kl.m

- Address of the lowest common parent: a.bc
- Relative address from A to B: -4/i.j.kl.m (The number -4 results from the 4 components d, e, f, and g the packet has to be raised)

To ensure the correct working of the routing, the relative address has to be adjusted with each pass through an intermediate node. It is sufficient to adjust the address offset. This is always done by the parent node: If a node receives a packet from one of its subnets, the address offset is increased by the length of the address component of this subnet:

- If the new address offset is < 0 the packet must be forwarded to the parent node.
- If the address offset >= 0 the packet must be forwarded to the child node the local address of which is located at the position described by the address offset within the relative address. First, the address offset must be increased by the length of the local address of the child node to ensure that the node sees a correct address.

A special situation arises when the error described above occurs while determining the common parent. In this case the address offset at the “real” common parent is negative, but the magnitude is greater than the length of the partial address of the subnet the packet originates from. The node must detect this case, calculate the local address of the next child node based on the address of the previous node and the length difference, and adapt the address offset such that the next node will see a correct relative address. Also in this case the address components themselves remain unchanged, only the address offset will be modified.

Broadcast addresses

There are two types of broadcast - global and local ones. A global broadcast is sent to all nodes within a control network. The empty node address (length 0) is reserved for this purpose.

Local broadcasts are sent to all devices of a network segment. For this purpose, all bits of the network address are set to 1. This is possible both in relative and in absolute addresses.

A block driver must be able to handle both broadcast addresses, i.e. empty network addresses and network addresses with all bits set to 1 must be interpreted and sent as broadcast.

3.11.5 Data Server

A Data Server can be added to an application in order to get access to remote data sources. "Remote data sources" in this context means that variables (data items), which are defined and used in other devices, can be used also in the current application.

In contrast to the data exchange via Network Variables (fix variables lists in sender and receiver devices, transmission via broadcasting to all network participants which own the respective variables list) the Data Server establishes defined point-to-point connections between the source(s) and the local application. The data items to be exchanged are defined and it depends on their access flags, whether they get updated in the data source resp. in the current application each time when their value changes each on the other "side".

Using a data server is the faster alternative to providing access to data via a symbol configuration. Unfortunately data handled via e.g. an OPC-Server cannot yet be accessed via a data server. In this case implementing a symbol configuration is still the modus operandi.

3.11.6 Network Variables

If supported by the device (target system), data exchange within a network is possible via Network Variables.

Network Variables must be defined in fix variables lists as well in the sender as in the receiver devices (handled in one or in different projects) and their values will be transmitted via **broadcasting**. Regard that this is basically different from exchanging data via a Data Server, which uses defined point-to-point connections between the local application and remote data source(s).

Network Variables are handled in

- Global Variable Lists (**GVL**) in a sending device (**Sender**) and in a corresponding
- Global Network Variables List (**GNVL**) in a receiving device (**Receiver**). GNVLs are viewed in the Network Variables List Editor.
- The corresponding GVL and GNVL must contain the same variable declarations.
- A Global Variable List (GVL) which should define the Network Variables of a sender, must have special network properties. These are protocol and sending parameters, according to which the variables values will be broadcasted within the network and can be received by all devices which have a corresponding GNVL.
- Regard that the direction of transmission is always **one-way**, from GVL (sender) to GNVL (receiver) ! However each device can define GVLs as well as GNVLs and thus also can act as a sender AND a receiver. A sender GVL can be provided within the same or by another project. So, when creating a GNVL, the sender GVL can be either chosen from a selection list of all available GVLs within the network, or it can be read from an **export file**, which previously has been generated (for example by using the Link to File Properties dialog) from the GVL. Definitely an export file will be needed, if the sender GVL to be used is defined within another project.

As a precondition for network variables exchange the respective **network libraries** must be installed, which e.g. is done automatically for the standard network type UDP as soon as the network properties for a GVL are set.

Example:

In the following example a simple network variables exchange will be established. In the sender device a GVL is created and in the receiver device the corresponding GNVL:

Preparatory work in a standard project, where a sender device "Dev_Sender" and a receiver device "Dev_Receiver" are available in the devices tree:

- Create a POU (program) 'prog_sender' below 'Application' of 'Dev_Sender'.
- In the Task Configuration of this application add task 'Task_S' calling 'prog_sender'.
- Create a POU (program) 'prog_receiver' below 'Application' of 'Dev_Receiver'.
- In the Task Configuration of this application add task 'Task_R' calling 'prog_receiver'.

1. Define global variable list in sender:
Select 'Application' below device 'Dev_Sender'. Via the 'Add Object' dialog **add a global variable list** 'GVL_Sender'. Fill this list with variable definitions:

```
VAR_GLOBAL
iglobvar:INT;
bglobvar:BOOL;
strglobvar:STRING;
END_VAR
```

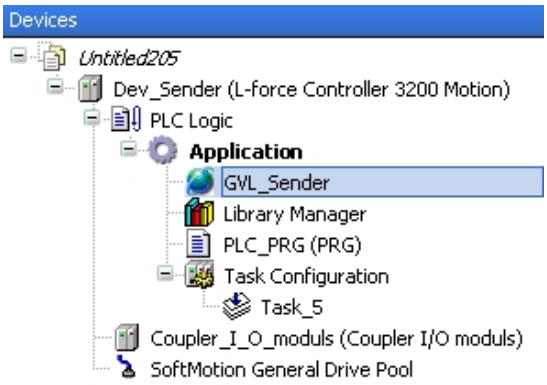


Abb. 23 Insert GVL in sender device

2. Define the network properties of the sender GVL:

Select 'GVL_Sender' in the devices tree and in the context menu choose **Properties** and fill the Network properties as follows:

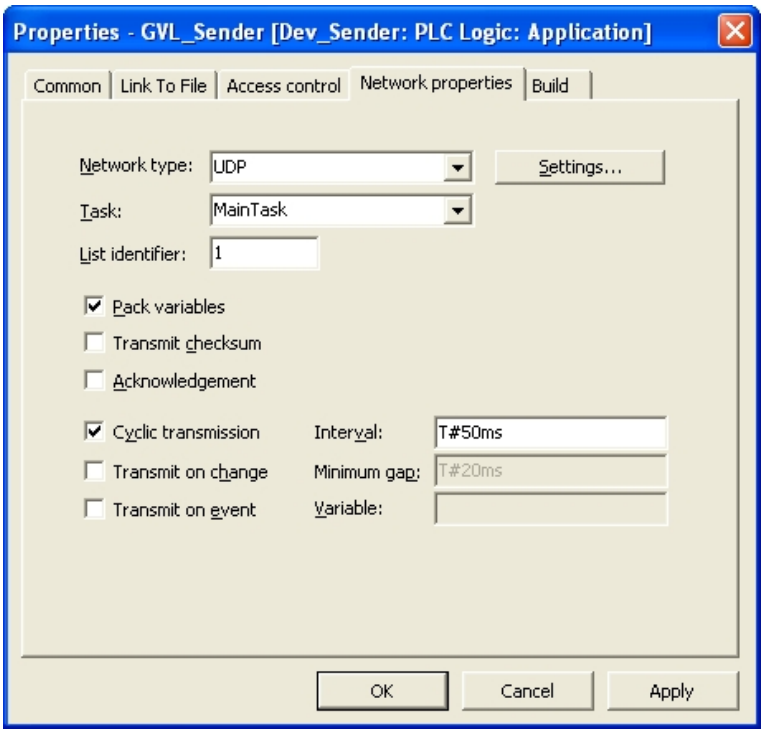


Abb. 24 Set GVL network properties

3. Add a Global Network Variable List in the Receiver:

Select 'Application' below device 'Dev_Receiver'. Open the 'Add Object' dialog to **add a global network variables list** which will be the complement to the above defined GVL:

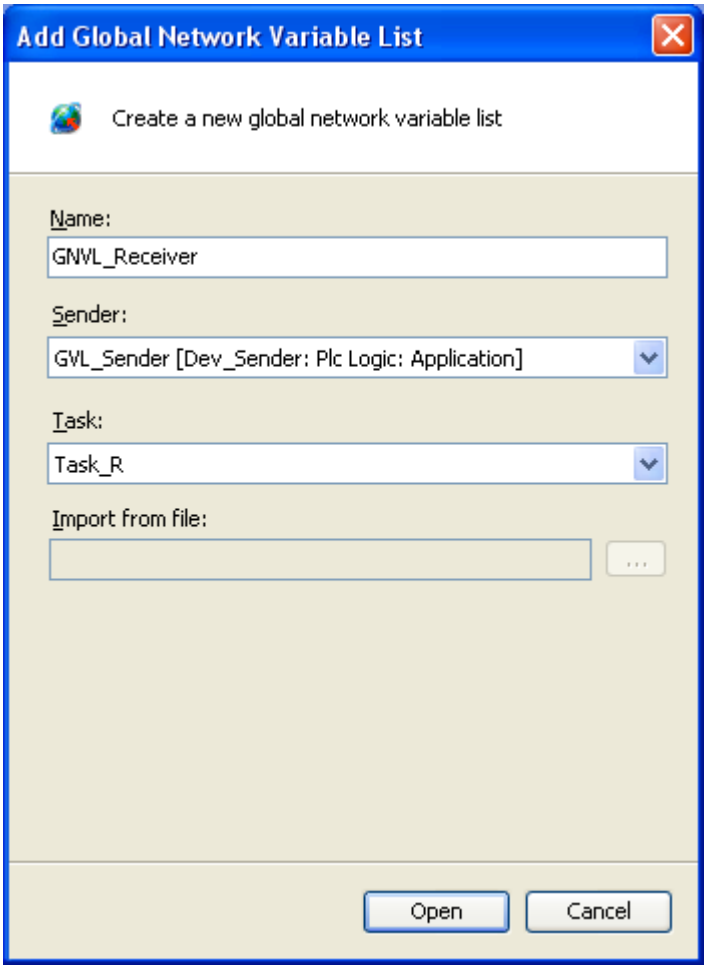


Abb. 25 ***C****reate GNVL list in receiver device*

Enter name "GNVL_Receiver". In field 'Sender' you find a selection list of all GVLs with network properties currently available in the project. For the current example only 'GVL_Sender' will be offered. (Just for information: If you wanted to use a GVL from another project, an export file of this list would have to be available, which then could be imported here via "Import from file")

Close the dialog with OK to get object 'GNVL_Receiver' inserted below 'Application' of 'Device_Receiver'. This GNVL will automatically contain the same variables declarations as 'GVL_Sender':

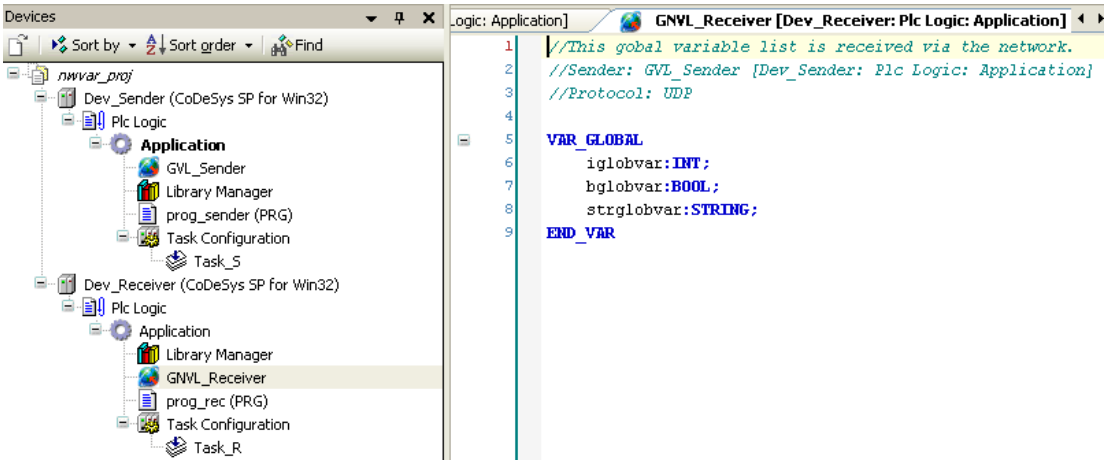


Abb. 26 GNVL_receiver contains same variables as GVL_Sender

4. Optional: View the network settings of the Global Network Variables list:

If necessary, the **Network Settings** of the GNVL list, which have been defined when adding the object, can be viewed and modified. For this purpose select GNVL_Receiver and from the context menu open the Properties dialog and there the 'Network Settings' sub-dialog.

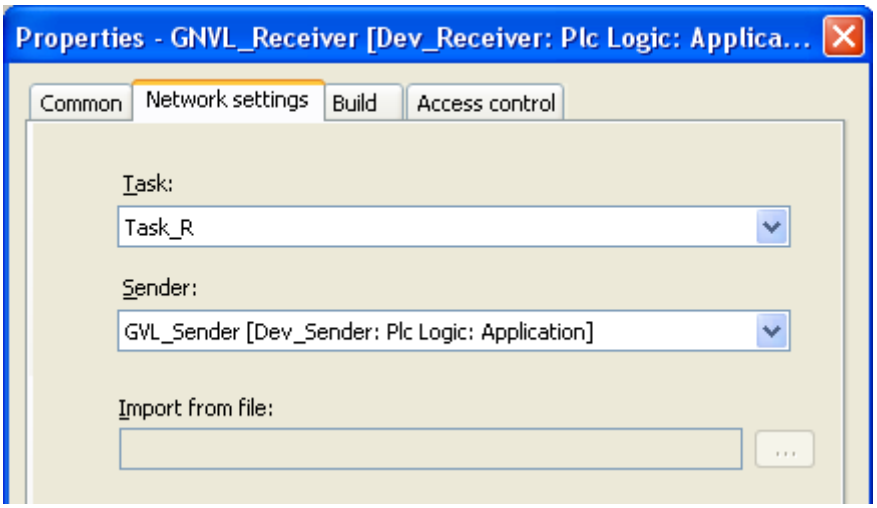
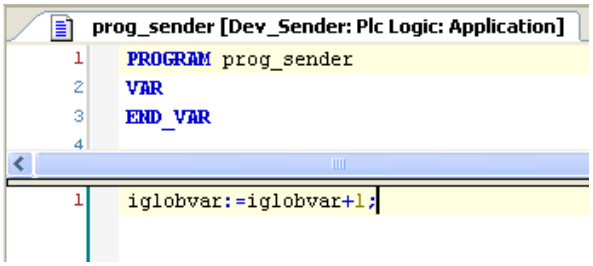


Abb. 27 GMVL network settings

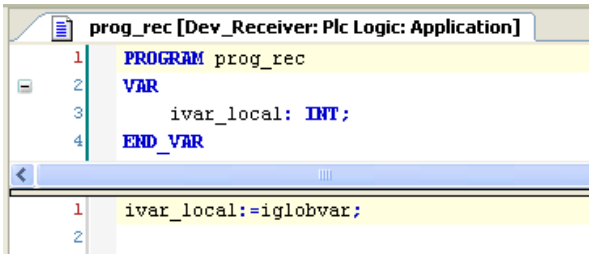
5. Test the network variables exchange:

To check the network variables in online mode, perform the following steps:

- in prog_sender in the sender application enter the following use of variable iglobvar:



- in prog_rec in the receiver application also use variable iglobvar:



- log on with sender and receiver application within the same network and start the applications. Watch, if ivar_local in the receiver gets the values of iglobvar as currently shown in the sender.

Network variables communication between V2.3 and V3 controllers:

A communication via network variables is even possible, if the participating controllers work with applications of different versions (V2.3 <-> V3) of the programming system. However in this case you cannot just use the export/import mechanism for setting up the exactly matching variables lists, which are needed each in the sender and receiver project. This is due to the fact, that the V2.3 and V3 variables' export files (*.exp vs *.gvl) contain different information. If a reading GNVL list is set up in V3, the required network parameters configuration must be provided by a *.gvl-file previously exported by a V3 sender. An export file *.exp created from a V2.3 sender however does not contain this information.

Possible solution for getting network variables exchanged between V2.3 and V3 applications:

1. Recreate the V2.3 NVL in V3 (add a GVL with network properties, containing the same variables declarations as the 2.3 NVL)

2. Export the new GVL to a *.exp-file ("Link to File" properties)

Hint: Set the option "Exclude from build" and you will be able to keep the GVL in the project without getting precompile errors and ambiguous names. Just disable the option to recreate the exp-file again in case any modifications on the GVL are required.

3. Reimport the list, that means create a new GNVL by using the previously generated *.exp-file in order to get an appropriately configured receiver list.

Example

There is a project 23.pro which contains a global variables list GVL_23 with the following declaration:. VAR_GLOBAL trans23: INT; END_VAR.

Variable trans23 should be readable by a V3 application.

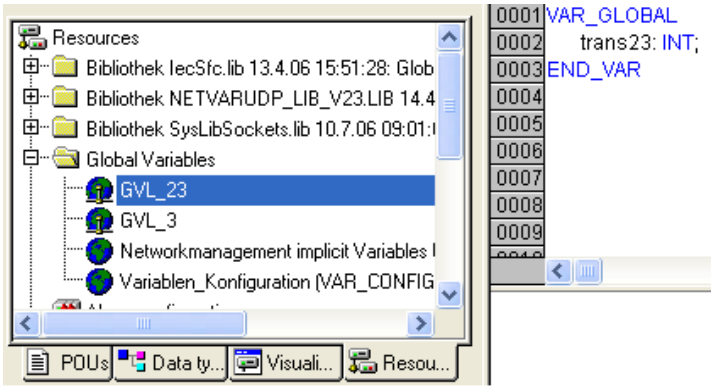


Abb. 28 GVL in V2.3 project

The network properties of GVL_23 are configured as follows:

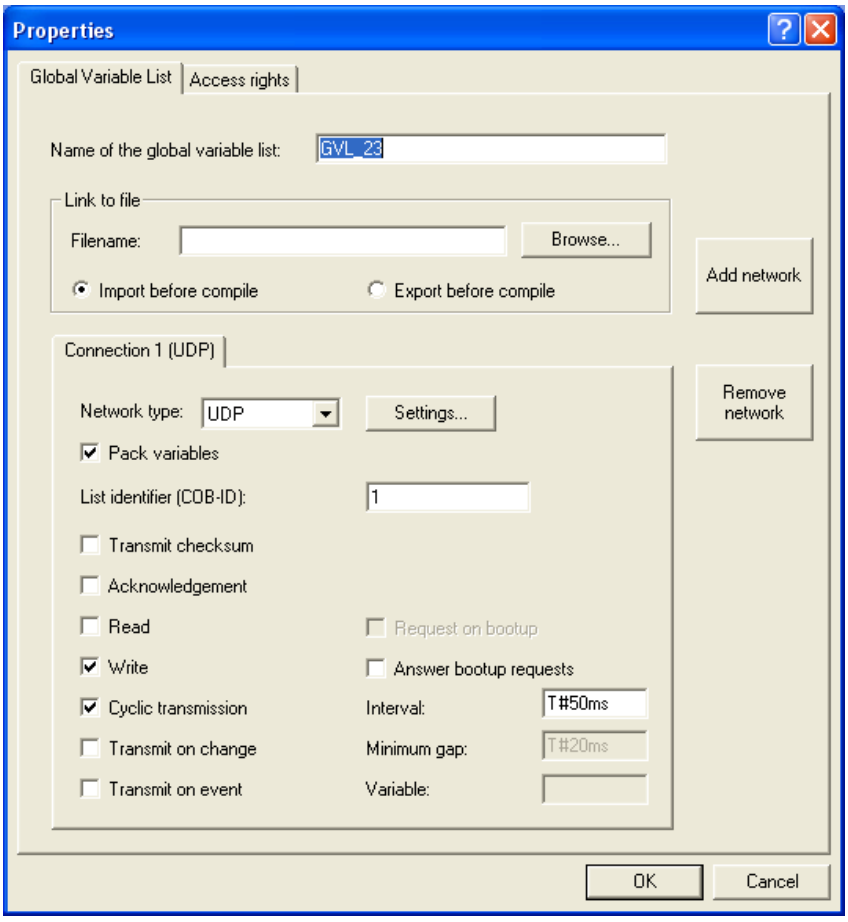


Abb. 29 Properties of GVL_23

If you exported GVL_23 to an *.exp-file, then this file only would contain the declaration VAR_GLOBAL trans23: INT; END_VAR; For this reason it is necessary first to recreate GVL_23 in V3 (see GVL_23 in the figure below):

In a V3 project add a GVL object named GVL_23 to an applicaton and in doing so:

- set the network properties as defined within 23.pro
- in the "Link to File" properties configure a target export file "23.gvl"
- recommendation: set the "Exclude from build" option in order to be able to keep the file on disk for later modifications
- compile the V3 project in order to generate the 23.gvl file (contains variable + configuration settings !)

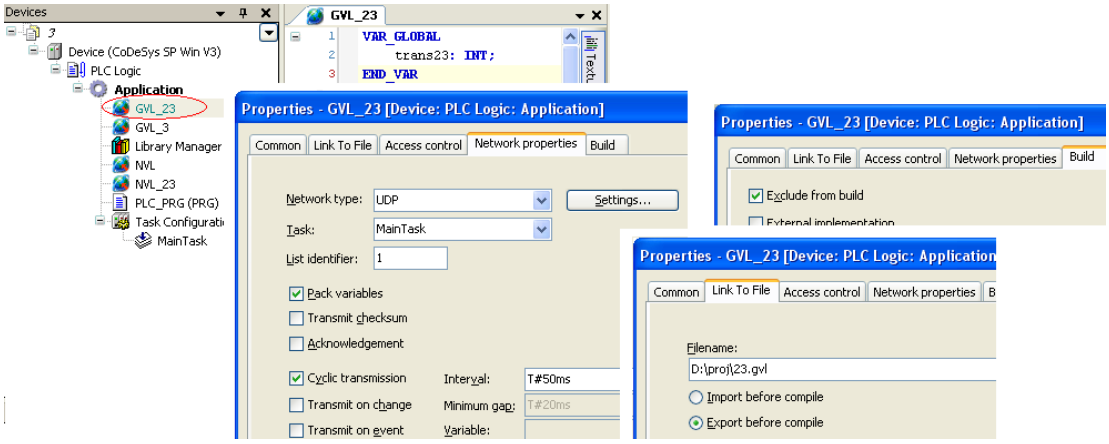


Abb. 30 Recreating GVL in V3

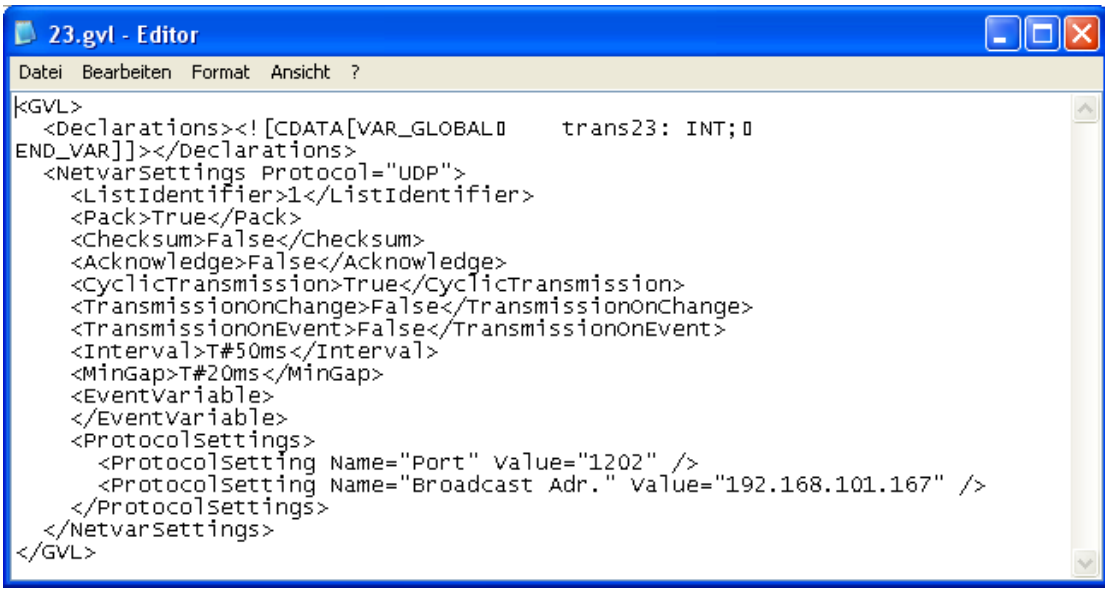


Abb. 31 Resulting export file "23.gvl"

Now add an GNVL object in the V3 project from the 23.gvl (option "import from file") which serves to read variable trans23 from the 2.3 PLC.

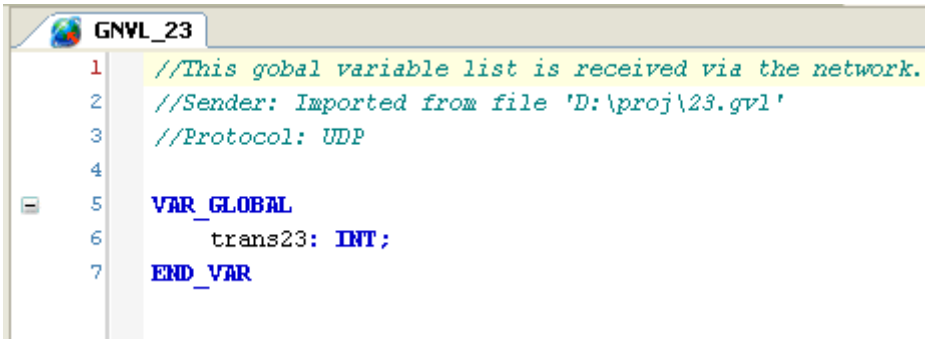


Abb. 32 GNVL in V3 project

Now, if both the V23 project as well as the V3 application are running within the network, the V3 application can read variable trans23 from project 23.pro.

3.12 Code Generation and Online Change

Code Generation, Compile Information

Machine code will not be generated until the application project gets downloaded to the target device (PLC, simulation target). At each download the compile information, containing the code and a reference ID of the loaded application, will be stored in the project directory in a file "**<projectname>.<devicename>.<application ID>.compileinfo**". The compile info will be deleted when the Clean <application> resp. Clean all command is performed.

Regard that no code generation is done when the project is compiled by the 'build' commands. The build process is done to check the project for syntactical errors. Those will be dumped in the Messages window (message category 'Build').

Online Change

WARNING!
Online Change modifies the running application program and does not effect a restart process.
Make sure that the new application code nevertheless will effect the desired behaviour of the system.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

Notes:
1. When an online change is done, the application-specific initializations (homing etc.) will not be executed because the machine keeps its state. For this reason the new program code might not be able to work as desired.
2. Pointer variables keep their values form the last cycle. If there is a pointer on a variable, which has changed its size due to an online change, the value will not be correct any longer. Make sure that pointer variables get re-assigned in each cycle.

If the application project currently running on the controller has been changed in the programming system since it has been downloaded last, just the modified objects of the project will be loaded to the controller while the program keeps running there.

There are two ways to perform an Online Change:

1. As soon as you try to log in again with a modified application (checked via the compileinfo, which has been stored in the project folder during the last download), you will get asked whether you want to do an online change, a download or login without changing.

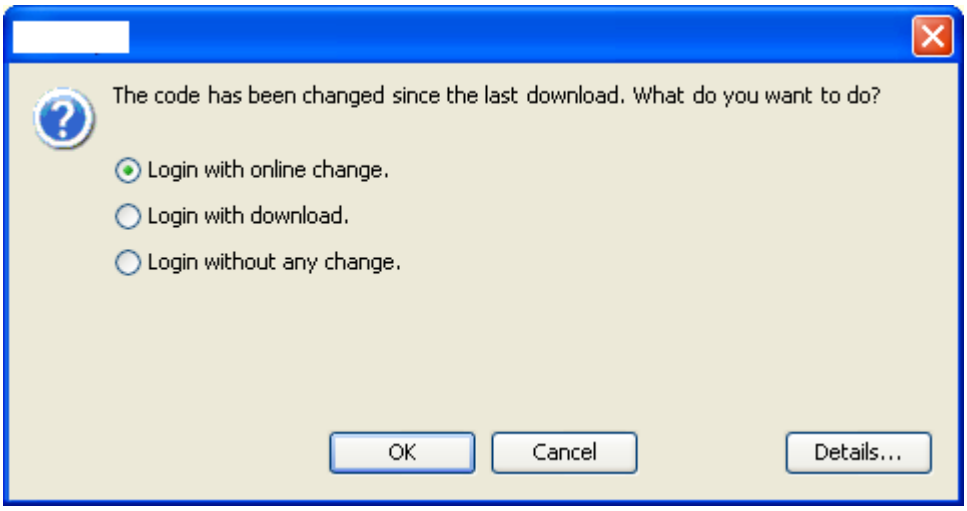


Abb. 33 Login dialog

Login with online change: This option is selected per default. So if you confirm the dialog with **OK**, the modifications will be loaded and immediately shown in the online view (monitoring) of the respective object(s).

Login with download: Activate this option if the application project should be compiled and loaded completely.

Login without any change: Activate this option in order to keep the program running on the controller unchanged. Afterwards an explicit download (Download) might be done, thus loading the complete application project, or at the next re-login you will be asked again whether an online change should be done.

Via the Details button in the login dialog you can get some information (Project name, Last modification, IDE version, Author, Description) on the currently concerned application within the IDE (integrated development version = programming system) in comparison to that currently available on the PLC:

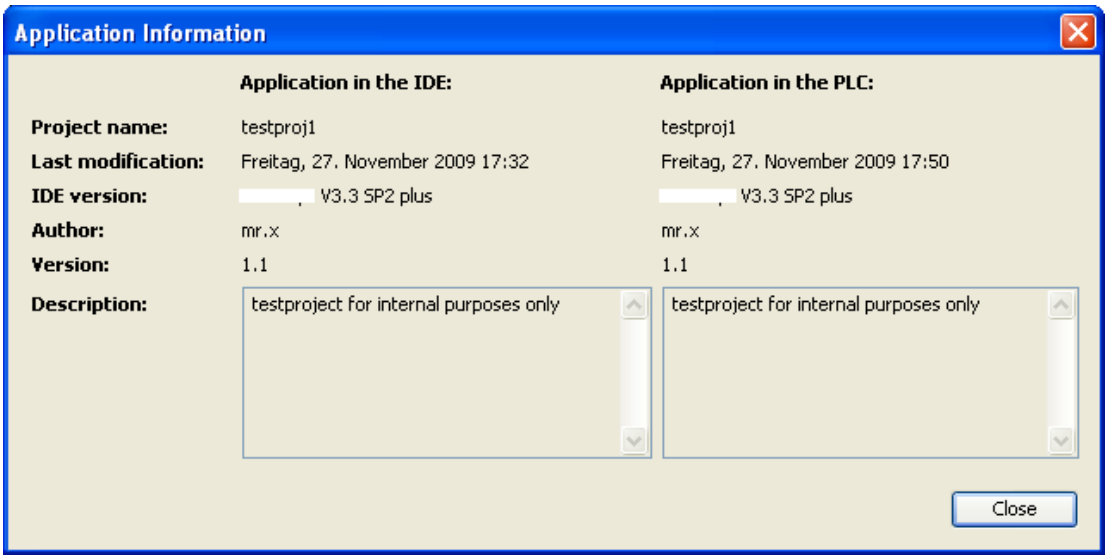


Abb. 34 Application Information dialog

If other applications are currently available on the PLC, you will get additional dialog boxes on how to handle the situation. For a description please see: Log in.

2. By using the command Online Change <application> you can explicitly perform an Online Change on a particular application.

Regard that Online Change of a modified project is no longer possible after a '**Clean**' operation (Clean all, Clean <application>). In this case the information on which objects have been changed since the last download will be deleted. Thus only the complete project can be downloaded.

Consider the following before going to do an Online Change:

Is the changed code without any errors?

Application specific initializations (reference run etc) will not be processed, because the machine is keeping its state. Can the new program code really do without such initializations?

Pointer variables keep their value from the last cycle. If you point on a variable which now has changed its size, the value will not be correct any longer. For this reason you should re-assign pointer variables in each cycle.

If the active step in a function chart gets removed, the chart will remain inactive.

Boot application (boot project)

Currently: At each successful download automatically the active application will be stored in a file "application.app" in the target system folder, thus available as boot application. Command 'Create boot application' allows also in offline mode to save the boot application to a file.

A boot application is the project which will be started automatically when the controller gets started ("booted"). For this purpose the project must be available on the PLC in a file <project name>.app. This file can be created in offline or online mode by command Create boot application (command category 'Online').

3.13 monitoring

In online mode there are various possibilities to display the current values of the watch expressions of an object on the PLC:

- **Inline monitoring** in the implementation editor of an object. For details see the description of the respective editor.
- **Online view of the declaration editor** of an object. For details see the description of the declaration editor.
- **Object-independent watch lists**. For details see the description of the watch views.
- **Trace sampling**. Recording and display of variable values from the PLC. For details see the description of the Trace functionality.
- **Recipes**. User defined set of variables for the purpose of writing and watching these variables on the PLC. See Recipe Management.

3.14 Debugging

To evaluate programming errors you can use the »PLC Designer« debugging functionality in online mode. In this context regard the possibility to check an application in **simulation mode**, i.e. without the need of connecting to a real hardware target device.

Breakpoints can be set at certain positions to force an execution break. Certain conditions, such as which tasks are concerned and in which cycles the breakpoint should be effective, can be set for each breakpoint. Stepping functions are available to get a program executed in controlled steps. At each break the current values of the variables can be examined. A call stack can be viewed for the currently reached step position.

Breakpoints

A **breakpoint**, set in an application program, will cause a break during the execution of the program. The possible breakpoint positions depend on the editor. In each case there is a breakpoint at the end of a POU.

A main facility is the Breakpoints dialog providing an overview on all breakpoints, allowing adding, removing and modifying breakpoints.

New: Conditional breakpoints; the halt at a breakpoint can be made dependant on the number of run through cycles or on which task is currently processed.

Symbols:

- breakpoint enabled
- breakpoint disabled
- halt on breakpoint in online mode

Stepping

Stepping allows a controlled execution of an application program, e.g. for debugging purposes. Basically you step from one instruction to the next one by repeated use of the key <F10>, but also you can step over POUs which are called or ....

...New in contrast to »PLC Designer« V2.3:

- The next statement to be executed can be explicitly defined ('Set next statement').
- The next execution break can be defined simply by placing the cursor there ('Run to cursor').
- 'Step Out' steps back to the previous caller.

Symbols used in text editors:

current step position, indicated by a yellow arrow before the respective line and a yellow shadow behind the concerned operation, example:

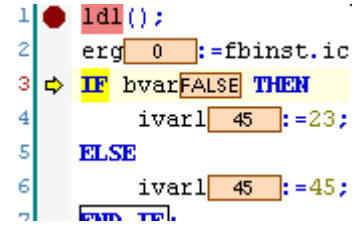


Abb. 35 Step Into, example

3.15 Printing

The currently active editor view can be printed by using the Print commands.

Regard also the possibility to create a "documentation" of severall or all objects of the project, with a defined layout and a table of contents.

3.16 Visualization

For information on visualizations in »PLC Designer« see Visualization Editor.

3.17 Library Management

Libraries can provide functions and function blocks as well as data types, global variables and even visualizations which can be used in the project just like the other POUs and variables which are defined directly within the project.

The default extension for a library file is "**.library***, in contrast to ".lib" used for the files in »PLC Designer« V2.3 and earlier versions.

The management of the libraries in a project is done in the Library Manager, the preceding installation on the system in the Library Repository dialog.

The project functions for local and global search and replace also work for included libraries.

Note: The extent of available library management features depends on the currently used feature set !

See on this page general information on:

- Installation and Including in project
- Referenced Libraries
- Library Versions
- Unique access to library modules or variables
- Creating V3.x libraries, Encoding, Documentation
- »PLC Designer« V2.3 libraries
- Internal and external library modules

Installation and Including in project

- Libraries can be managed on the local system in one or various repositories (folders, locations). A library cannot be included in a project before it has got **installed** in a repository on the local system. The installation is to be done in the **Library Repository Dialog**.
- As a precondition for installation, a library must have got assigned a title, a version info and a company name in its **Project Information**. Optionally a category can be defined which might serve later for sorting in the Library Manager.
- If no **category** assignment is defined in the Project Information, the library automatically will belong to category "Miscellaneous". Further categories might be defined in one or several xml-files ***.libcat.xml** which can be loaded in the Project Information dialog in order to select one of the categories. Please see also below, Creating V3.x libraries.
- The Library Manager is used to **include libraries** in a project. In a "standard project" a Library Manager object by default is automatically assigned to the default device. However a Library Manager object can also be added explicitly in the Devices or POUs view window. This is to be done like for other objects with the Add Object dialog. Libraries referenced in other libraries by default are also displayed in the Library Manager, however also "hidden" libraries are possible, see below, Referenced Libraries, "When creating...", 2..
- If the "**.library***" file is available (and not only its compiled version "*.compiled-library"), the POUs of the related library may be opened by a double click on their referencing within the library manager.
- If a library module is called by an application, all libraries and repositories will be searched in that order which is defined in the Library Repository dialog. See below for unique accessing.

Referenced Libraries

- Libraries can include further libraries (referenced libraries), whereby the nesting can have any desired depth. When you add a library including other libraries in the library manager, the referenced libraries will be added automatically too.
- When creating a library project which includes other libraries, in the Properties of each referenced library it can be defined, how it should behave later when getting inserted in a project with its "father" library:
- 1. Its pure visibility in the Library Manager, indented below the "father" library, can be deactivated. Thus **"hidden" libraries** can be available in a project.
- 2. If you are just creating a pure **"container" library**, that is a library projects which is not defining own modules, but only is referencing other libraries, the access on the modules of these libraries can be simplified. A "container" library is created because you want to include a complete set of libraries in a project at the same time by just including the container library. In this case it might be desired to simplify the access on the modules of these libraries by making them "top-level" libraries, which allows to leave out the namespace of the "container" library in the access path. This can be reached by activating option "Publish...". However this option should really **only be activated when creating a container library** and should be handled with great caution!
- For how to uniquely access modules or variables of referenced libraries see below.

Library Versions

- Multiple versions of the same library can be installed on the system.
- Multiple versions of the same library can be included in a project. It is regulated like described in the following, which version of a library an application will use:

  If multiple versions are available on the same level within the same Library Manager, it will depend on the current library properties, which version will be accessed (a defined one or always the newest).

  If multiple versions of the same library are available on different levels within the same Library Manager (which is the case for referenced libraries), unique access to the library modules or variables is reached by adding the appropriate **namespace** (see the following paragraph).

Unique access to library modules or variables

- Basically, if there are several modules or variables with the same name within a project, the access on a module component must be unique, otherwise compile errors will be detected. This applies for local project modules or variables as well as those which are available in included libraries and in libraries referenced by the included ones. The uniqueness is reached in such cases by prefixing the module name by the appropriate library namespace.

- The **default namespace** of a library is defined in the **library properties**. If it is not defined explicitly, it will equates with the library name. However, when creating a library project, also a different standard namespace can be specified in the properties dialog. Later, when the library has got included in a project, the namespace always can be modified locally by the user, again in the properties dialog.
- Examples: Assume that for the following examples the "namespace" of library *Lib1* is specified in the library properties to be "Lib1". See in the right column the use of the namespaces for unique access on variable *var 1* which is defined in the modules *module1* and *POU1* .

	Variable var1 is available at the following locations within the project:	**Access on var1 via using the appropriate namespace path...**
(1)	1. In library *Lib1* in the global Library Manager in the POUs window	"Lib1.module1.var1"
(2)	2. In library *Lib1* in the Library Manager below Application *App1* of Device *Dev1* in the Devices window	"Dev1.App1.Lib1.module1.var1"
(3)	3. In library *Lib1* which is included in library *F_Lib* in the global Library Manager in the POUs window	By default (option "Publish..." in the library Properties is deactivated): "F_Lib.Lib1.module1.var1" If option "Publish..." was activated, *module1* would be treated like a component of a top-level library. Thus accessing would be possible by "Lib1.module1.var1" or "module1.var1". In the current example this however would cause compiler errors because the access path is not unique, see (1) and(4).
(4)	4. In object *module1* which is defined in the POUs window.	"module1.var1"
(5)	5. In object *POU1* which is defined in the POUs window.	"POUxy.var1"

Creating V3.x libraries, Encoding, Documentation

- A »PLC Designer« project can be saved as library (<projectname>.library) and optionally can be installed at the same time in the "System" Library Repository. Only the objects managed in the POUs window will be regarded for a library project. If you pointedly are going to create a library project, it is recommended to choose template "Empty library" in the 'New Project' dialog.
  Notice further on the following:
- In the Project Information a **Title** and a **Version** and the **Company** must be specified. If the **Default Namespace** should be another than the library name, this can be defined immediately here. Additionally it is recommended to specify a **category**, because these will later serve for sorting the entries in the Library Repository and Library Manager dialogs. If the library should belong to another category than the default "Miscellaneous", an appropriate category description must be loaded, either from a XML-file ***.libcat.xml** or from another library, which already contains the information of such an description file. If necessary, a new category description file must be issued or an existing one must be modified. The information of the selected categories and the basic category description file will be transferred to the local library project and later, when installing the library - to the Library Repository. So the categories will be known in the Repository. If afterwards another library again brings a description file with the same ID but different content, then the information of the new file will be valid in the repository.
- If the library includes further libraries, you should consider, how those **referenced libraries** should behave later, when the "father" library will be included in a project. This concerns version handling, namespace, visibility and access properties, which can be configured in the properties dialog of the particular, referenced library. If the library later, when it gets included in a project, always should reference another, device-specific library,a **placeholder** can be used when configuring the reference.
- If the library modules should be protected against viewing and accessing, a library project can be saved in encoded in **encoded format** (<projectname>.compiled-library).
- Data structures of a library can be marked as libinternal. These non-public objects carry the attribute 'hide' and therefore do not appear within the library manager, the "List Components" functionality or the input assistant.
- In order to provide the user with information on a library module in an easy way, an appropriate comment can be added to the declaration of a module parameter. This comment will be displayed later, when the library is included in a project, on the '**Documentation**' tab of the Library Manager.

- The following commands are available by default in the 'File'-menu for saving a library project:
  - Save Project As...
  - Save Project And Install Into Library Repository
  - Save_project_as_compiled_library (Encoding)

»PLC Designer« V2.3 libraries

- Libraries created with »PLC Designer« V2.3 (*.lib) and earlier versions, are supported further on:
- An "old" library project (*.lib) can be opened in »PLC Designer« V3.x and thereby directly converted to a "V3.x library" (*.library).
- If you open an old project, referencing old libraries, you can choose, whether these references should be kept, replaced by others or removed. If they should be kept, the respective libraries will be converted to the new format and automatically be installed to the System Library Repository. If they do not contain the necessary Project Information, those can be immediately added. The scheme, according to which a particular old library has been handled once when converting an old project, can be saved in the project options. Thus, if the same library appears again when converting an old project, its handling must not repeatedly be defined but will be done automatically.

External and internal libraries resp. library modules, Late linking

- An "**external library**" in contrast to an internal library (»PLC Designer« library project) is a library file which has been programmed outside of »PLC Designer« in ANSI-C/C++ . It must be available on the target system and will be linked when the application is running there.
- In order to prepare the creation of a completely external library file, »PLC Designer« allows to **generate the appropriate C-frame and C-interface files** (m4 file format) from the respective »PLC Designer« library project. Using this functionality guarantees the consistency of the »PLC Designer« library and its external implementation. See the respective command 'Generate runtime system files'.
- Like in V2.3 it is also possible to define a »PLC Designer«-programmed library to be an external library, getting **linked later**, that is not before the application is running in the runtime system. In addition now it is even possible to define particular modules of a library for this "late link". For this purpose for the respective modules (all or just some) of the library the "late link" property can be activated (option "External implementation" in the object properties).

3.18 Security

Project:

The access control for projects, particular objects resp. the right to perform certain actions in a project can be configured and managed via dialogs of the Project Settings and object Properties.

A project basically can be protected by an **encrypted password**, see Project Settings, Security.

Access rights concerning objects always are assigned to particular user groups, not to single users. For each object a list of allowed actions can be defined for each user group.

Each particular user however can get an own **password**.

PLC:

Depending on the device there might be also an access control concerning objects and files on the PLC.

4 Quickstart

4.1 Getting Help

Currently a non-dynamic version of online help is installed. By default it can be accessed via the 'Help' menu. The language in which the help pages are displayed can be changed in the Options / International Settings dialog.

Context sensitive help

Short Cut: <F1>

You may press <F1> in within an active window, a dialogue or on a menu command to open the online help.

If a menu command is selected, the corresponding help page will be displayed.

Likewise, if <F1> is executed on a selected text (for example a key word, a basic function or an error message within the message window), the corresponding help page will be displayed.

4.2 Uninstallation, Update, Repair

To uninstall the programming system and its additional components or to modify the current installation execute the current setup file. You will be guided.

4.3 Create and run a project

See in the following a description of how to create a simple project containing a PLC program, further how to load this program via a Gateway Server to the PLC (target device) and to get it run and monitored. The PLC runtime system used for this example project by default is provided with the PLC Designer setup.

The sample program will be written in Structured Text language and consist of a program *PLC_PRG* and a function block *FB1*. *PLC_PRG* will contain a counter variable *ivar* and call function block *FB1*; *FB1* will get input "in" from *PLC_PRG*, will add "2" on this input and will write the result to an output *out*; *out* will be read by *PLC_PRG*.

Notice that the following descriptions refer on the default configuration of the user interface provided with the currently installed version of the programming system.

Note: If your system supports PLC Designer-SoftMotion, regard the description of some sample projects within the help pages on the SoftMotion device editor.

Start PLC Designer and create a project

(1) Start PLC Designer

From the Start menu on your PC choose

Programs > **Lenze** > **PLC Designer** > **PLC DesignerV3.x**

Alternatively you can start via the PLC Designer Icon which will be available on the desktop after installation.

(Future: You will be asked to select a Profile and after pressing **Continue** the user interface will open.)

The programming system will be automatically started with a predefined profile. So, you do not to have to select a profile at the moment.

When you start the programming system the first time after first installation on the system, you will be asked to choose the default collection of settings and features.

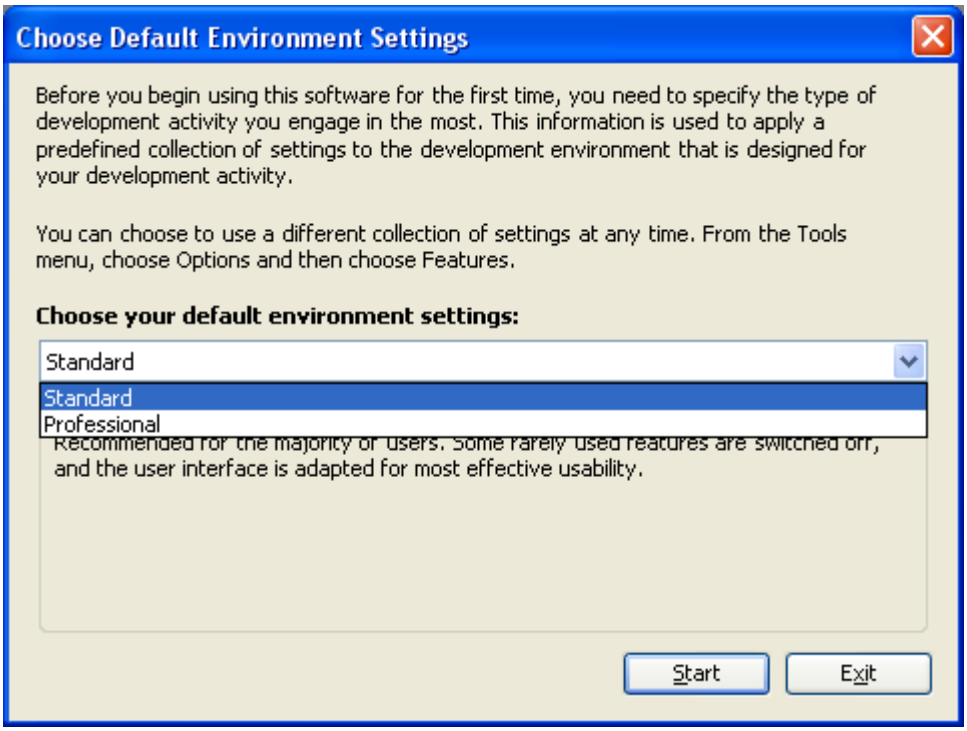
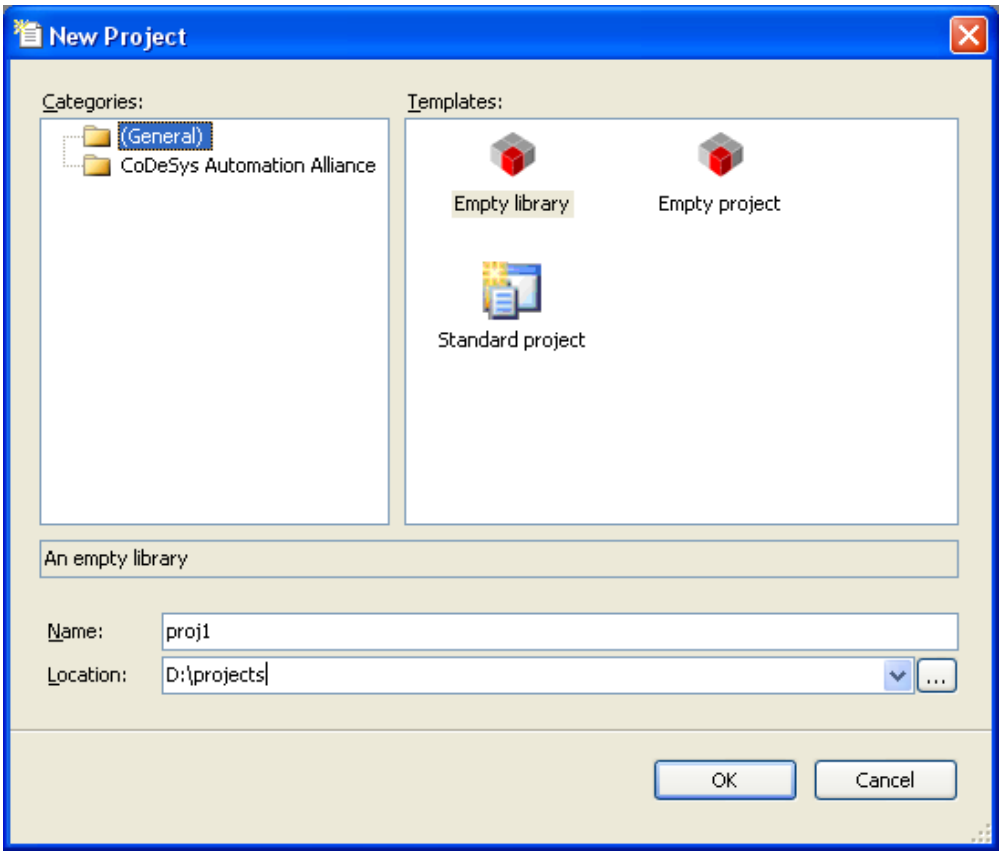


Abb. 36 Dialog 'Choose Default Environment Settings'

Choose "Standard" for the first steps described here, you always can switch to the "Professional" settings (via the Project Options dialog) if necessary . In the "Standard" environment the user interface is adapted for most effective usability and some rarely used features are left out.

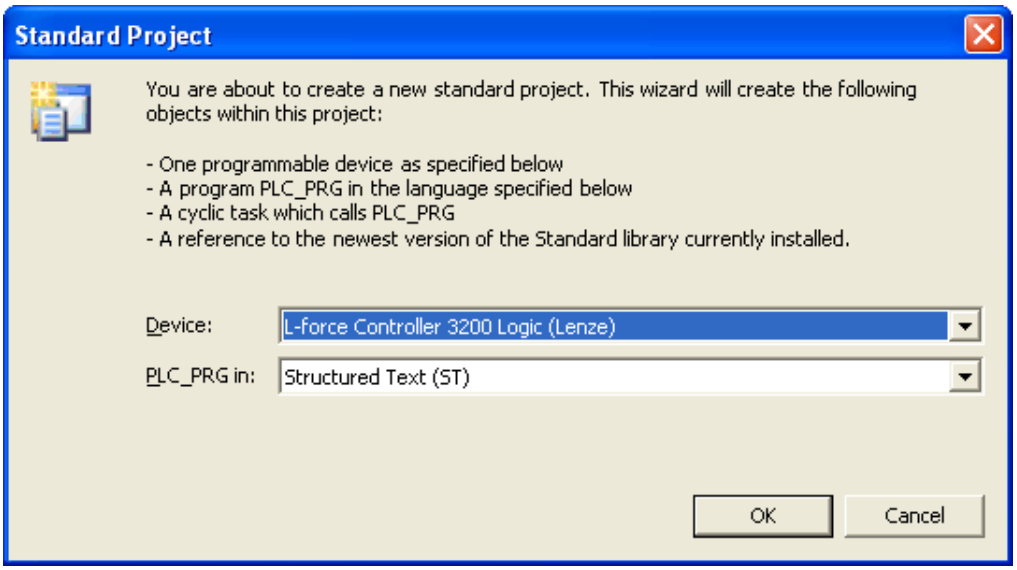
(2) Create a project

To create a new project, choose command **New project** from the **File** menu:



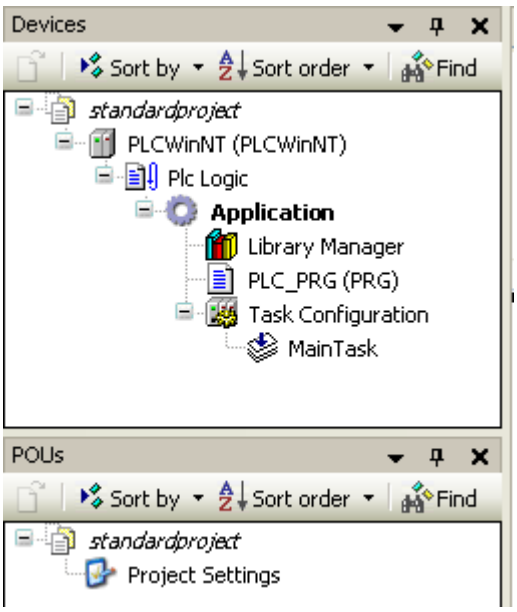
In the **New Project** dialog select **Standard project** in the Templates field and enter a **Name** and a **Location** path for the project file. Press **OK** to confirm.

A wizard dialog will open:



Choose device L-force Controller 3200 Logic (Lenze) or L-force Controller 3200 Motion (Lenze) and programming language Structured Text for PLC_PRG. Click **OK**.

The project name now will appear in the title bar of the PLC Designer user interface and as a symbolic root node in the POUs and the Devices view windows.



The POUs window contains the Project Settings.

The Devices window shows tree with a device "Device (L-force Controller 3200)" and with an application adjoined below. The latter includes the program "PLC_PRG" to be edited in Structured Text and the obligatory Task Configuration defining a "MainTask" that controlls PLC_PRG.

Furthermore there is a Library Manager, automatically including the library "IoStandard.library" necessary for IO-configurations and the Standard.library providing all functions and function blocks which are required matching IEC 61131-3 as standard POUs for an IEC programming system.

(At the moment do not care about the additional node "Plc Logic" inserted below the device node "Device (L-force Controller 3200)". It is just a symbolic node indicating that the device is a "programmable" one.)

The symbolic device name can be renamed by selecting this entry, opening an edit field with the <Space> key and entering another name instead of "Device". Do this and enter "PLCWinNT" instead.

Write a PLC program

(3) Declare variables in PLC_PRG

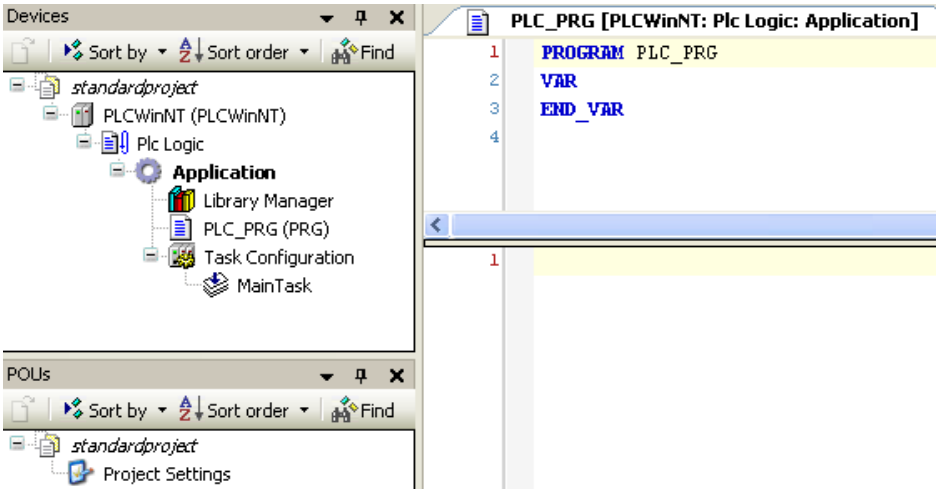
Open POU "PLC_PRG", which by default is already available in the Devices window, in a **ST language editor** window. For this purpose perform a double-click on entry "PLC_PRG" in the Devices tree.

The POU "PLC_PRG", which by default is already available in the Devices window, is automatically opened in a **ST language editor** window in the center part of the PLC Designer user interface. Basically a POU always can be opened in its editor view by a double-click on the entry in the devices resp. POUs tree.

The ST editor consists of a declaration part (upper) and a "body" (lower part), separated by a movable screen divider.

The declaration part shows line numbers at the left border, the POUs type and name ("PROGRAM PLC_PRG") and the embracing keywords "VAR" and "END_VAR" for the variables declaration.

The body is empty, only line number 1 is displayed:



In the declaration part of the editor put the cursor behind VAR and press the <Return>-key. An empty line will be inserted where you may enter the declaration of the variables "ivar" and "erg" of type INT and fbinst of type FB1:

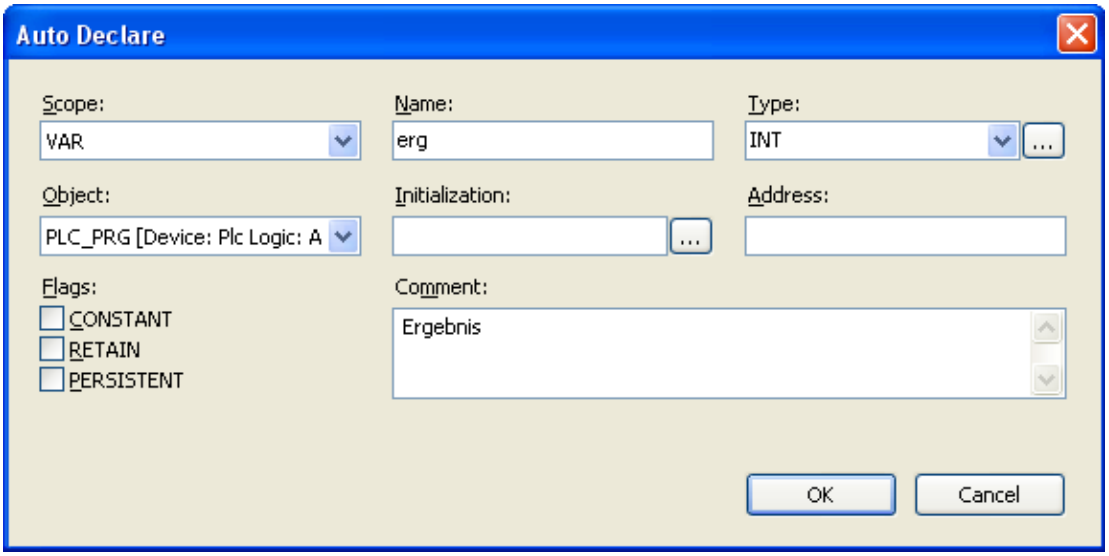
```
PROGRAM PLC_PRG
VAR
ivar: INT;
fbinst: FB1;
erg: INT;
END_VAR
```

Alternatively, you may directly type an instruction in the implementation part of the editor (body) and make use of the Autodeclare function.

(4) Enter programming code in the body of PLC_PRG

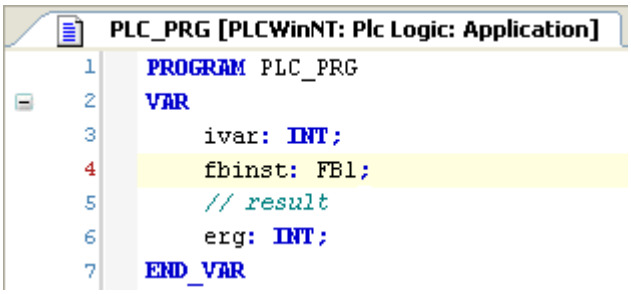
```
ivar := ivar+1;          // counter
fbinst(in:=11, out=>erg);   // call function block of type FB1,
// with input parameter "in
// ouput is written to "erg"
```

Instead of steps (3) and (4) you can use the Auto Declaration feature: Without an preceding declaration enter an instruction immediately in the body of the program, then press the <Return>-key. For each not yet declared variable found in the implementation line the Auto Declare dialog will open, where you now can carry out the declaration settings:



The variables name and scope as well as the current POU (Object) will be filled in automatically. Enter the desired type and initialization value according to the declaration described in (3) and add a comment as shown in the dialog picture.

Confirm the dialog with OK. This will enter the declaration of erg in the declaration part of the POU with the comments ahead:



(5) Create a further programming POU (ST function block FB1)

We supply another function block FB1, which will add "2" on the input given by variable "in". The result will be written to output "out":

Choose command**Add object** from the **Project** menu.

Select 'POU' in the left part of the 'Add Object' dialog. Enter the name "FB1" for the POU and activate option **Function Block** in the Type section.

Choose 'Structured Text (ST)' for the **Implementation language**.

Press button **Open** to confirm the object settings.

A further editor window will open for the new function block FB1. Declare there in the same way as done for PLC_PRG the following variables:

```
FUNCTION_BLOCK FB1
VAR_INPUT
 in:INT;
END_VAR
VAR_OUTPUT
 out:INT;
END_VAR
VAR
 ivar:INT:=2;
END_VAR
```

In the implementation part of the editor enter the following:

```
out:=in+ivar;
```

Define the Resource objects for running and controlling the program on the PLC

(6) Start Gateway Server and PLC

Start Gateway Server:

The Gateway Server is started automatically at system start as a service. Make sure that there is an icon in the system tray, indicating that the gateway is running. If the icon is looking like , the gateway is currently stopped.

(This icon is part of the GatewaySysTray program which is available for controlling and monitoring the Gateway service. It provides a menu with a start and a stop command, thus allowing the user to stop or restart the service manually. NOT YET AVAILABLE: Gateway Inspector function. The menu also includes the command Exit Gateway Control, which just terminates the GatewaySysTray program, not however the Gateway service. The GatewaySysTray program is started automatically when Windows is started, however it also can be started manually via the Programs menu.)

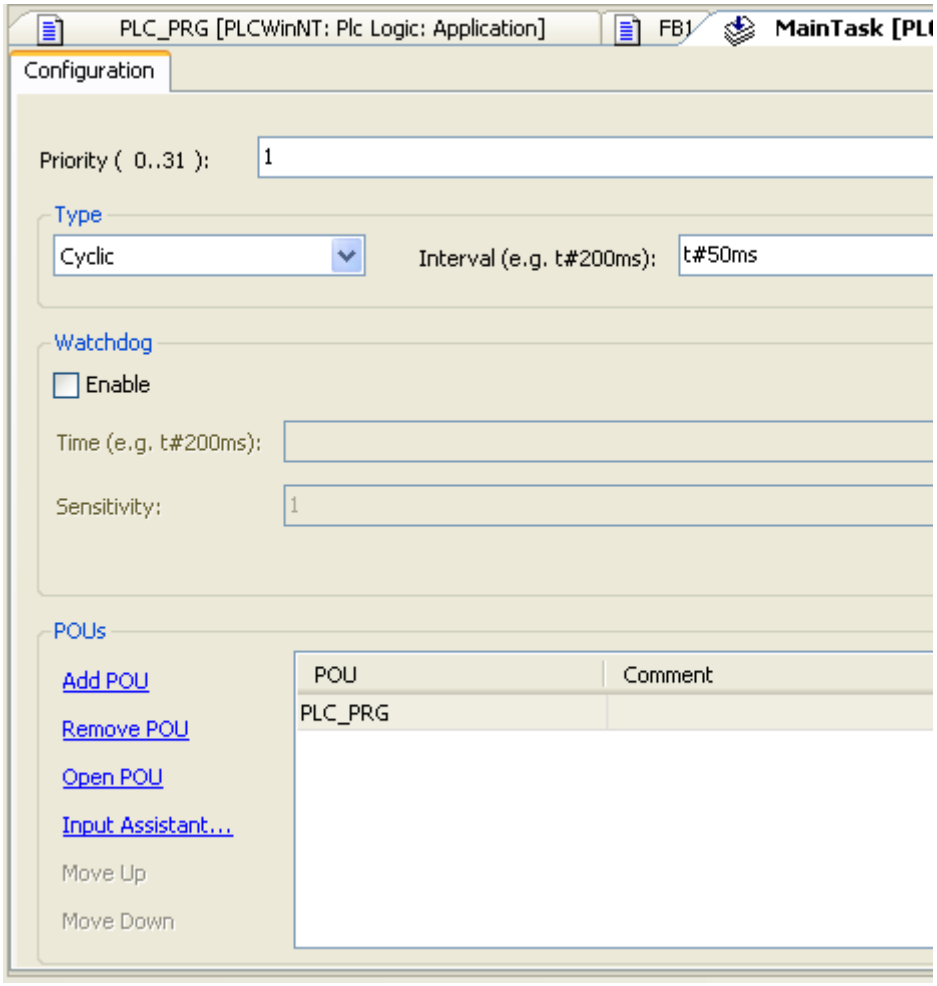
Start PLC:

The PLC (L-force Controller 3200) is available as a service at system start. It is represented by an icon in the system tray: for status 'stopped', for status 'running'. If allowed by the system, the PLC service will be automatically started at system start. Otherwise you have to start it manually by command 'Start PLC' from the menu you open by a mouse-click on the icon.

(This icon is part of the PLCDesignerSPSysTray program which is available for controlling and monitoring the L-force Controller 3200 service. It provides a menu with a start and a stop command, thus allowing the user to stop or restart the service manually. The menu also includes the command Exit PLC Control, which just terminates the PLCDesignerSPSysTray program, not however the PLCservice. The PLCDesignerSPSysTray program is started automatically when Windows is started, however it also can be started manually via the Programs menu.)

(7) Set the "Active Application"

In consequence of a mouse-click on ‘MainTask’ in the devices window of “standardproject” an editor view containing the configuration of the task opens:



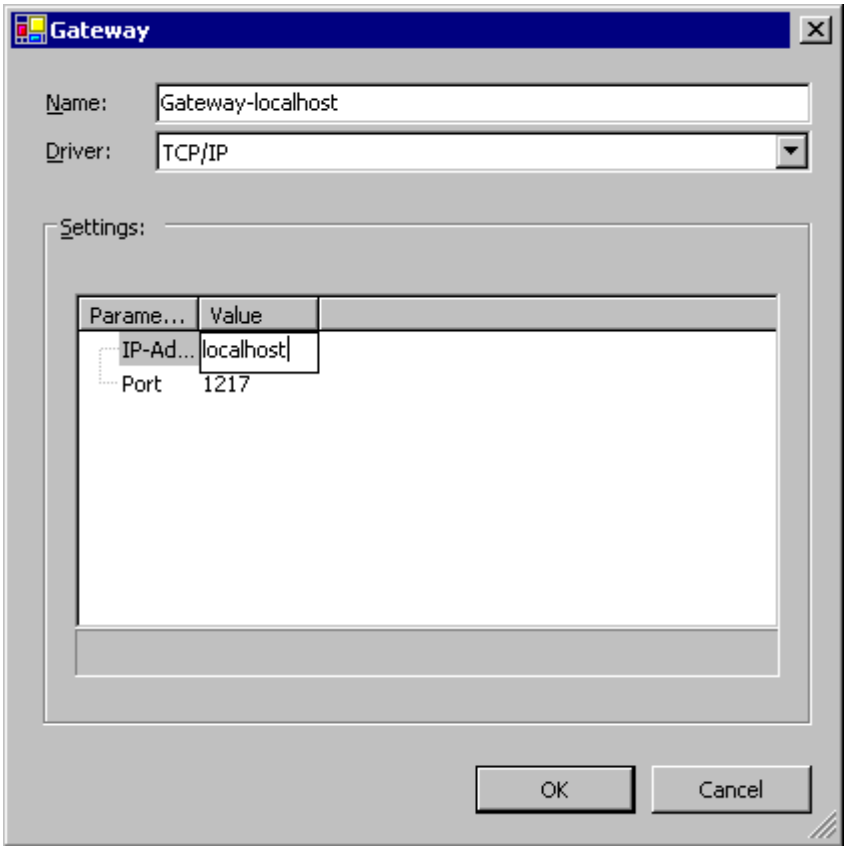
In the Devices window the name ‘Application’ is displayed in bold letters. This means that this application is set as “active application”. Thus all commands and actions concerning the communication with the PLC will refer to this application. Basically in order to set an application as "active one" select the application entry in the Devices window and choose command 'Set Active Application' from the context menu.

(8) Configure a communication channel to the PLC

Perform a double-click on entry “Device (L-force Controller 3200)” in the devices window. The dialog “PLCWinNT” will open with subdialog “Communication Settings”. Here you have to set up the connection between the PLC (target, device) and the programming system according to the following steps. The resulting connection will finally be entered in the line below Select network path to the controller:

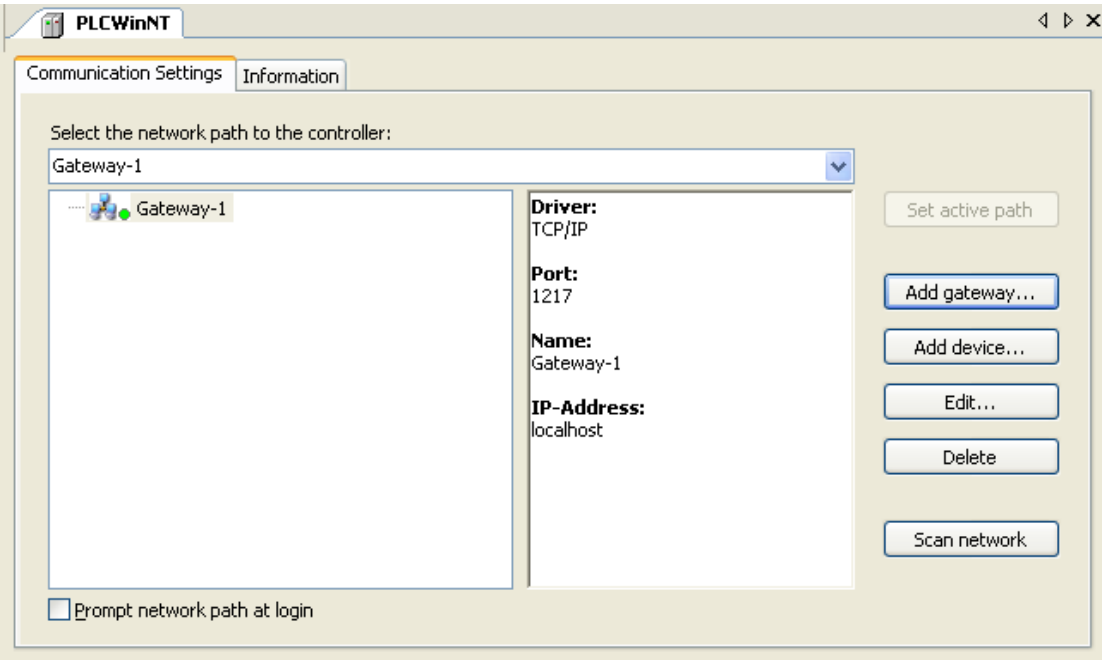
If this is your first communication setup with PLC Designer V3.x, you now have to define the local Gateway Server. (If you have defined the server already in previous sessions, it will be displayed in the communication settings dialog as shown in the figure below in chapter "Now define....". In this case you can skip this step and continue with defining the communication channel to the target, see also next chapter "Now define...".)

The server is provided with the PLC Designer setup. Press button **Add gateway** to open the Gateway dialog:



Enter a symbolic **Name** for the Gateway, define the Driver type "TCP/IP" and enter IP-address "localhost" (Perform a double-click on the column field to open an edit frame). Leave the setting for the Port . Confirm with **OK**.

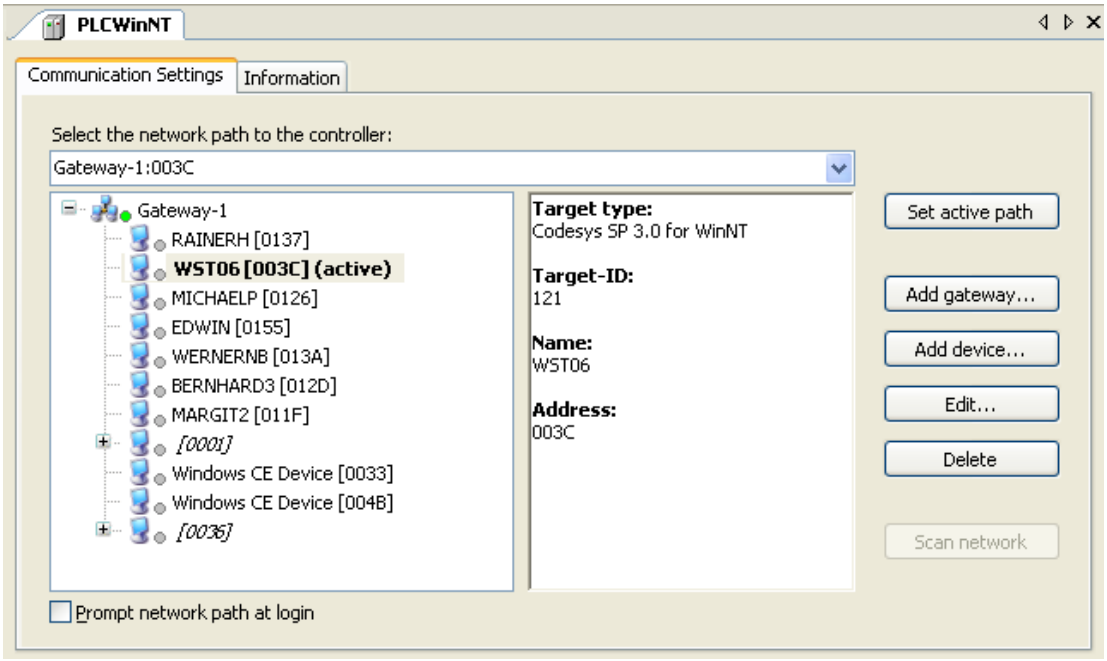
The gateway will be entered in the field on the left part of the Communication dialog and the name of the Gateway will be added to the selection list under 'Select the network path to the controller'. When the gateway is properly running, a green bullet is displayed before the entry, otherwise a red one:



Now define a channel to the target device which should be connected via the set gateway:

Press button **Scan network** to search for available devices in your local network.

You should at least find the PLC installed with the PLC Designer setup: It will be displayed indented below the gateway: instead of "WST06 [003C]" shown in the picture below you should find the name and address of your computer. If your PLC is not found, check if it is running, see (6).



Now select the PLC (device) entry and press button **Set active path**.

This will set this communication channel as the active one, which means that all actions concerning communication will exactly refer to this channel. Notice this later when you may have set up several communication channels in your project.

Close the Communication dialog with **OK** to apply the settings.

Run and watch the application on the PLC

(9) Compile and load application to the PLC

If you just want to check your "active" application program for syntactic errors, perform command 'Build Application' (Context menu, Build menu) or use <F11>. Note: No code will be generated in this case. Information, warnings and error messages will be displayed in the Messages window which is placed at the lower part of the user interface by default.

Even if this syntactical check has not be done before, you can log into the PLC. (Therefore make sure, that the PLC is running, that is the symbol in the system bar is colored).

Use command Login to ‘Application’ (Online menu).

If the communication settings have been configured as described in (8) the following message box will appear (otherwise you will be asked to correct the communication settings): "There is no device application on target PLCWInNT. Do you want to create it and proceed with download?" Confirm with **Yes** to start the compilation and download of the application.

The compile messages will be displayed in the Messages window. If the project has been created correctly, no compilation errors are to be expected, so that the application can now be started, see (10).

(10) Start and monitor application

Having created a 'standardproject' and downloaded the application 'Application' as described in the previous steps, this application can now be started on the device L-force Controller 3200 (‘PLCWinNT').

Starting the application on the PLC:

Perform command 'Start Application', which is in the context menu by default, when the application object is selected. In consequence the program starts running. A green RUN will be displayed in the status bar at the bottom of the user interface.

Monitoring the application:

There are three possibilities for watching the variables of the application program:

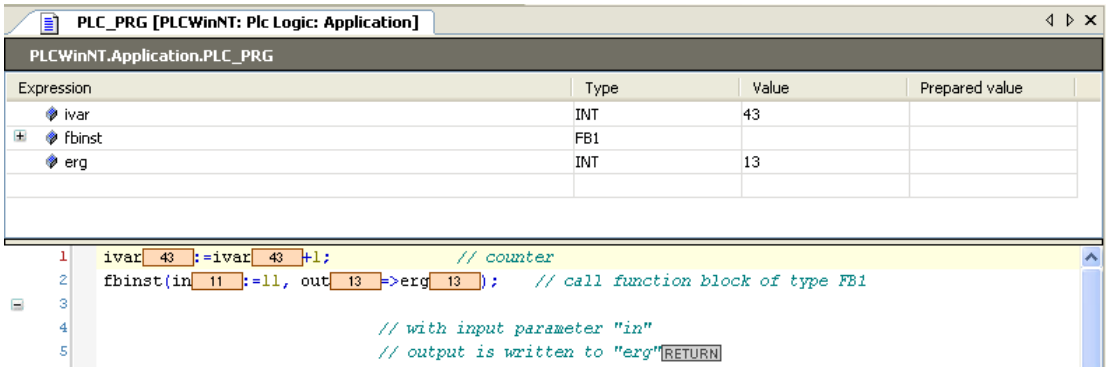
1. Watch views with defined watch lists
2. Writing and forcing variables
3. Online views of the particular POUs

1. Open an instance window of the program

The instance view of a POU provides all watch expressions of that instance in a table view in the declaration part and – if activated – as “inline monitoring” also in the implementation part.

In order to open the online view, perform a double-click on "PLC_PRG" in the Devices window or select this entry and choose command Edit object (context menu).

In the lower part of the view you see the code lines as entered in offline mode, supplemented by the little inline monitoring windows behind each variable, showing the actual value. In the upper part a table shows the watch expressions of the POU, that is the current values of the respective variables in application "Application" on the PLC.



2. Writing and forcing variables:

You can write or force a "Prepared value" to variable ivar on the PLC, which means that ivar will be set to this value at the beginning of the next cycle. Perform a double-click on the field in column Prepared value, enter a desired integer value and leave the field by <Return> or by a mouse-click outside of the field. Perform command **Write values** resp. **Force values** (Debug menu) to write or force this value to the PLC. You will see the result at once in column Value.

3. Use the watch views:

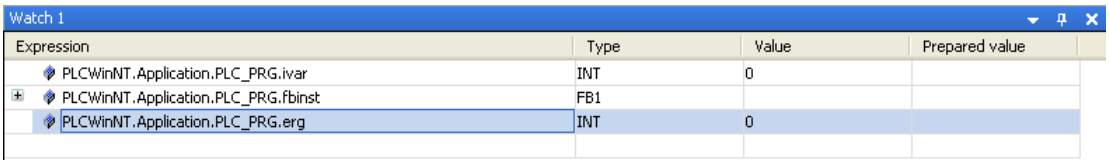
Watch view windows can be used to configure specific sets of watch expressions of the application, for example for the purpose of debugging.

From the View menu use command **Watch** -> **Watch1**. The watch window will open.

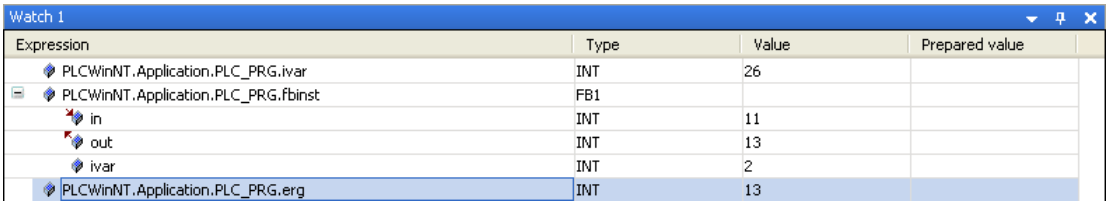
In column Expression perform a mouse-click in the first line of the table to open an edit frame. Enter the complete path for variable ivar which should be monitored: "PLCWinNT.Application.PLC_PRG.ivar".

It is recommended to use the input assistant via button for this purpose. Close the edit frame with <Return>. The type will be added automatically.

Do the same for the further variables. The watch list shown in the next picture just contains expressions of PLC_PRG, but of course you might create a set of any variables of your project. Notice that for instance variables, for example for the FB1-instances, it is sufficient to enter the expression "PLCWinNT.Application.PLC_PRG.fbinst". The particular variables will be entered automatically and the corresponding lines can be opened via the plus-symbol: The current value of a variable is displayed in the 'Value' column:



If not yet done, now select the application object and perform command **Start** from the context menu. The application will be started on the PLC and the current value will be displayed in column Value:



Writing and forcing values is also possible here like described above in 2.

To disconnect from the PLC perform command Logout from the Online menu.

Debug an application

(11) Set breakpoint and step through the program

In online mode you can set breakpoints as defined halt positions for the program execution.

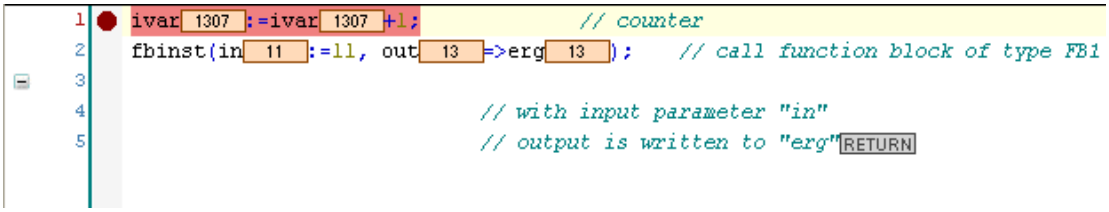
When the program has reached a breakpoint you can execute the program in single steps. At each halt position you see the current value of the variables in the monitoring views.

Try the following:

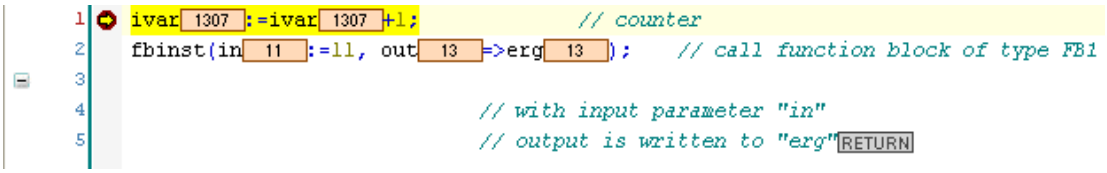
Select line 1 of PLC_PRG. Press key <F9>, which equals the command **Toggle Breakpoint** from the Debug menu.

The breakpoint will be indicated.

If the application currently is in STOP state, this will look like:



A running application will stop at a breakpoint:



Now you can step further by using <F8>, which represents command **Step Into** from the Debug menu and therefore will step also into the function block instance. To skip the steps of the function block use <F10> which equals the command Step Over. Each variable value currently read from the PLC will be displayed.

You might also have a look at the breakpoints dialog to be opened via command Breakpoints from the View menu. Here, the breakpoints currently set can be viewed and edited and new breakpoints might be entered.

Notice also that the breakpoint positions will be remembered when you log out. They will be indicated by faded red bullets.

5 User Interface

5.1 Customizing the User Interface

The actual look of the user interface, which means arrangement and configuration of the particular components, depends on the following:

- Standard pre-settings for menus, keyboard functions and toolbars. Default settings are installed with PLC Designer, but can be overwritten by the user via the **Customize** dialogs (typically found in the "Tools" menu) . Always the current settings get saved on the local system. A reset function is available for restoring the default values at any time.
- Properties of an editor as defined in the respective **Options** dialogs. Also these settings can be overwritten by the user and the current configuration also will be saved on the local system.
- Arrangement of views or editor windows within the project, done by the user. The current positions are saved with the project (see below).

Arranging menu bars and tool bars

The menu bar is always positioned at top of the user interface, between window title bar and view windows. A toolbar can be positioned within the same area as the menu bar (fix) or as an independent window anywhere on the screen.

To re-position a bar, click with the cursor on the **dotted line** at the left end of the bar, keep the cursor pressed and shift the bar to the desired position.

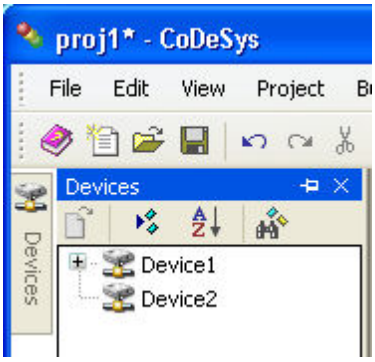


Abb. 37 Dotted line on left side of menu and tool bars

Note: In view windows like for example POUs, Devices or Visual Element Properties a special toolbar is available, providing buttons for sorting, viewing, searching within the window. This bar cannot be configured.

Arranging windows and views

Closing a view or editor window: Press the ✕ button in the upper left corner.

Opening a closed view: By default the views of standard components can be re-opened via the commands '<viewname>' available in the 'View' menu. An editor window can be re-opened via the command 'Edit object' or via a double-click on the respective entry in the POUs or Devices view.

Resizing a view or window within the frame window: Move the separator lines between neighboured views. Independent view windows on the desk top can be resized by moving the windows borders.

Moving a view to another position on your desk top or within the frame window: Click in the title bar or in case of tabbed views alternatively on the tab of the view, keep the mouse-button pressed and move the view to the desired place. Arrow symbols will appear showing every possible target position. Keep the mouse-button pressed and choose the desired position by moving the cursor on the respective arrow symbol. The target position will be indicated by a blue-shadowed area.

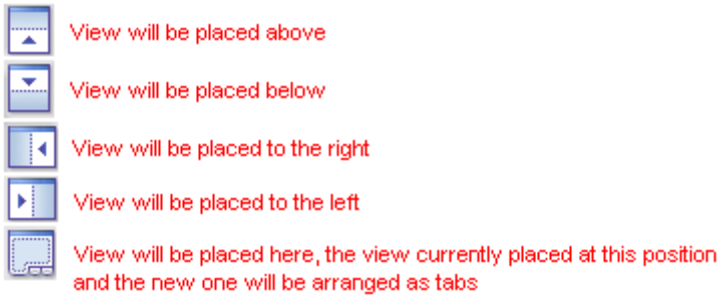


Abb. 38 Symbols indicating new position

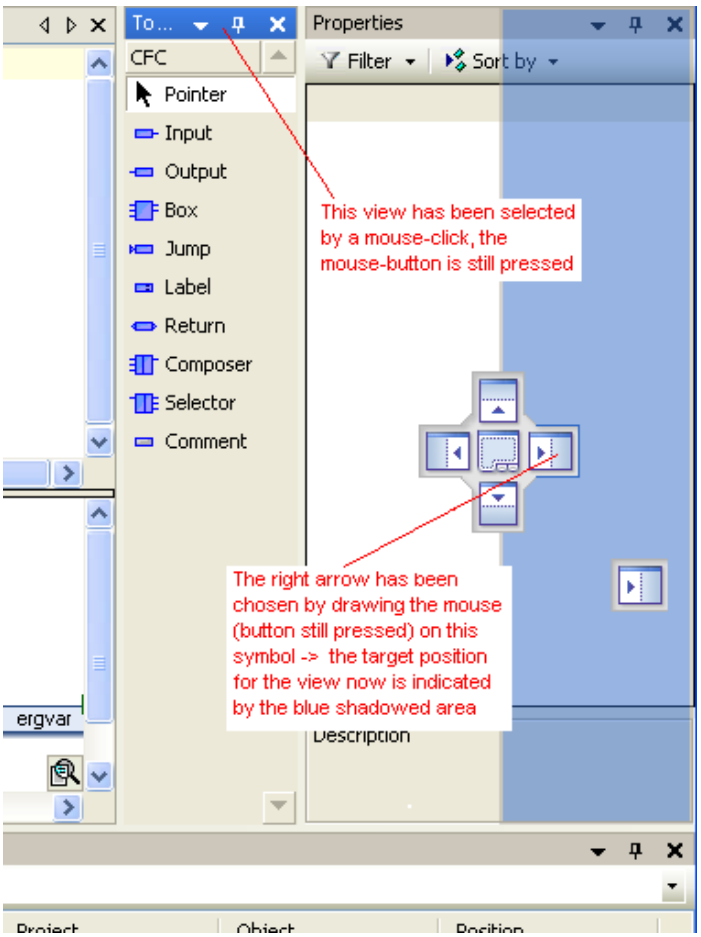


Abb. 39 Example of navigation by arrow buttons

When leaving the mouse-button, the view will be placed at the new position. Views with a **docking button** can be placed as independent windows (**floating**) anywhere on the screen by moving them and not drawing them on one of the above described arrow symbols. In this case it will loose the docking button. Alternatively for this purpose the dock and float commands can be used, which by default are part of the Windows menu.

Hiding views: Views with docking buttons can be "hidden" at the border of the PLC Designer frame. Press button in the upper left corner of the view. The view will be represented by a tab at the nearest border of the frame window. The content of the view will only be visible as long as the cursor is placed in this tab. The tab displays the view icon and name. This state of the view is indicated by the docking button changed to .

To "unhide" the view, press the button. This hiding and unhiding of a view also can be handled via the option Auto hide, which per default is part of the Windows menu.

Notice that the **Information and Status Line**, by default on the lower border of the user interface cannot be repositioned.

Zoom

Each editor window provides a zoom function. The zoom button in the lower left corner of the window opens a list from where you can choose one of the zoom levels: 25, 50, 75, 100, 150, 200, 400 percent. Notice that a printout always refers to the 100% view.

Customization of the user interface is **possible in offline and in online mode**.

5.2 User Interface in online mode

As soon as you log in with the project, all objects which have been opened already in offline mode, automatically will be viewed in online mode.

To open an object in online mode which had not been opened already in offline mode, perform a doubleclick on the object entry in the POUs or Devices window or use command 'Edit Object'.

If your choice is unambiguous, the object will be opened in online mode. Otherwise, for example if there are several instances of the selected object (function blocks etc.) contained in the project, a dialog named '**Select Online State <object name>**' will appear, where you can choose whether an instance or the base implementation of the object should be viewed and whether the object should be displayed in online or offline mode:

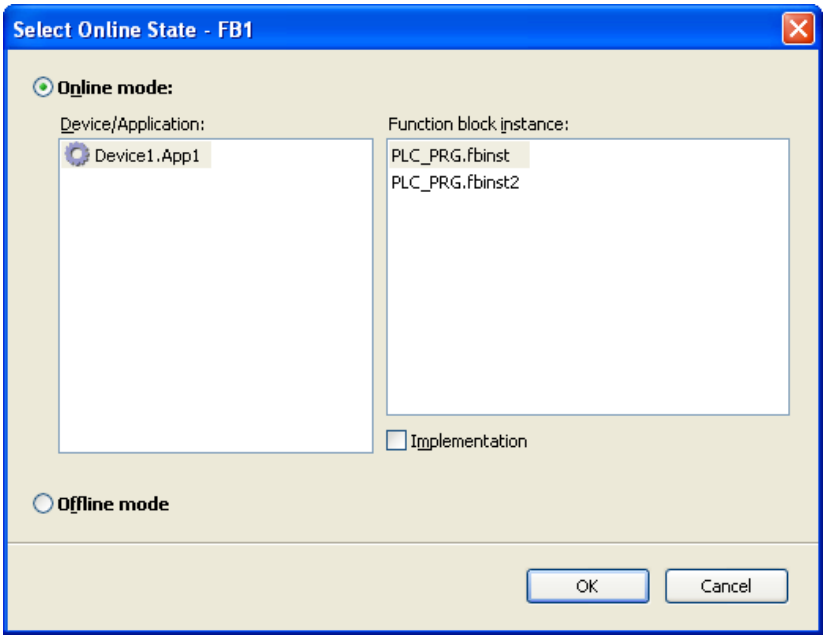


Abb. 40 Select Online State Dialog

In the **Device/Application** field see the Device and Application to which the respective object is associated.

To open the online view of the object activate option **Online mode** and press OK. To see the offline view activate option **Offline mode**.

If the object is a function block, in the **Function block instance** field you will find listed all instances currently used in the application. In this case you can

- either select one of the instances and activate Online or Offline mode, or
- select option **Implementation** which - independently of the selected instance - will open the base implementation view of the function block. The Implementation option is of no effect for non-instantiated objects.

For more information on the online views of the particular editors refer to the respective editor descriptions.

The status bar will provide information on the current status of the application.

5.3 User Interface Components

The PLC Designer programming user interface is an arrangement of "Components" (see in detail below).

The look of the user interface on the one hand depends on the **customization** settings, which are stored in a special file and define the availability of menus, toolbars, keyboard usage. The current customization settings can be looked up and modified in the **Customize** dialog, which typically is found in a **Tools** menu.

On the other hand the look of the user interface depends on the **arrangement** of "views" and windows which can be modified by the user anytime via shifting, docking/undocking views, resizing or closing windows.

The user interface provides **menus** and **tool bars**, editor and object organization and watch and message **windows** and an **information and status line**.

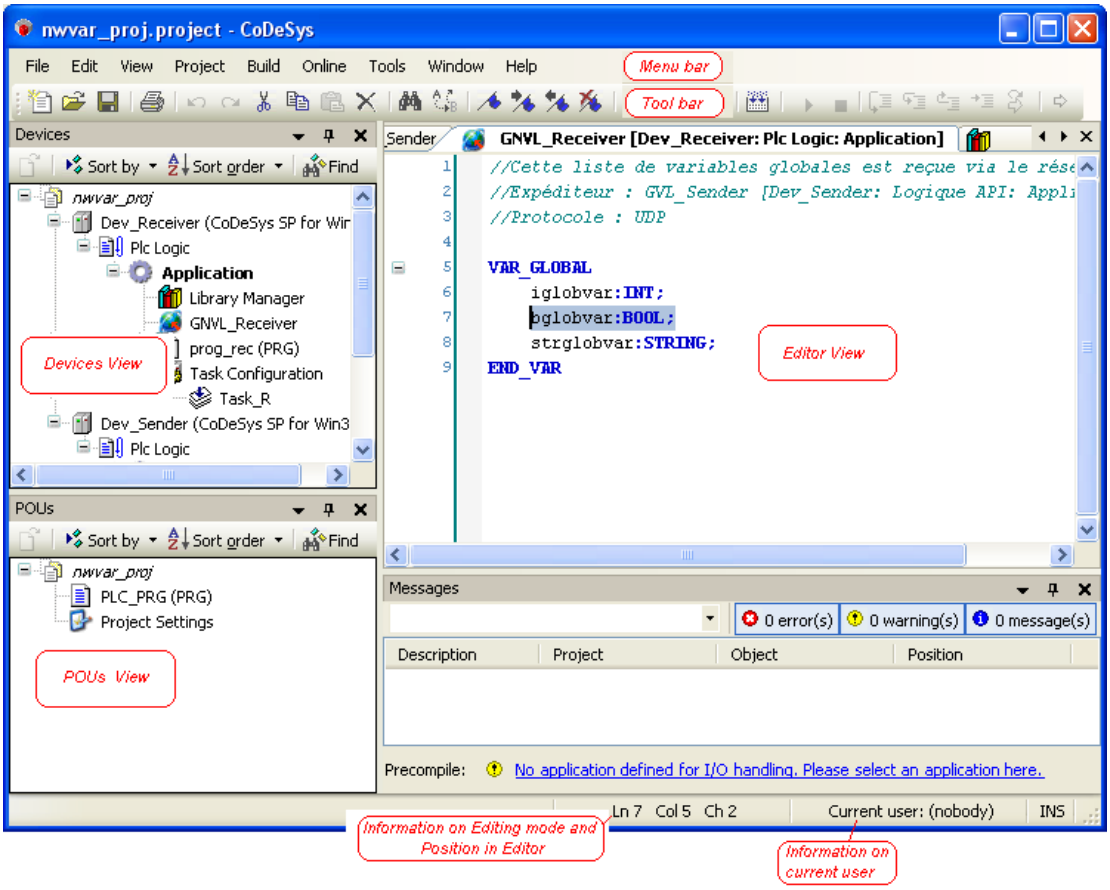


Abb. 41 Example of the PLC Designer V3.x User Interface

The standard components:

- **Menu Bar**: Provides menus which contain all currently available commands as defined in the Customize dialog. By default visible.
- **Tool Bar**: Contains tool buttons for all currently available tools as defined in the Customize dialog resp. provided by a toolbox plug-in. By default visible.
- **ToolBox window: Provides editor-specific tools (open from View menu).**
- **POUs** window (view): For organizing the programming units (POUs, DUTs etc.) of a project in a tree structure (open from View menu).
- **Devices** window (view): For organizing the device resource objects of a project in a tree structure (open from View menu).
- **Editor** window (view): Used for creating the particular object in the respective editor. In case of language editors (for example ST-Editor, CFC-Editor) usually the window combines the language editor in the lower part and the declaration editor in the upper part. In case of other editors It also can provide dialogs (for example Task-Editor, Device-Editor). The POU's or Resource object's name always is displayed in the title bar of the window. The objects can be opened in the editor window in offline or online mode via command Edit Object.

For information on what's currently going on in your project in off-line or online mode see the following components:

- **Messages** Window: Precompile, Compile, Build, Download messages etc. are displayed in this window (message view).
- **Watch** Windows and online views of editors: Shows a monitoring view of a POU resp. a user-defined list of watch expressions.
- **Information and Status Line:** The line at the lower border of the user interface provides information on the currently logged-in user. Also - if you are currently working in an editor window - the current position of the cursor and the status of editing mode. In online mode the current status of the program will be indicated.

  Current user: Each project has a user and access management. The currently logged-in user will be named in the status line.

  Position: Counted from the left resp. upper margin of the editor window:
  Ln = Number of lines
  Col = Number of columns (a column includes exactly 1 space, character or digit)
  Ch = Number of characters (in this context a character can be a single character or digit as well as a tab including for example 4 columns)
  By a double mouse-click on one of the fields you get the dialog **Go To Line**, where you can enter a different position where the cursor should be placed.

  Status of editing mode: **INS** = insert mode, **OVR** = overwrite mode. By a double mouse-click on this field you can toggle the setting.

  Online mode information: status of the application on the device:
  RUN = program running
  STOP = program stopped
  HALT ON BP = program halted on a breakpoint
  Program loaded = program loaded on device
  Program unchanged = program on device matches that in the programming system
  Program modified (Online Change) = program on device differs from that in the programming system, online change required
  Program modified (Full download) = program on device differs from that in the programming system, full download required

Windows, Views, Editor windows:

The windows you get displayed within or beside the user interface frame window at first sight all look the same. But there are two types:

1. Some can be **docked** to any margin of the frame window or alternatively can be positioned on the screen as undocked windows independently from the frame window . Also they can be "**hidden**", that is just represented by a tab in the frame border. Those windows display information which is not depending on a single object of the project, for example Messages, Devices, POUs, Toolbox. They can be accessed via the '**View**' menu commands and also are named "views". Most views include a non-configurable toolbar with buttons for sorting, viewing, searching within the window.

2. Others open when you are viewing or **editing** a specific project object in the respective editor. Those are displayed in a tabbed editor area or depending on the given user interface settings as MDI windows. They **cannot be** "**hidden**" resp. **undocked** from the frame window. They can be accessed via the '**Window**' menu commands.

Additional types of windows or views might be added via manufacturer specific components.

6 Standard menus and commands

See in the following an overview on the default structure of the main menus and commands as provided with the "standard project" template.

Further commands may be available and can be inserted additionally in these menus or in user-defined menus. For example the special commands for a certain editor usually are available in a corresponding menu which will be available when the editor is opened (example: when you edit an object in the SFC-Editor, the "SFC" menu is added to the menu bar).

To modify the menu structure or shortcuts use the Customize dialogs (Tools menu).

Abb. 42 Standard menu bar

Standard menus:

File menu: Commands for actions on the project file (open, close, save, print, page setup, source download/upload...).

	New Project...	<Ctrl>+<N>
	Open Project...	<Ctrl>+<O>
	Close Project	
	Save Project	<Ctrl +<5>
	Save Project As...	
	Save Project And Save Into Library	
	Project Archive	
	Extract Archive...	
	Save/Send Archive...	
	Source download...	
	Source upload...	
	Print	
	Page Setup...	
	Recent Projects >	
	<n> <project path>	
	...	
	Exit	<Alt>+<F4>

Edit menu: Commands available for working in editors (language editors, declaration editor).

	Undo	<Ctrl>+<Z>
	Redo	<Ctrl>+<Y>
	Cut	<Shift>+<Del>
	Copy	<Ctrl>+<Ins>
	Paste	<Shift>+<Ins>
	Delete	<Del>
	Select All	<Ctrl>+<A>
	Find & Replace >	
	Find	<Ctrl>+<F>
	Replace	<Ctrl>+<H>
	Find Next	<F3>
	Find Next (Selected)	<Ctrl>+<F3>
	Find Previous	<Shift>+<F3>
	Find Previous (Selected)	<Shift>+<Ctrl>+ <F3>
	Insert File As Text...	
	Browse	
	Go To Definition	
	Advanced >	
	Overwrite Mode	<Ins>
	Go To Line...	
	Make Uppercase	<Shift>+<Ctrl>+C
	Make Lowercase	<Ctrl>+<V>
	Go To Matching Bracket	
	Select To Matching Bracket	
	Bookmarks >	
	Toggle Bookmark	<Ctrl>+<F12>
	Next Bookmark	<F12>
	Previous Bookmark	Umschalt+F12
	Clear Bookmarks	
	Input Assistant...	F2
	Auto Declare...	Umschalt+F2
	Next Message	F4
	Previous Message	Umschalt+F4
	Go To Source Position	

View menu: Commands for activating the particular standard views, that is getting them displayed in a window in the user interface. Also notice the Window menu (see below).

	POUs	
	Devices	
	Messages	
	Element Properties - for SFC element - for Visualization element	
	ToolBox	
	Watch >	
	Watch <n>	
	...	
	Breakpoints	
	Cross Reference List	
	Call Stack	
	Pending Checkins	
	Start Page	
	Full Screen	<Shift> + <Ctrl> + <F12>
	Properties...	

Project menu: Commands for handling project objects and the general information on the project, for copying (merge) and export of projects and for Source Control and User Management.

	Add Object...	
	Add Device...	
	Insert Device...	
	Plug Device...	
	Scan for Devices...	
	Update Device	
	Add Folder...	
	Edit Object	
	Edit Object With...	
	Set Active Application	

	Project Information	
	Project Settings	
	Project Environment	
	Document...	
	Compare...	
	Export	
	Import	
	Merge Project...	
	User Management	
	User Logon...	
	User Logoff	
	Permissions...	
	Source Control	
	Open from Source Control	
	Change Source Control	
	Add Objects From Source Control	
	Get Latest Version (Entire Project)	
	Get Latest Version (Selected Objects)	
	Checkout...	
	Checkin...	
	Undo Checkout	
	Show Differences	
	History...	
	Project History...	
	Label...	
	Refresh Status	
	Login...	

Build menu: Commands for building the project, that is for doing a pre-compilation run including a syntactical check; also commands for deleting the last compile information (clean) which is of meaning for Online Change and for offline code-generation.

	Build	<F11>
	Rebuild	
	Generate code	
	Generate runtime system files...	
	Clean	

Online menu: Commands for logging in and out to/from the controller, for loading the project on the controller and for reset.

	Login	<Alt>+<F8>
	Logout	<Ctrl>+<F8>
	Create boot application	
	Download	
	Online Change	
	Source download to connected device...	
	Multiple_download	
	Reset warm	
	Reset cold	
	Reset origin	

Debug menu: Commands for controlling the program run on the controller (start, stop) and for debugging actions (breakpoints, stepping, writing, forcing).

	Start	<F5>
	Stop	<Shift>+<F8>
	Single Cycle	
	New Breakpoint...	
	Toggle Breakpoint...	<F9>
	Step Over	<F10>
	Step Into	<F8>
	Step Out	<Shift>+<F10>
	Run to Cursor	
	Set next statement	

	Show Current Statement	
	Write Values	<Ctrl>+<F7>
	Force Values	<F7>
	Unforce all Values	<Alt>+<F7>
	Add all Forces to Watchlist	
	Display Mode >	
	Binary	
	Decimal	
	Hexadecimal	

Tools menu: Commands for opening tools which serve to prepare the environment for working on a project (installation of libraries and devices, customization of the user interface, options for editors, loading&saving etc.)

	Library Repository	
	Device Repository...	
	Visual Element Repository...	
	Customize...	
	Options...	

Window menu: Commands for handling (arrangement, opening, closing etc.) the windows in the user interface. Also notice the View menu (see above).

	Next Editor	<Ctrl>+<F6>
	Previous Editor	<Shift>+<Ctrl>+<F6>
	Close Editor	<Ctrl>+<F4>
	Close All Editors	
	New Horizontal Tab Group	
	New Vertical Tab Group	
	Float	
	Dock	
	Auto Hide	
	Next Pane	<F6>
	Previous Pane	<Shift>+<F6>
editor icon	<n> <window title> (application path)]	
	Windows...	

Help menu: Commands for getting online help and information on the programming system.

7 User and Access Rights Management

The "User Management" and "User Management UI" plug-ins provide functions for defining user accounts and configure the access rights within a project. Notice that also device specific user management might be supported for controlling the users access rights on the PLC file system and objects during runtime.

The rights to access project objects via specified actions are assigned only to user groups, not to a single user account. So each user must be member of a group.

User Management

Before setting up users and user groups, notice the following:

- The configuration of users and groups is done in the **Project Settings** dialog.
- Automatically there is always a **group "Everyone"** and by default primarily each defined user or other groups are members of this group. Thus each user account at least automatically is provided with defined default settings. Group "Everyone" cannot be deleted, just renamed, and no members can be removed from this group.
- Also automatically there is always a **group "Owner"** containing one **user "Owner"**. Users can be added to or removed from this group, but at least one user must remain. This group also cannot be deleted and always has all access rights. Thus it is not possible to make a project unusable by denying the respective rights to all groups. Both group and user "owner" might be renamed.
- When starting the programming system resp. a project, primarily no user is logged on the project. But then the user optionally might log on via a defined user account with user name and password in order to have a special set of access rights.
- **Notice that each project has its own user management!** So, for example to get a special set of access rights for a library included in a project, the user must separately log on to this library. Also users and groups, set up in different projects, are not identical even if they have identical names.

Note: The user passwords are stored irreversibly. If a password gets lost, the respective user account gets unusable. If the "Owner“-password gets lost, the entire project might get unusable!

Access Right Management

User management in a project is only useful in combination with the access right management. Notice the following:

- In a **new project** basically all rights are not yet defined explicitly but set to a **default** value. This default value usually is: "granted".
- In the further run of working on the project **each right can be explicitly granted or denied** resp. set back to default. The access right management of a project is done in the Permissions dialog or - for object access rights - in the Access Control dialog which is part of the object Properties dialog.
- Access rights on objects get "**inherited**". If an object has a "father" object (example: if an action is assigned to a program object, that is inserted in the structure tree below the program, then the program is the "father" of the action object) , the current rights of the father automatically will become the default settings of the child. Father-child relations of objects concerning the access rights usually correspond with the relations shown in the POUs or Devices tree and are indicated in the Permissions dialog by the syntax "<father object>.<child object>".
  Example: Action ACT is assigned to POU object PLC_PRG. So in the POUs window ACT is shown in the objects tree indented below PLC_PRG. In the Permissions dialog ACT is represented by "PLC_PRG.ACT" indicating that PLC_PRG is the "father" of ACT. If the "modify" right would be denied explicitly for PLC_PRG and a certain user group, the default value of the "modify" right for ACT automatically also would be "denied".

8 Visualization

8.1 Visualization in »PLC Designer« V3

General mechanism

In contrast to Version V2.x the remote visualization clients are nothing else but interpreters of draw commands. Independent of its type (web, target, HMI) each client will get the same instructions, so that the resulting visualizations will all be the same.

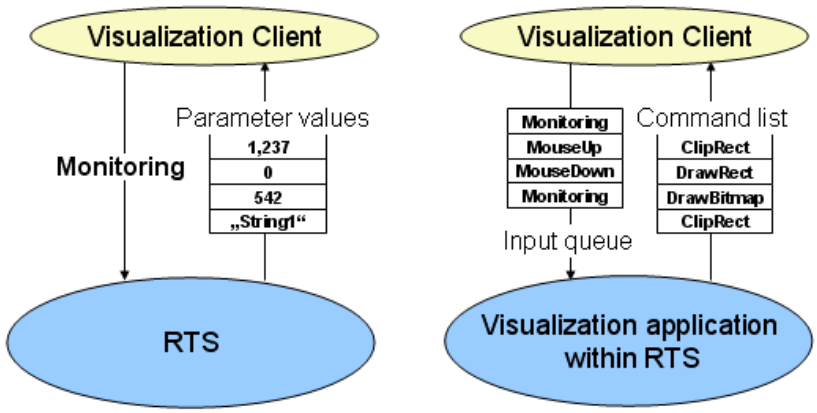


Abb. 43 Comparison of visualization mechanisms - Version V2.x (at left) versa Version V3.x (at right)

As a consequence the related visualization application has to be downloaded to the PLC, which might cause trouble for small PLCs. To overcome this problem a data server is integrated into the runtime system to provide the necessary data for the visualization application.

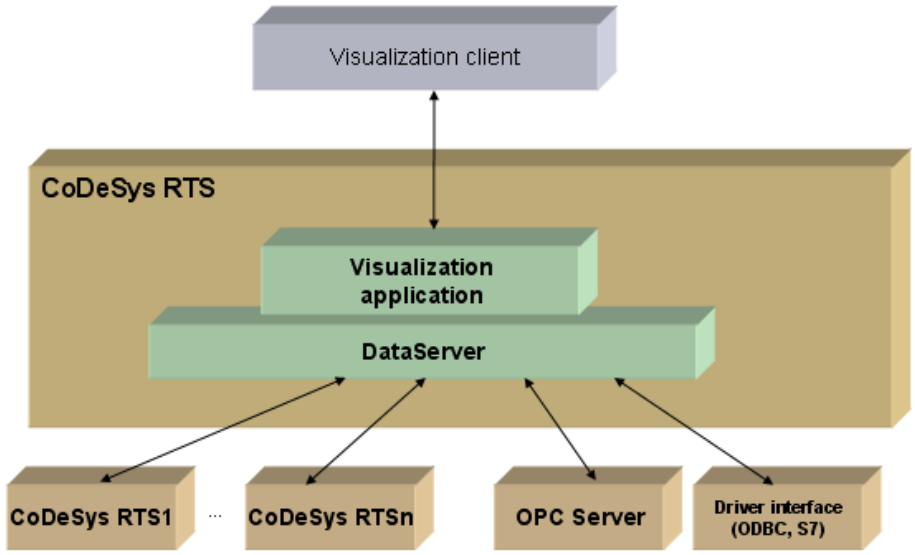
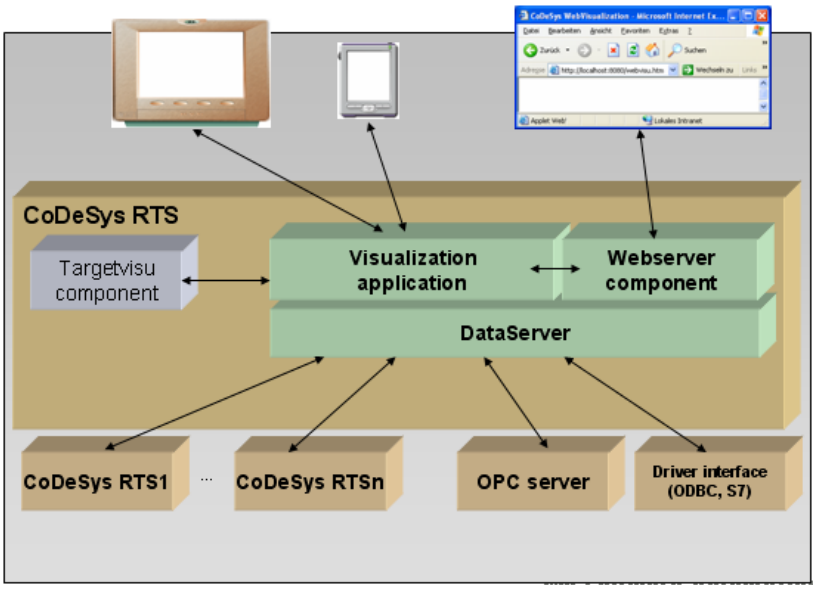


Abb. 44 Data Server for providing visualization data

Variant modes of visualization

A visualization application, i.e. the visualization elements are created in IEC code in the visualization editor of the programming system. In case the visualization is running on the device, visualization code will be generated and loaded there. The four possible visualization modes are described in the following. Which of them is used, depends on whether at all and which type(s) of client object are inserted below the Visualization Manager.



Target-Visualization combining PLC and visualization on one device.

»PLC Designer« HMI to run the visualization on an external device.

Web-Visualization using a web server to connect to the related application.

Diagnosis Visualization: visualization running in the programming system, corresponding to a pure online visualization in »PLC Designer« V2.x.

All visualizations share the following characteristics:

- Within a visualization arbitrary expressions, even function calls, are valid.
- In contrast to »PLC Designer« V2.3 all visualization elements, so also the visualization objects themselves, are implemented as IEC61131-3 function blocks. So visualizations can be instanced and thus be referenced in other visualizations and be used to extend other function blocks. A placeholder concept is realized by handling placeholders as input variables of a visualization. They can be edited in an interface editor. The use of Frame elements makes possible to reach easy toggling between several visualizations within one visualization object.
- A visualization can be created as an object in the global object pool, that is the POUs window, or in the Devices window, directly assigned to an application.
- Visualizations below an application are managed by a Visualization Manager, handling common settings for all application-specific visualizations.
- Some general settings for all visualizations of a project (file paths, size fit etc.) can be done in the Visualization options.

- Each particular visualization has own properties like its designated use ("visualization", "numpad/keypad" or "dialog") or the display size. Notice in this context, that a visualization might be created and configured explicitly for being used as user input dialog in other visualizations. Implicitly also a standard numpad and a keypad mask are available for this purpose. The use of such key pads and dialogs can be defined in the input configuration of a visualization element.
- The visualization editor is assisted by a toolbox providing the available visualization elements and a properties editor for the configuration of the inserted elements. The elements are provided via appropriate libraries according to the currently active visualization profile. The visualization elements can be easily arranged and grouped. »PLC Designer« V2.3 visualizations can be imported.
- The visualization elements can be animated by direct use of the »PLC Designer« project variables, which also might be entered in the form of expressions, that is combined with operators and constants. For examples this allows to scale the variables for the usage in the visualization.
- Language switching (ANSI or UNICODE) within a visualization is possible by use of textlists generated in »PLC Designer«.
- The variables of an application can be entered (written) or displayed (read) in the visualization in a text field element. The formatting of this input and output bases on the standard function sprintf (C-Library).
- External data sources can be used.
- Each element can get assigned a tooltip.
- Besides the zoom function also an automatic scaling (visualization size matches screen size) of the visualization is possible.
- Visualizations can be stored in libraries and thus be made available for use in other projects.
- VisualElement Repositories are used to manage the visualization elements and profiles available on the local system.

Note: The visualization works with the help of an integrated runtime system, so during primary editing actions you will get messages on starting and downloading.

8.2 Preconditions

A project which should contain a visualization basically needs the same structure as a project without visualization. So it must provide a Device definition including Application and Task Configuration and a program.

Due to the fact that components of the »PLC Designer« V3.x Visualization are realized according to IEC61131-3 standard, appropriate libraries must be included, providing the necessary functions and visualization elements. For specifying the selection of visualization libraries and further on the exact selection of visualization elements which should be provided in the visualization editor, visualization profiles are used. For this reason each visualization project must base on a profile.

8.3 Options

In the Options dialog in category "Visualization" among others you can define defaults for the project settings, e.g. for the visualization text and image file directories to be used when configuring a visualization.

For a detailed description see Project Options, category Visualization.

8.4 Creating a visualization object

To create a visualization object use command Add Object. A visualization can be added in the POUs window (global object pool) or application-assigned in the Devices window depending on whether it is intended for general use within the project or for special use on a certain application.

When the first visualization object is added in the project, the respective libraries will be added automatically in the Library Manager according to the currently active visualization profile.

When the first visualization is inserted below an application, automatically the Visualization Manager object will be added too.

The new visualization object will be immediately opened in the visualization editor.

8.5 Editing a visualization

A visualization is edited in the Visualization Editor window. The **Visualization Editor** works in combination with the visualization **Toolbox** (providing visualization elements basing on the included element libraries) and the visualization element **Properties** editor (configuration of the visualization elements) which per default both can be accessed via the 'View' menu.

To open a visualization object, perform a double-click on the object entry in the POUs or Devices view resp. use command Edit Object.

For detailed descriptions on how to use see:

- Visualization Editor
- Visualization ToolBox (available visualization elements)
- Visualization Element Properties (configuration of elements)

8.6 Visualization Manager

If a visualization is inserted in the devices tree, i.e. below an application, automatically a Visualization Manager object will be added.

The Visualization Manager handles some common settings for all visualizations assigned to the current application, for example the display size, start visualization, settings for Web- and Target-Visualization).

For details on the particular dialogs please see: Visualization Manager.

8.7 Start Visualization

The "Start Visualization", that is the visualization object which should be opened first after login on the PLC with an application, must be inserted in the Devices window below the respective application object.

If only one visualization is assigned to an application, that will be used as Start Visualization automatically.

If several visualizations are assigned to the same application, the Start Visualization must be explicitly defined in the Visualization Manager. In this case this option will be deactivated automatically for the other visualization objects.

8.8 Frames, References, Interface - Placeholder

The »PLC Designer« V2.3 concept of visualization references and placeholders is substituted by a similar concept in »PLC Designer« V3:

- Basically a visualization can be inserted in another visualization, thus getting "referenced". For this purpose use the **frame** element which can enclose one or several **references** to visualizations. Which visualizations are currently available for getting inserted depends on their positions within the POUs or Devices trees in relation to the current visualization.
- Each visualization can provide an **interface** (to be opened with the Interface Editor command), where you can define input variables like usual for a function block. These inputs work as "**placeholders**". In an instance (reference) of the visualization they must be replaced by values or expressions for the specific usage in the local object.
  The replacement is to be done in the Properties view of the frame element containing the visualization instance. Mind that the inputs of a visualization instance assigned to a certain application must be assigned to valid application variables.
- **Toggling** visualizations: If a frame element includes several visualizations (Frame Configuration), you can configure the inputs on further visualization elements in a way that they each effect the display of a particular visualization in the frame. The Input Configuration for this purpose provides the option 'Switch Framevisualization'.
  Example: a visualization contains three buttons 1,2,3 and a frame having got assigned the visualizations Visu1, Visu2 and Visu3. The buttons are configured in a way (input: OnMouse-actions: change frame visualization) that activating a button will display the assigned visualization. Thus - in contrast to »PLC Designer« V2.3 - in one and the same visualization element various other visualizations and a "switch board" can be displayed.

8.9 Text and Language in Visualization

Text can be assigned to a visualization element by appropriate definitions in the Properties window or directly ("inline") in the editor window. For the latter a text input field can be opened by pressing the <spacebar> when the respective element is selected.

Text lists are used to manage the texts. A text list is a POU containing text strings which are uniquely (within the project) referenced by the combination of the textlist name and the list-internal text string ID. Different language versions of a text can be defined there and so each text additionally is marked by a language identifier. At least a default language is defined for each text.

Note: Directories providing image files for the usage in a visualization are specified in the Project Settings, Category Visualization.

In the properties of a visualization element you specify a textlist name and an ID (string) and thus determine which text should be displayed in the element in online mode. You can additionally influence the text display by using formatting sequences (see below). Due to the fact that the ID can be specified via a variable, **dynamic switching of texts** is possible.

If the currently used textlist(s) provide several language versions of a text, the language to be used in the visualization **can be switched** by the help of visualization elements which are configured for that purpose. That means they must have an input property defining a "Change the language" mouse-action.

Thus dynamic use and multi-language support is possible for visualization texts.

Basically there are two types of text in a visualization:

1. Static text:

Static text assigned to a visualization element cannot be changed in online mode (just the language, if multiple language versions are available in the textlist, always will be adapted to the language currently used in the visualization). Static text is defined in category 'Texts' in the Properties dialog of the visualization element and is managed in an automatically created textlist named 'GlobalTextList'.

2. Dynamic text

Dynamic text can be changed in online mode by the input of a variable or a user input. It is managed in Textlists which must be created by the user. Dynamic text is assigned to a visualization element in category 'Dynamic texts' in the Properties dialog of the element. There one of the available dynamic textlists as well as the ID of the desired text are to be specified.

Formatting of text

Besides the input of a pure text string you also can use formatting sequences to determine the text display in online mode. A formatting sequence always consists of a "%" followed by a character. It can be used singly or in combination with a pure text string.

If you include "**%s**" in a text string, then this location in online mode will be replaced by the value of the variable which is specified in the 'Text variables' property 'Text variable'. If you want to display the instance name of a variable being passed to a visualization function block, you have to make use of the pragma attribute 'parameterstringof' for an example explaining the details.
Notice that you can use any formatting string conforming with the standard C-library function 'sprintf', you just have to make sure that it fits to the type of the used variable.

See here a sample list of strings and arguments:

Character	Argument / Output as
d,i	Decimal number
b	Binary number
o	Unsigned octal number (without leading zerol)
x	Unsigned hexadecimal number (without leading 0x)
u	Unsigned decimal number
c	Single character
s	String: This location in online mode will be replaced by the value of the variable which is specified in the 'Text variables' property 'Text variable'.
f	REAL-values; syntax: `%|<alignment><minimal width>.<accuracy>|f` The alignment is defined by a minus-sign (left aligned) or a plus-sign (right aligned, default); accuracy defines the number of places behind the comma (default: 6); see example below.

Example:

Entry in 'Text' property: `Fill level %2.5f mm`

Input in the 'Text variable' field (variable of type REAL, providing the fill level value), for example: `plc_prg.fvar1`

-> Output in online mode for example: `Fill level 32.8999 mm`

Note: If you want to get displayed a percent sign % combined with one of the formatting strings mentioned above, you must enter "%%". For example: Enter "Rate in %%: %s" to get displayed in online mode "Rate in %: 12" (if the text display variable currently is "12").

Support of Unicode format can be activated. See the description on Textlists for detailed information.

System time output

If you enter "**%t**", followed by a sequence of special placeholders in squared brackets, then in online mode this location will be replaced by the system time. The placeholders define the display format, see the table below.

Syntax for an entry in the 'Text' property field: %t[<placeholders separated by "." resp. ":">].

Example: %t[dd.MM.yyyy HH:mm:ss] will create an output of the date and time like for example: 12.10.2009 08:22:03

Note: Import of V2.3 projects: The time formattings %t, which are used in »PLC Designer« V2.3 will automatically get converted to the new %t[] format when importing an old project; however the following placeholders are not supported any longer: %U, %W, %z, %Z.

Valid placeholders:

ddd	Name of the weekday, abbreviated, for example "Wed"
dddd	Name of the weekday, for example "Wednesday"
ddddd	Weekday as decimal number (0 – 6; Sunday is 0)
MMM	Name of the month, abbreviated, for example "Feb"
MMMM	Name of the month, for example "February"
d	Day of month as number (1 – 31), for example "8"
dd	Day of month as number (01 – 31), for example "08"
M	Month as number (1 – 12), for example "4"
MM	Month as number (01 – 12), for example "04"
jjj	Day of the year as number (001-366), for example "067"
y	Year without specifying the century (0-99), for example "9"
yy	Year without specifying the century (00-99), for example "09"
yyy	Year with specifying the century, for example "2009"
HH	Hour, 24-hours format (01-24), for example "16"
hh	Hour, 12-hours format (01-12), for example "8" for 16 o'clock
m	Minutes (0-59), without preceded null, for example "6"
mm	Minutes (00-59), with preceded null, for example "06"
s	Seconds (0-59), without preceded null, for example "6"
ss	Seconds (00-59), with preceded null, for example "06"
ms	Milliseconds (0-999), without preceded null, for example "322"
t	Identifier for the display in 12-hours format: A (hours <12) resp. P (hours >12), for example "A" in case of 9 o'clock in the morning

tt	Identifier for the display in 12-hours format, AM (hours <12) resp. PM (hours >12), for example "AM" in case of 9 o'clock in the morning
''	Text strings containing one of the above listed placeholders must be included in single quotation marks; all other texts within the format string can be used without quotation marks; for example 'update', because it contains "d" and "t"

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`

Font

The font to be used for the online display of the text also can be defined in the Properties of the visualization element. See in this concern the categories 'Text properties' and 'Font variables'.

Alignment

The horizontal and vertical alignment of the text is defined in the 'Text properties' of the visualization element.

8.10 Images in Visualization

An image can be inserted in a visualization by the 'image' element.

Note: Directories providing text list files for the usage in a visualization are specified in the Project Settings, Category Visualization.

Images are managed for this purpose in **Image Pools**. These are POUs containing image files which are uniquely (within the project) referenced by a combination of the image pool name and the list-internal ID.

In the Properties of an image element you specify the ID and (for unique accessing) the image pool name to determine which image should be used. Due to the fact that the image reference can also be specified via a variable, **dynamic switching of images** is possible.

Further on a background image can be specified for the complete visualization in the Background dialog (command 'Background').

8.11 External data sources

External (remote) data sources might be used in visualizations. For this purpose a Data Server must be available for the local application, handling the respective data sources.

8.12 Visualization Libraries

Due to the fact that visualization elements in »PLC Designer« V3.x are implemented as function blocks, they are provided via libraries.

When adding a visualization object in a project, a certain set of visualization libraries will be added to the Library Manager in the POUs view. The names as well as the versions of these libraries are defined by the currently used visualization profile. The profile also defines in particular which elements coming from the libraries will be available in the visualization editor ToolBox.

A visualization library always is set up as a special type of "**placeholder library**", which effects that the exact version of the library to be used is not resolved until the library gets included in the project. Only then the currently active profile will define which version actually is needed. Notice that this type of libraries is different from the device-specific "placeholder libraries", where the placeholders get resolved by the device description.

See in the following the basic libraries which are included by default when adding a visualization object in a standard project. These libraries include further libraries, which are not listed here. Usually you do not have to include the visualization libraries manually or to use them explicitly in your applications:

- **VisuElems.library** (basic set of visualization elements)
- **VisuElemMeter** (meter and bar display elements)
- **VisuElemWinControls** (table, textfield, scrollbar elements)
- **VisuElemTrace** (trace element)
- **VisuInputs.library** (handling of inputs on a visualization)

8.13 Visualization Profiles

Each visualization project, that is a project containing at least one visualization object, must base on a visualization profile. This profile defines the following:

- the names and versions of the visualization libraries which get automatically included in the project, when a visualization object is created
- a selection of visualization elements provided by the included libraries which should be available in the visualization editor toolbox

The **profile configuration** is done outside of a project in the visual element repository editor. Multiple profiles might be defined and stored on the local system.

The **profile to be used in a project** by default is defined in the Project Settings (Visualization Profile). There you also can anytime explicitly change to another profile. Notice that the chosen profile is valid for all devices and applications.

If you change the profile, you will get a message that this might prevent a login without online change or without download.

Changing the profile causes an automatic update in the library manager concerning the needed libraries. You will not have to adapt them manually.

A visualization profile always must define at least one **"main" visualization library** (a library provided with the necessary **replacement definitions** for the placeholders which are used in other visualization libraries). In case of the standard installation provided by 3S-Smart Software Solutions GmbH this is by default the main library is **VisuElems.library**.

If you open an old project, which has not yet been created basing on a visualization profile, you will be asked whether you want to switch to the new profile mechanism or not.

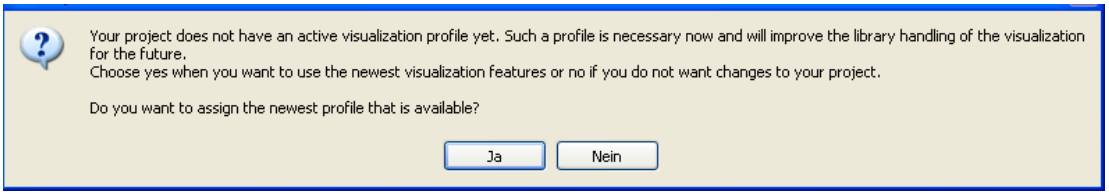


Abb. 45 Message box in case of missing profile

If you answer Yes, the "newest profile" will be applied. If you choose No, the oldest available profile will be entered in the Project Settings (named "**compatibility profile**") but no further changes will be done in the project.

8.14 Target-Visualization

The code created for the visualization objects of the project can be downloaded to the controller.

If it is supported by the target system and if an appropriate monitor is available, the "Target-Visualization" can be started directly on the PLC computer. It is not necessary that the programming system is running in order to run the visualization. This means a considerable reduction of storage use.

Note: If the programming system is running, the visualization can additionally be displayed there. However this is not equivalent to the pure "Diagnosis visualization" mode.

The Target-Visualization will be started automatically on the controller when the application is started there, if::

- a start visualization is defined (see Visualization Manager)
- the name of the application project on the controller is "<application>.app" (this file is automatically created at a download of a visualization project)
- an appropriate runtime system ("with Target-Visu") is started

Note: The VISU_TASK, which automatically gets added to the Task Configuration with the adding TargetVisualization or WebVisualization below the Visualization Manager, will be removed when the last of these client objects was deleted or set to "Exclude from build" in its property settings..

Settings concerning the **start visualization** and the **display** of a Target-Visualization are handled by the **Visualization Manager** in sub-object TargetVisualization.

In case of small target systems consider the option to run the visualization integrated in the programming system.

8.15 Web-Visualization

The Web-Visualization can be used, if there is an **WebVisualization** object inserted below the Visualization Manager object. This is only possible if the **target system** supports Web-Visualization. In this case a special **Web Server** must be available, enabling the communication between target system and web client. Whether the WebVisualization object is inserted automatically or must be added manually, depends on the device description of the target.

The Web-Visualization is realized as a Java applet which during startup requests the complete visualization paint information from the web server. After that only the paint changes get transferred cyclically. At a **download** of a visualization project all files required for the Web-Visualization are transferred to the subdirectory **/visu** of the target system. This concerns the **Java applet**, the base **HTML-page** (*.htm-file) of the visualization and all **images** needed in the visualization.

Preconditions for a Web-Visualization:

- Target system with appropriate configuration and web server
- Specification of the start visualization and the name of the htm-file to be downloaded (in the Visualization Manager)
- Web browser: IExplorer or Mozilla Firefox; plus Java-VRM as from V1.4

Note: The VISU_TASK, which automatically gets added to the Task Configuration with the adding TargetVisualization or WebVisualization below the Visualization Manager, will be removed when the last of these client objects was deleted or set to "Exclude from build" in its property settings.

Configuration in the programming system

The settings for a Web-Visualization concerning data transfer, display size, start visualization and htm-file must be done in the Visualization Manager in sub-object Web-Visualization.

Configuration of the target system

It might be that the Web-Visualization functionality still must be configured in the target system. The following entries must be valid in the configuration file <targetsystem>.cfg:

```
[ComponentManager]
Component.<n>=CmpWebServerHandlerV3
Component.<n>=CmpWebServer
[CmpWebServer]
FileUploadDirectory=C:\Programme\3S »PLC Designer«\GatewayPLC\visu
```

FileUploadDirectory must describe the path of the runtime system, in which the Web Server is available. "\visu" must be appended to the path. There all files required for the Web-Visualization will be transferred.

Calling a Web-Visualization via Internet

Enter the following address in the web browser:

```
http://<IP_address of webserver>:<port of webserver>/<webvisu>.htm
```

Example:

```
http://localhost:8080/webvisu.htm
```

<webvisu>.htm is the htm-file defined in the Visualization Manager, via which after the call the start visualization - also defined in the Visualization Manager - will be displayed in the browser window. Thereafter the visualization can be operated in the browser.

8.16 Visualization running in programming system

For diagnosis purposes it might be desired to run the application-assigned visualization(s) only within the programming system without the need of downloading visualization code to the device.

This "**Diagnosis Visualization**" as **available from V3.3.2.0** will be used **automatically if** there is no client object "TargetVisualization" or "WebVisualiaztion" inserted in the devices tree below the Visualization Manager. In this case **no visualization code** is created and transferred to the device. This however implies several restrictions as listed below.

If you do not want to remove the TargetVisualization and/or WebVisualization objects below the Visualization Managers' tree in order to get run the diagnosis mode of the visualization, alternatively you might exclude the client objects from build by activating the setting "Exclude from build" in the respective object properties.

Note: If in a project, which originally has been created with a version < V3.3.2.0, a defined compiler version had been set, then the project will behave according to the features of that version! Before V3.3.2.0 the setting "Do not use PLC for visualization" had to be activated in order to get a visualization only run in the programming system.

Numeric values visualized in a diagnosis visualization element ("%s" in the text property), will be displayed according to the currently set **online display mode** (binary, decimal, hexadecimal).

Note: In contrast to the Target-Visualization in the diagnosis visualization VAR_INPUT variables behave like VAR_IN_OUTs.

Restrictions

Expressions, Monitoring:

The diagnosis visualization mode only supports expressions which can be handled by the monitoring mechanism of the programming system. These are:

- normal variable accesses like PLC_PRG.myPou.nCounter
- complex accesses as listed in the following; note that for those a runtime system V3.3 SP2 or newer is required!:
- access on an array of scalar data types with one variable used as index (a[i])
- access on an array of complex data types (structures, function blocks, arrays) with one variable used as index (a[i].x)
- access on a multidimensional array of all sorts of data types with one or several variable indices (a[i, 1, j].x)
- access on an array with constant index [a[3])
- accesses like listed above, in which simple operators are used for 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 like p^.x
- not supported are methods or function calls except the following: all standard string functions, all type conversion functions like INT_TO_DWORD, all operators like SEL, MIN, ...

Inputs:

Within the input action "Execute ST-Code" only a list of assignments is supported.

Example:

```
PLC_PRG.n := 20 * PLC_PRG.m;
// not allowed
IF PLC_PRG.n < MAX_COUNT THEN
   PLC_PRG.n := PLC_PRG.n + 1;
END_IF

//use the following statement instead:

PLC_PRG.n := MIN(MAX_COUNT, PLC_PRG.n + 1);
```

Note: If a list of assignments is used, the value on the left side will not get assigned before the next cycle. A direct subsequent processing in the successive line is not possible.

Visualization interface:

Within the interface of a visualization no type "INTERFACE" may be used.

DataServer:

The Diagnosis Visualization - because it is only designated to inspect the current application - in contrast to the Target-Visualization will not monitor the current values of variables which are provided via the Data Server from another data source! For those just the initial values or the last transferred values will be displayed.

8.17 Visualization running on external devices

By use of »PLC Designer« HMI (HMI=Human Machine Interface) a visualization that has been created with the »PLC Designer« programming system can be executed on a device being allowed to differ from the device, where the visualized application is processed on.

In contrast to the »PLC Designer« HMI integrated in version V2.x it is no longer necessary to execute a download of the complete project. Today the »PLC Designer« HMI is a **runtime system** embracing the client, a visualization application as well as a data server that will update the variables driving the visualization. Hence, the use of »PLC Designer« HMI does not depend on the platform (Windows, CE, Linux, QNX, ...). Because of the internal treatment of the HMI as target visualization it has a high performance.

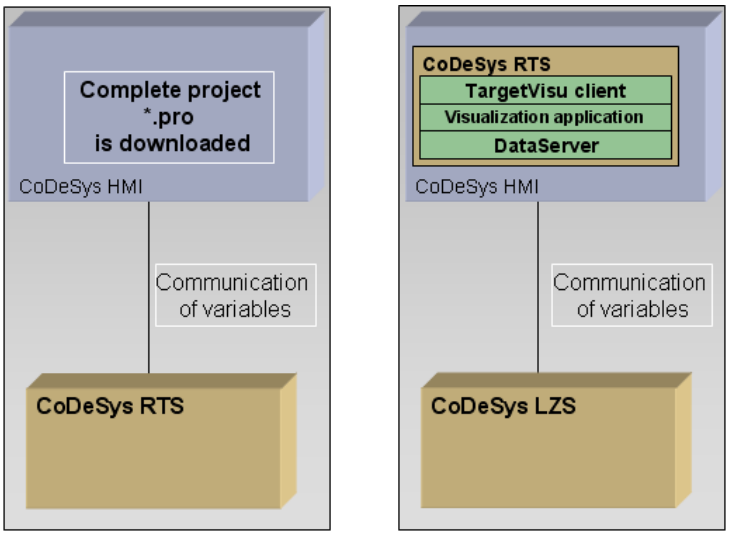


Abb. 46 Concept of »PLC Designer« HMI - Version V2.x (at left) versa Version V3.x (at right)

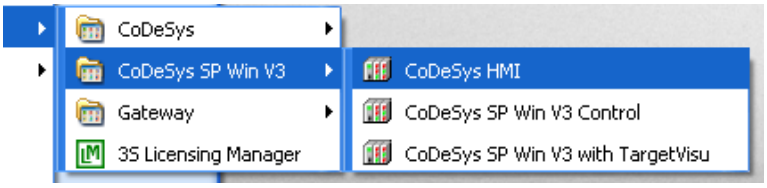
Having started »PLC Designer« HMI on the corresponding device (»PLC Designer« HMI for Win32) the related visualization (specified in the Visualization Manager) will be displayed in full screen mode. With help of the keyboard/mouse the user can handle the control and watch functions being integrated in the visualization. These control and watch functions are realized by visualization elements that can be operated in online mode. Therefore they provide an interface to the visualized application and allow to influence the behaviour of the project assigned.

Installation:

»PLC Designer« HMI is installed together with the standard »PLC Designer« setup. For testing purposes a time-limited demo version is available.

Start:

In principle, the »PLC Designer« HMI is a Target-Visualization and therefore can be executed on arbitrary platforms. In particular, on a Windows platform the »PLC Designer« HMI (»PLC Designer«SPHMI.exe) is started out from the start menu.



Visualization:

To run a visualization on an external device you have to include the **device "»PLC Designer« HMI for Win32"** into your project. This device is a restricted version of the PLC "»PLC Designer« SP Win V3" that is thought for visualization purpose only, but is not able to e.g. handle input/output.

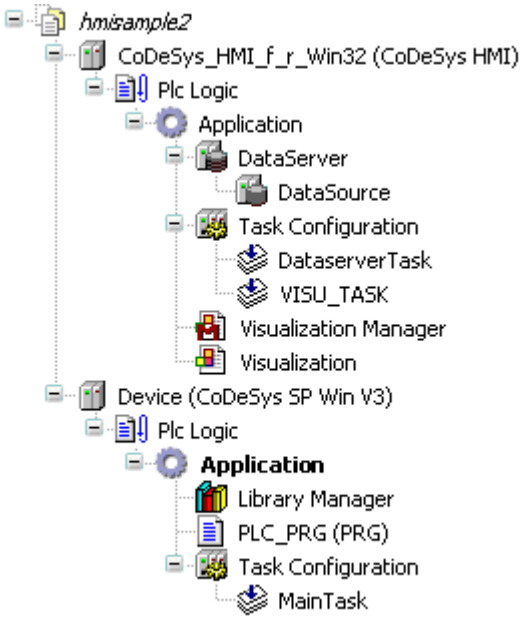
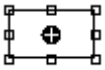
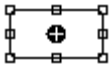
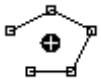


Abb. 47 Example of project making use of »PLC Designer« HMI for visualization

Below the related application a data server has to be configured appropriately to provide the parameters that are contained in the application to be visualized. These parameter values may now enter into the visualization that is implemented in the usual way. Remark that you have to activate the checkbox "Open Target-Visualization" when configuring the associated visualization manager.

8.18 Visualization elements of type Visual

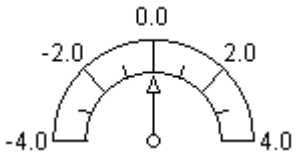
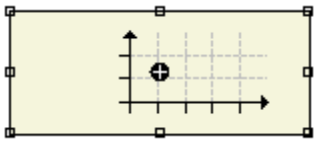
The category "Visual" of the Visual Editor's toolbox contains the following elements:

Rectangle		Rectangle, Rounded Rectangle and Ellipse are of the same element type. They can be converted to each other just by changing the "Type of element" property.
Rounded Rectangle		
Ellipse		
Line		The declination of a Line element can be switched between topleft-to-bottomright to bottomleft-to-topright.
Polygon		Polygon, Polyline and Curve are of the same element type. They can be converted to each other just by changing the "Type of element" property. An additional point, determining the sections and the shape of the element, can be inserted by keeping pressed <CTRL> while clicking on the point. A point can be removed by keeping pressed <SHIFT>+<CTRL> while clicking on the point. In the curve element the points next to the line are handles for forming the bend of the respective curve segment.
Polyline		
Curve		
Frame		A frame (like frames known from e.g. Internet pages) defines a partial area of the current visualization which contains several other visualizations between one can switch in online mode. Basically can be configured like a rectangle element. To define which visualizations should be displayed within a frame, open the 'Frame Configuration' dialog by selecting the frame and perform command 'Frame Selection'. Here all available visualizations will be offered and you can select the desired ones to get them inserted in the frame. It is recommended however to use only visualization from the global POUs pool window in order to avoid problems later in case a device gets renamed. A "filled" frame element then also in offline mode will show the contained visualization(s). Concerning the switching between visualizations regard the possible mouse actions.

Button		A button element can get assigned an image and a defined button-height.
Image		An image element gets assigned an image by specifying the image ID and if needed by the name of the image pool where it is managed. This can be configured dynamically, i.e. via a project variable. See in this concern: Image pools.

8.19 Visualization elements of type Complex Controls

The category "Complex Controls" of the Visual Editor's toolbox contains the following elements:

Meter (not yet implemented)		This element adds a tachometer to the visualization, the minimum and maximum value of which can be defined. The position of the needle is then adjusted to the value of the associated input variable. For particular value ranges background colors may be specified.
Bar Display (not yet implemented)		This element adds a bar display to the visualization, the minimum and maximum value of which can be defined. The length of the bar is then adjusted to the value of the associated input variable. For particular value ranges separate colors may be specified. Per default the bar display will be displayed horizontally; having inserted this element you may change its orientation in a vertical one within the element properties.
Trace		This element allows to integrate a trace within the visualization. The trace to be displayed is specified in the element properties. The configuration of the variables displayed is done within the configuration of the trace itself.

On the basis of examples you will find the elements explained when following the associated link:

- Meter
- Bar Display
- Trace

8.19.1 Configuration of the Meter element

The visual meter element can be added to a visualization by dragging it from the toolbox. Its appearance being designed like a tachometer allows to visualize the actual value of the associated input variable.

See in the following a sample of inserting and configuring a visual meter element:

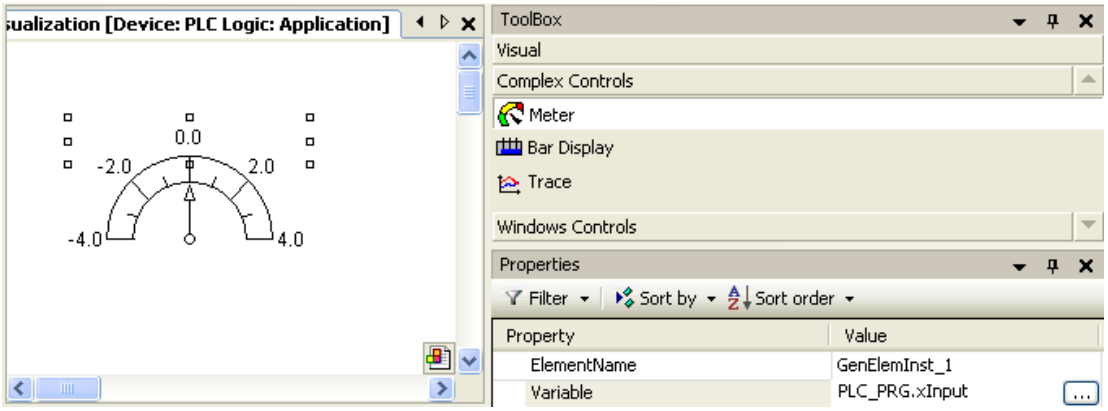


Abb. 48 Example - Inserting a visual meter element and associating the input variable

The associated input variable - a variable named "xInput" in the example - has to be specified within the element properties of the meter element; after clicking in the edit field you will find the button [...], which can be used for selecting the input variable by browsing your project. Take care to indicate the input variable by its entire path within the device tree.

The orientation and size of the scale display as well as shape and color of the arrow can be adjusted within the section "Arrow" of the element properties.

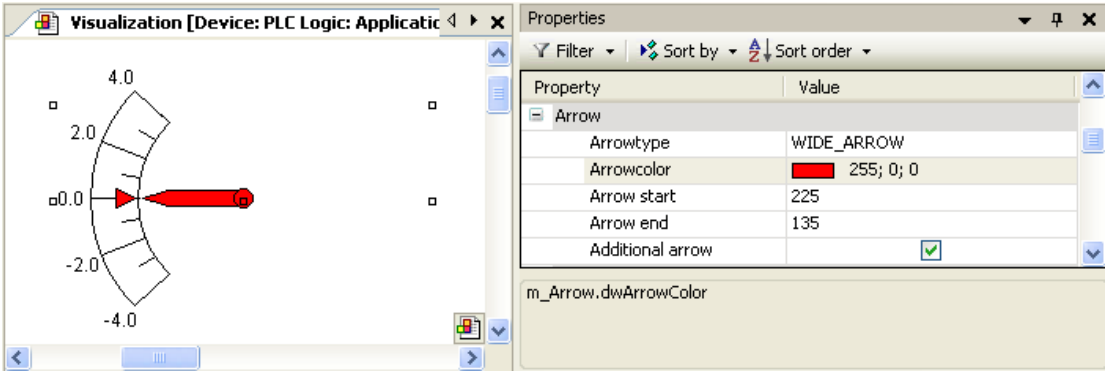


Abb. 49 Example - Configuration within the section "Arrow" of the element properties

The number in the edit field of Arrow start/Arrow end indicate the angle (in degree) that is included by the left/right bound of the scale display and the horizontal line. This angle is oriented mathematically, that is counter clockwise, so that the value in the field Arrow start must always be greater than the value of Arrow end! The pair of start and end value is periodic with 360.

The section "Scale" is provided for defining the numeric range of the scale and subdividing the scale by coarse and fine pitches.

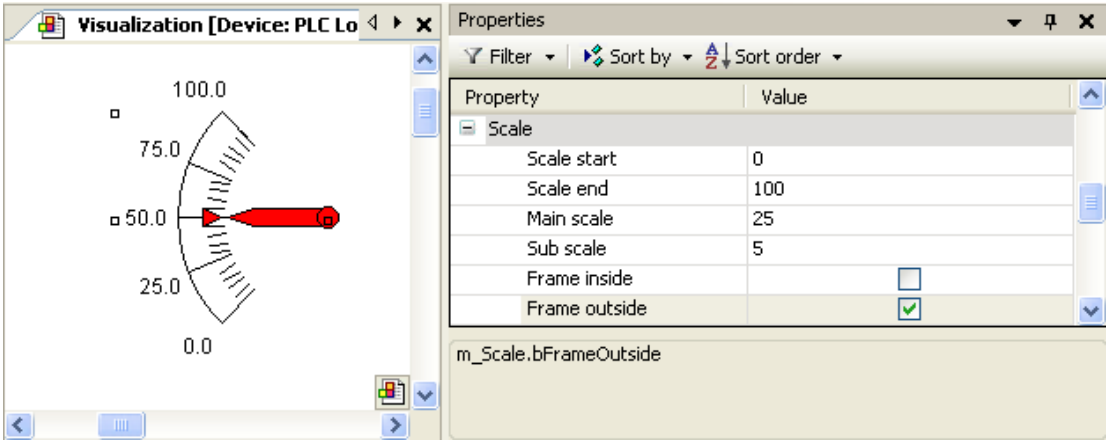


Abb. 50 Example - Configuration within the section "Scale" of the element properties

The value of scale start will be inserted at the left bound of the scale. Thus it has to be less than the value of scale end that will be entered at the right bound of the scale. In contrast to the main scale the finer sub scale may be omitted by setting the value of its fragmentation to 0. In that case no small scale pitches would be displayed. As we deactivated the checkbox "Frame inside", the inner arc of the scale display has disappeared in our example.

After defining the scaling range we have to design the display of the scaling labels within section "Label".

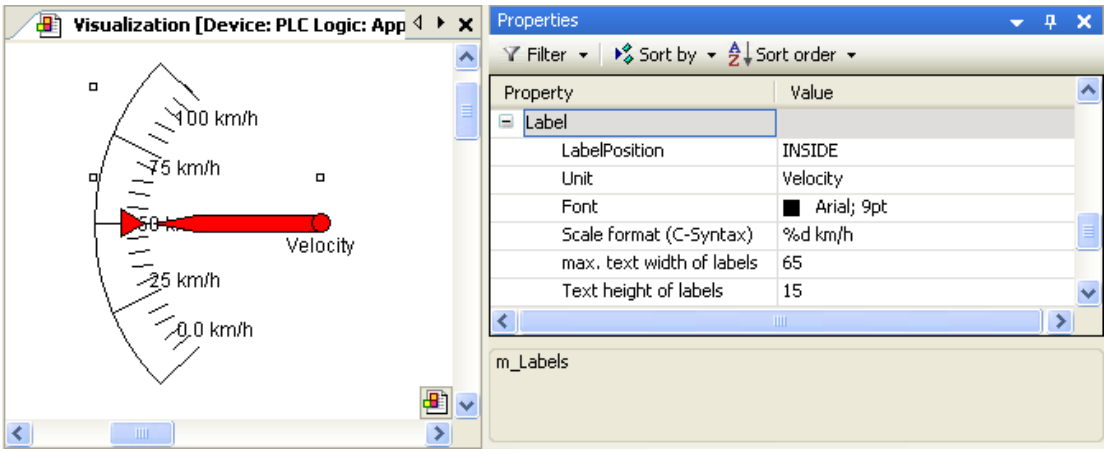


Abb. 51 Example - Configuration within the section "Label" of the element properties

By changing the "LabelPosition" from OUTSIDE to INSIDE we cause the labeling to be displayed at the inner side of the scaling arc. The entry in the edit field "Unit" is displayed below the root point of the arrow. Having chosen the appropriate (colour of the) font, we adjust the format of the labels: the numeric scale values have to be formatted according to the Syntax of the programming language C (use %d for integers and %.Xf for floating points, where X has to be replaced by the number of decimal places). The entries in the following two edit fields are adjusted to your preceding settings of this section. You only have to modify them in case, the automatic adjustment does not lead to the desired result.

Finally, within the section "Colors", we can color particular parts of the scale range by creating color areas with help of the button Create new. The color areas are numbered consecutively in ascending order. Each color area will be provided with own edit field within the element properties.

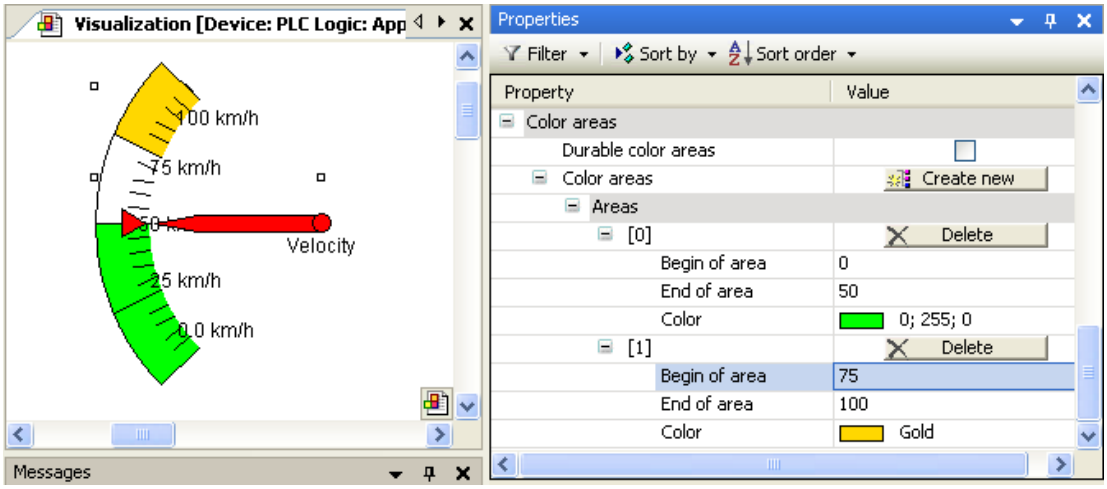


Abb. 52 Example - Configuration within the section "Colors - Color areas" of the element properties

Begin and end of the area are provided to specify the subrange of the scaling range. The coloring can be selected from an drop-down menu and a color area once created can be deleted by clicking on the associated button Delete.

The effect of the checkbox "Durable color areas" is visible in online mode only:

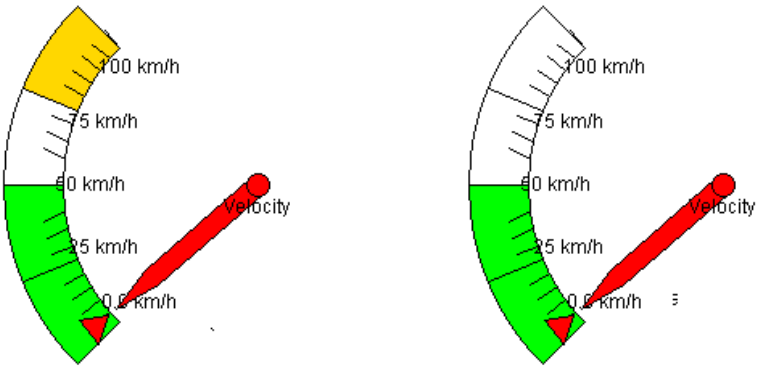


Abb. 53 Example - Meter element in online mode

While the meter element at right only shows the color area the arrow is located in, the meter element at left shows all designed color areas because of the checkbox "Durable color areas" being activated.

8.19.2 Configuration of the Bar Display element

The visual bar display element can be added to a visualization by dragging it from the toolbox. Its appearance being designed like a thermometer allows to visualize the actual value of the associated input variable.

See in the following a sample of inserting and configuring a bar display element:

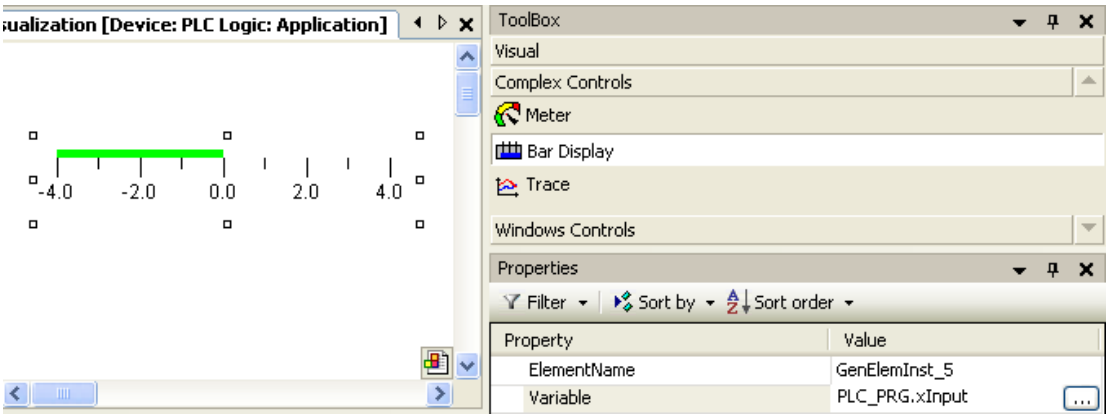


Abb. 54 Example - Inserting a bar display element and associating the input variable

The associated input variable - a variable named "xInput" in the example - has to be specified within the element properties of the bar display element; after clicking in the edit field of property "Variable" you will find the button ..., which can be used for selecting the input variable by browsing your project. Take care to indicate the input variable by its entire path within the device tree.

The orientation and position of the bar with respect to the scale can be adjusted within the section "Bar" of the element properties.

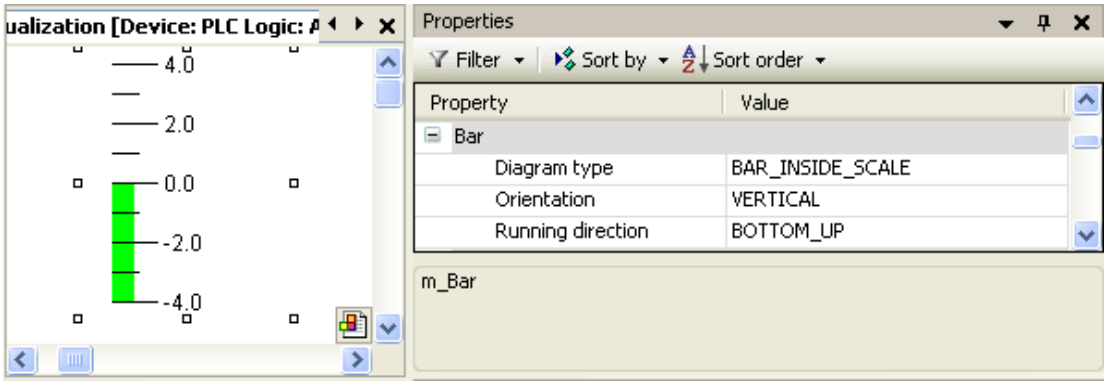


Abb. 55 Example - Configuration within the section "Bar" of the element properties

In the example the orientation of the bar was changed from HORIZONTAL to VERTICAL. This modification entails a switch of the possible entries for Running direction: instead of LEFT_RIGHT and RIGHT_LEFT we now have to chose between BOTTOM_UP (for an ascending scaling) or TOP_DOWN (for descending scaling). The position of the bar in respect to the scaling is specified by the Diagram type.

The section "Scale" is provided for defining the numeric range of the scale and subdividing the scale by coarse and fine pitches.

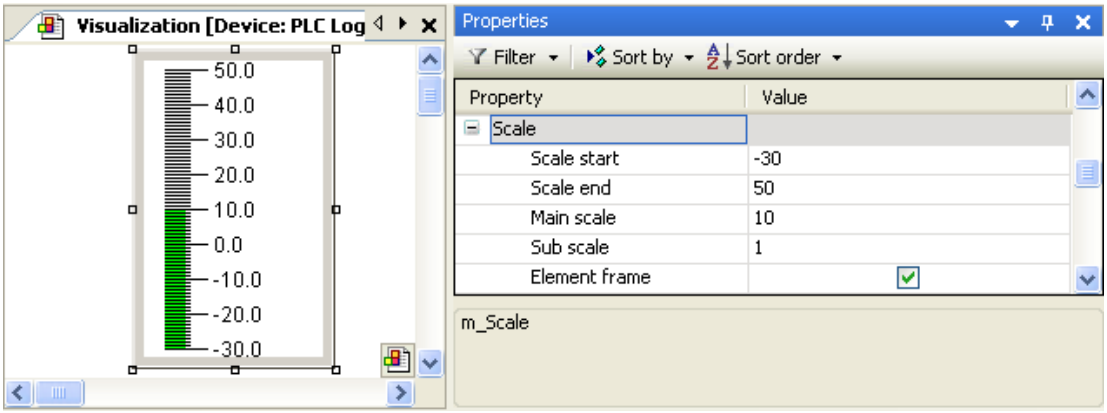


Abb. 56 Example - Configuration within the section "Scale" of the element properties

The range of the scale is limited by the Scale start from below and by the Scale end from above. Thus the start value has to be less than the end value. Both of the distances in Main and Sub scale may be set to 0 for disabling the display of the scaling: if the coarse scaling distance is set to 0 there will be no scaling displayed at all (regardless the setting made for the sub scale). If the sub scale is set to 0, only the coarse scale pitches will be drawn. As we want the bar display to be framed, we activate the checkbox "Element frame".

The section "Label" is provided for labeling the display bar:

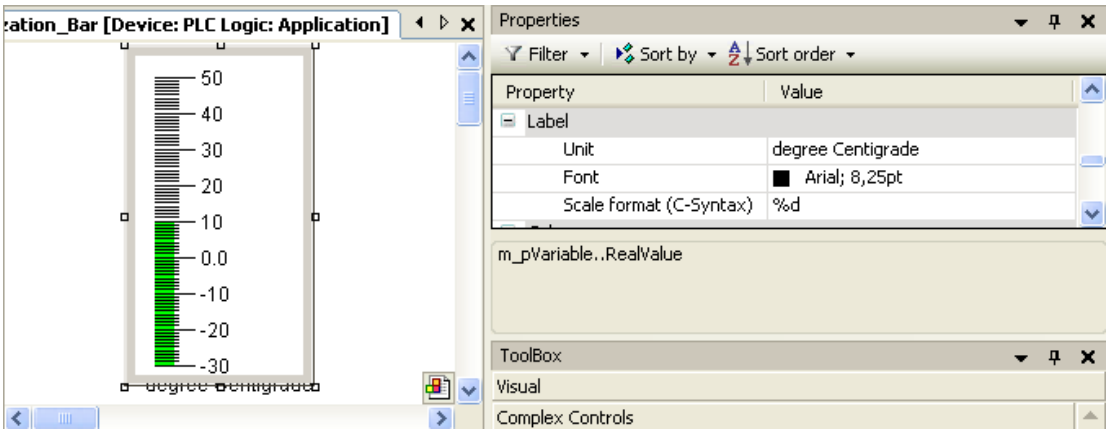


Abb. 57 Example - Configuration within the section "Label" of the element properties

The entry in the edit field "Unit" is thought for indicating the unit used within the scaling and displayed centered below the display bar. Having chosen the appropriate (colour of the) font, we adjust the format of the labels: the format for numeric scale values has to be indicated according to the Syntax of the programming language C (use %d for integers and %.Xf for floating points, where X has to be replaced by the number of decimal places).

Finally we may adjust the coloring of the element within the section "Colors":

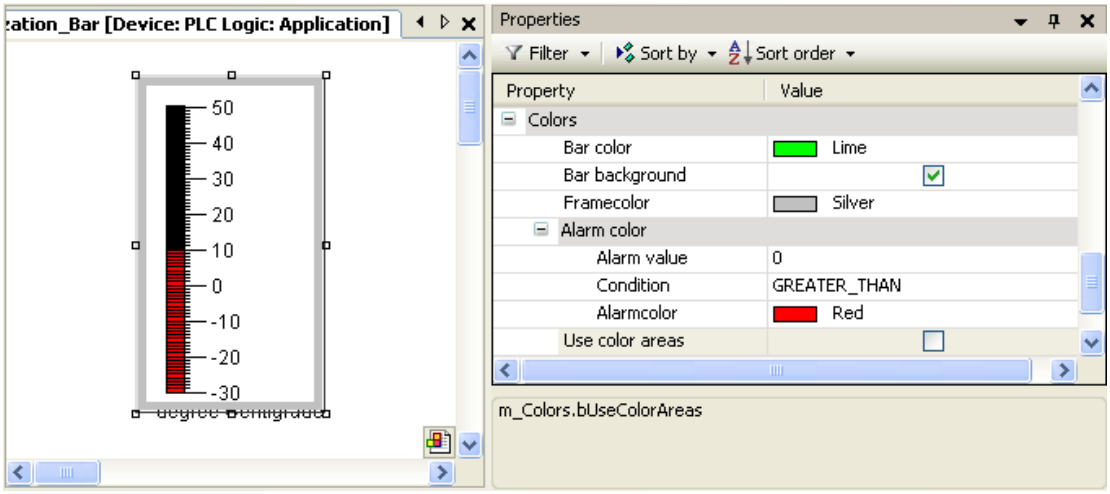


Abb. 58 Example - Configuration within the section "Colors" of the element properties

First of all the bar color may be selected. By default the background of the bar, that is the part of the bar line that is actually not filled out by the bar, is white. By activating the checkbox "Bar background" the background color can be changed into black. Furthermore the color of the frame can be specified.

Within the subsection "Alarm color" an Alarmcolor may be specified the bar will be displayed in if the actual value of the input variable fulfills the alarm condition. This condition results from defining an Alarm value, that may either not be exceeded (set Condition to GREATER_THAN) or not be underrun (set Condition to LESS_THAN).

These settings have no more influence, if so called color areas are specified and enabled by activating the checkbox "Use color areas":
Parts of the scale range can then be given a specific color by creating color areas with help of the button Create new. The color areas are numbered consecutively in ascending order. Each color area will be provided with own edit field within the element properties:
Begin and end of the area are provided to specify the subrange of the scaling range.
The coloring can be selected from an drop-down menu and a color area once created can be deleted by clicking on the associated button Delete.

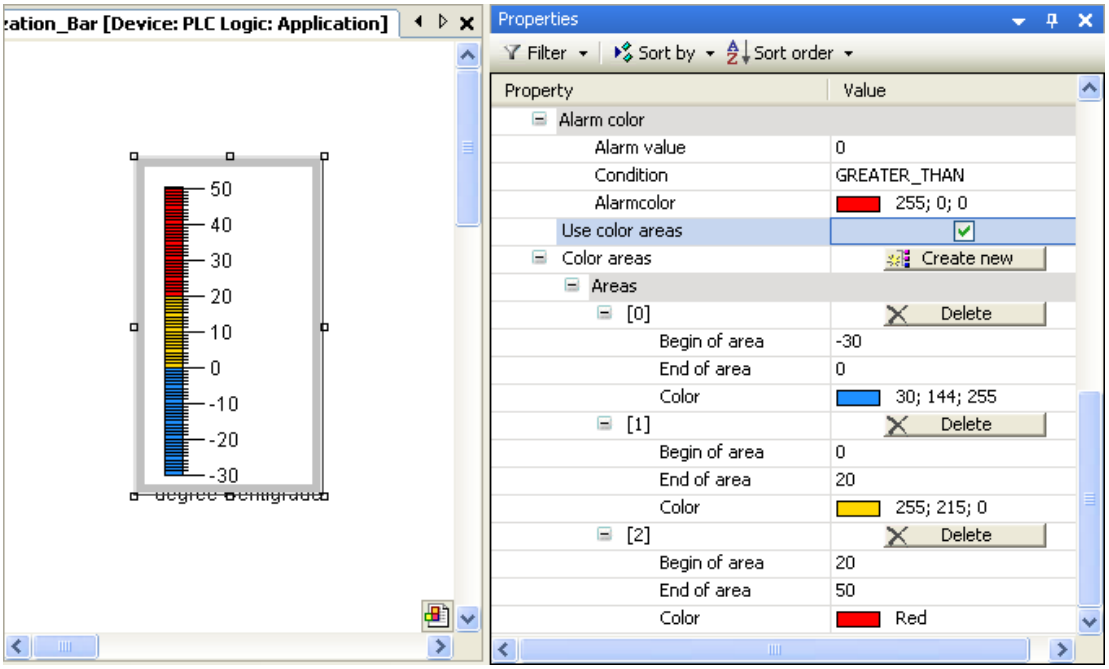


Abb. 59 Example - Configuration within the section "Colors - Color Areas" of the element properties

8.19.3 Configuration of the Trace element

To be able to display traced variables within your visualization, you first have to insert an appropriate trace object within your project. Therein, you configure the variables to be traced, an eventual trigger variable as well as the trace itself (axis, refresh rate, etc.). The option "Generate Trace POU for visualization" within the trace configuration will provoke the implicit generation of a component named "<Tracename>_<Recordname>_VISU". This option has to be activated, in order that the trace may serve as input of a visual trace element.

The visual trace element can be added to a visualization by dragging it from the toolbox.

See in the following a sample of inserting and configuring a visual trace element:

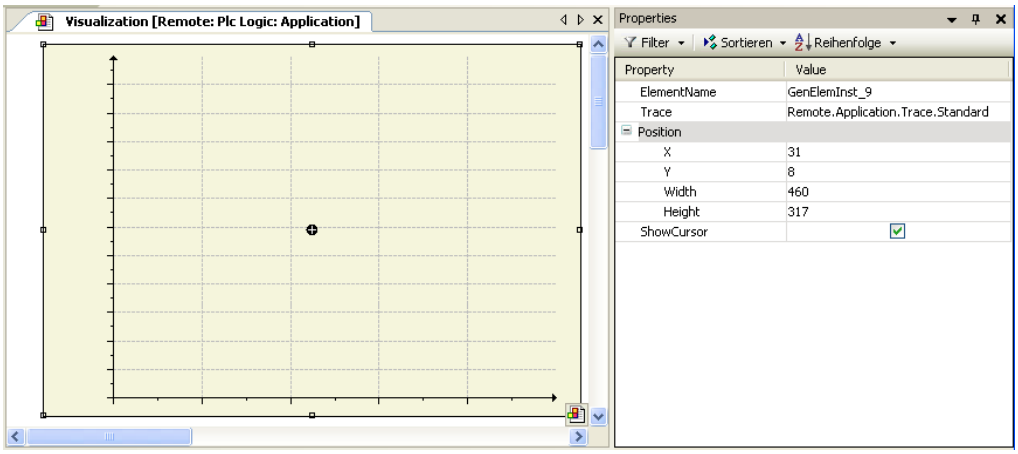


Abb. 60 Example: Inserting and configuring a visual trace element

The associated trace object is specified within the element properties of the trace element; take care to indicate the trace by its entire path within the device tree. See BELOW for the description of the option "ShowCursor" in the element properties.

To be able to control the trace visualized in online mode, the addition of further visual elements might be reasonable. The following template includes four buttons to handle the trace behaviour within the visualization:

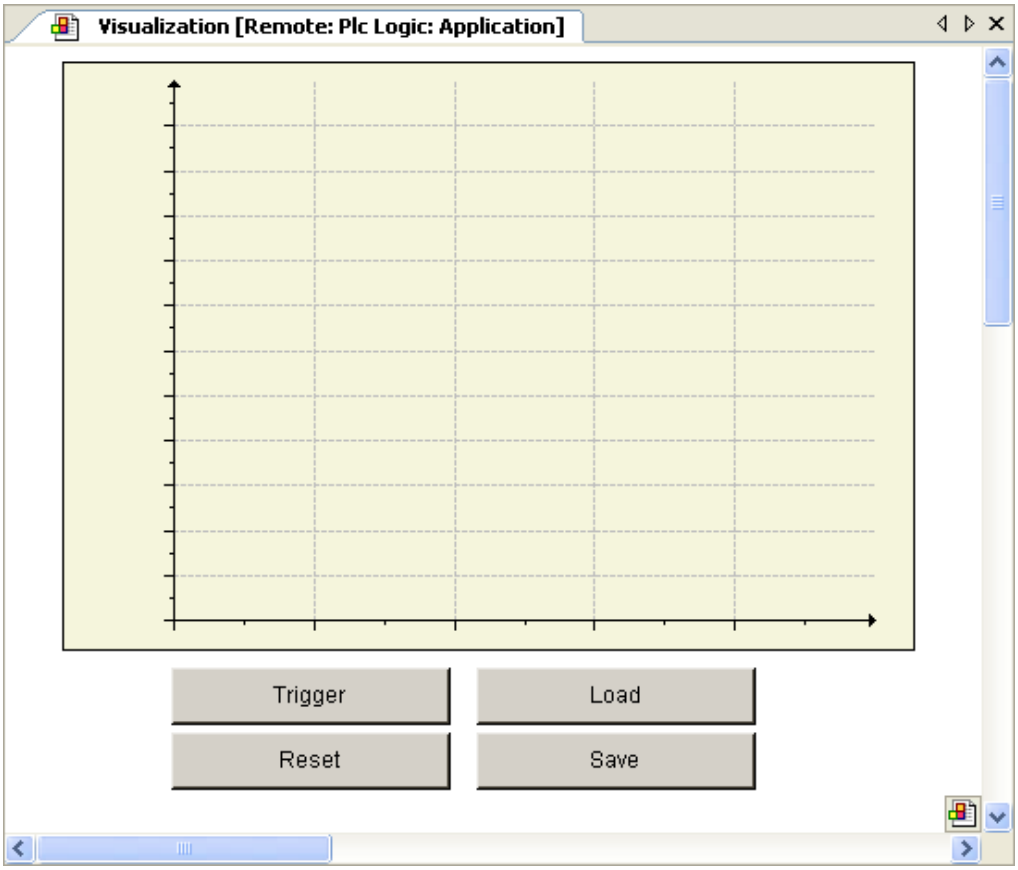
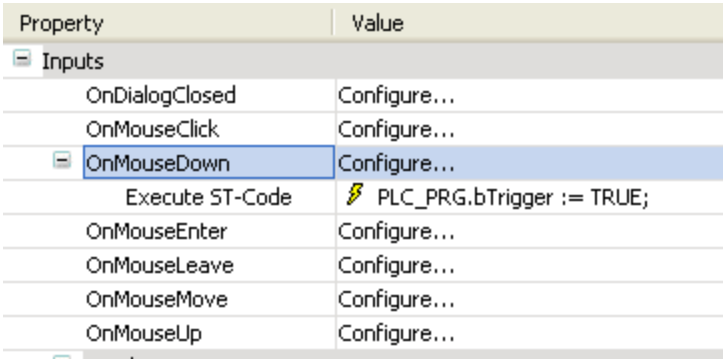


Abb. 61 Example: Additional elements to handle the visualization

The Buttons entitled "Trigger" activates the triggering of the trace, i.e. setting the associated trigger variable to TRUE. This is done within the item "inputs" of the element properties of the button:



The special action "OnMouseDown" is connected to the execution of a ST-Code setting the associated trigger variable to TRUE, here "PLC_PRG:bTrigger:=TRUE;". Likewise the button "Reset" is connected with a reset of the trace by setting the trigger variable to FALSE again (here "PLC_PRG.bTrigger:=FALSE;") and enabling a reset (e.g. "PLC_PRG.bReset:=TRUE;").

The third button entitled "Load" allows a modification of the trace displayed without having to logout from the corresponding application. Therefor, the attached ST-Code to be executed includes not only setting the trigger variable to "FALSE", but also enables the boolean variable bLoad being part of the implicit generated component "<Tracename>_<Recordname>_VISU" and controlling the load of a trace configuration file, e.g. "Trace_MainTask_VISU.bLoad:=TRUE;". Setting this variable to TRUE involves the loading of a trace configuration file (with default ending .tcg), which has to be stored in the directory of the runtime system. The trace visualization (and the trace itself) are adapted to the configuration within this file. See BELOW for a description of the configuration file.

The button "Save" allows to save the actual trace data displayed in the trace visualization. Therefor, the attached ST-Code to be executed includes not only setting the trigger variable to "FALSE", but also enables the boolean variable bSave being part of the implicit generated component "<Tracename>_<Recordname>_VISU", e.g. "Trace_MainTask_VISU.bSave:=TRUE;". Setting this variable to TRUE involves the generation of a data file <Tracename>_<Recordname>.tda within the directory of the runtime system. The data file will contain the actually displayed values of the traced variables and can later be reloaded in a trace.

The following picture shows the trace visualization in online mode after the trigger variable has been activated by pressing the button "Trigger":

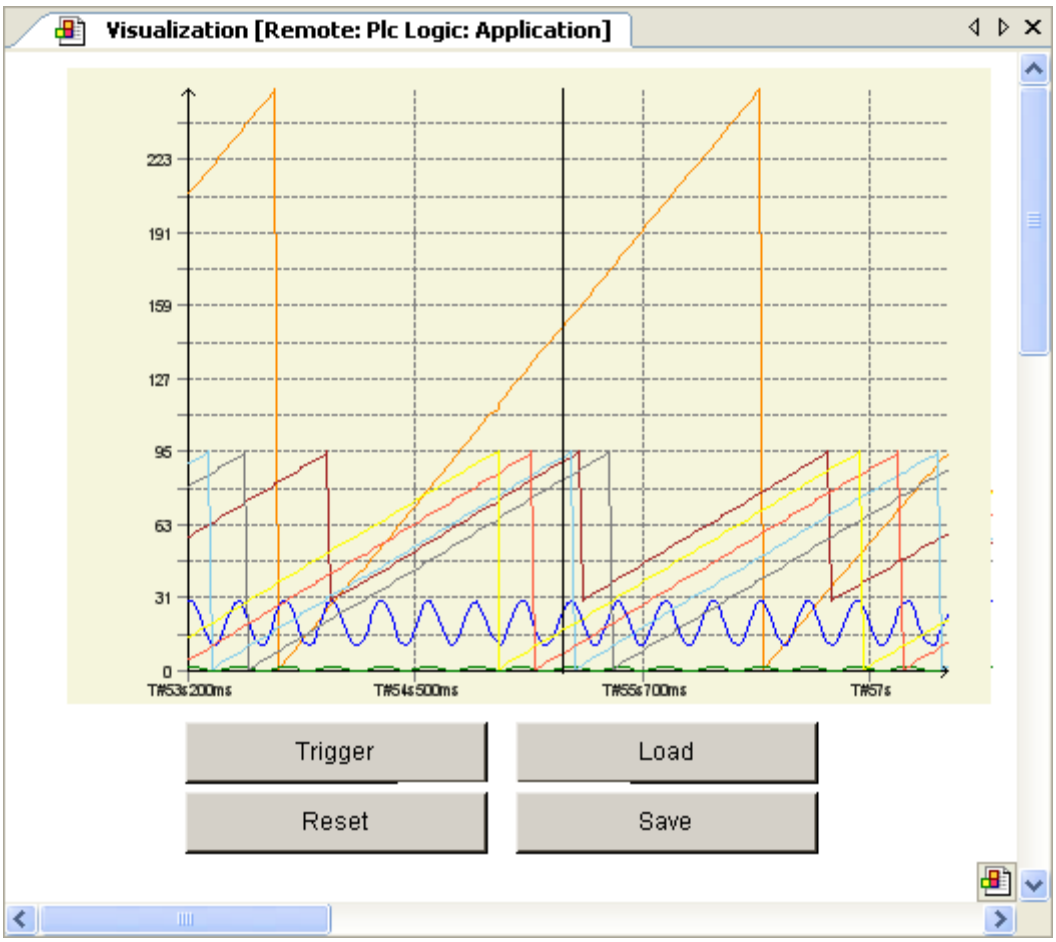


Abb. 62 Example: Trace in online mode

The edge of the trigger variable is marked by a black vertical line. Recording of variables continues for the specified percentage (TriggerPostion%) of the whole tracing interval, afterwards the visualization is paused.

If the option "ShowCursor" in the element properties of the visual trace element is activated, you are now able to get within a tooltip the values of the trigger and the traced variables displayed at a position marked by the position of the cursor in the visual trace element.

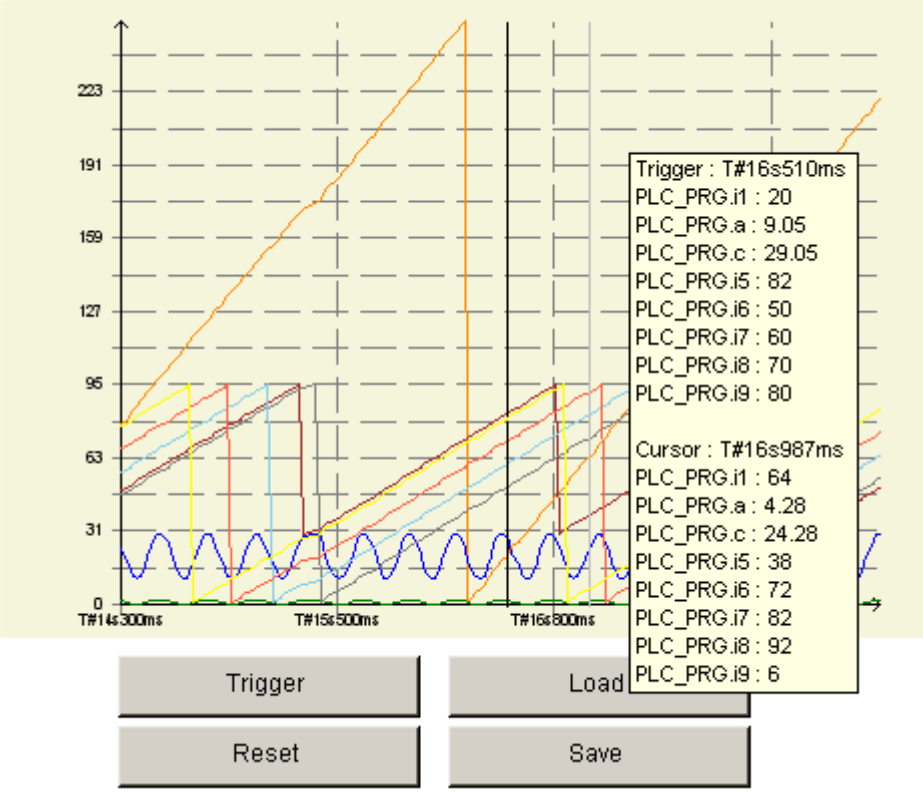


Abb. 63 Example: Effect of option "ShowCursor"

Trace Configuration File

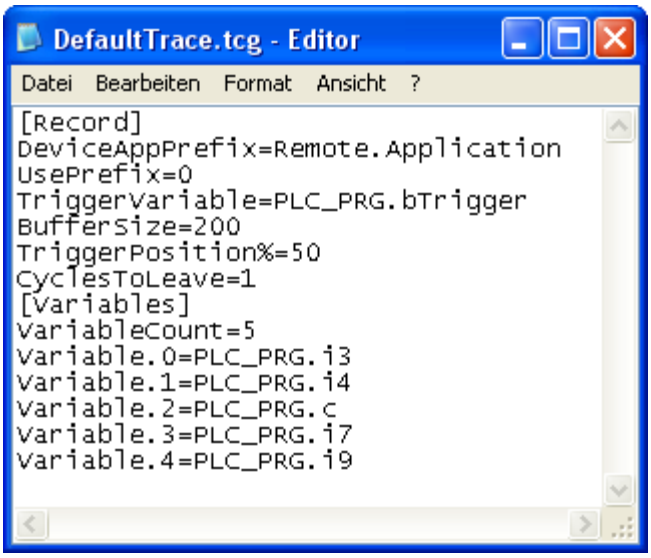


Abb. 64 Example: Trace Configuration File

The trace configuration file <Tracename>_<Recordname>.tcg has to be stored in the directory of the runtime system, e.g. in the directory "C:\Programs\3S »PLC Designer«\Gateway PLC".

[Record] Part:

Setting the "DeviceAppPrefix" to the common part of the path of the traced variables spares yourself the trouble of indicating the entire name (including the path) of each variable. This prefix will be visible within the tooltip, if "UsePrefix" is set to 1. Otherwise, if "UsePrefix" is set to 0, the prefix will be hidden. The parameter "CyclesToLeave" corresponds to the parameter "Measure in every <x>-th cycle" of the trace configuration.

[Variables] Part

The number of the variables to be traced is specified by "VariableCount". It can be maintained or reduced in comparison with the original trace. There must not be more variables than in the original trace! All variables to be traced have to be contained in a symbol configuration; alternatively, they can be declared via the data source.

8.20 Visualization elements of type Windows Controls

8.20.1 Configuration of the Table element

A table element can be inserted in a visualization in order to visualize one- or two-dimensional arrays, structures or the local variables of an POU.

The configuration of the table is done in the **visual element properties** view, please see the corresponding help page for detailed information on each property. In the following some introductive examples of configuring a table element are presented.

First of all the variable to be visualized must be specified. For this purpose in the element properties enter its complete path in property "Data array". The structure of the variable determines the number of columns and rows of the table.

The **rows** of the table represent the index of the first dimension of the specified array. The **columns** represent the second index of the array in case of two-dimensional arrays or they represent the particular components of a structure or the local variables of a POU. So, for example, a one-dimensional array with elements of basic type will be represented with help of only one column, a structure will be represented with help of one single row, whereas an array with elements of structure type will be represented in a table having as many rows as array elements and as many columns as structure components.

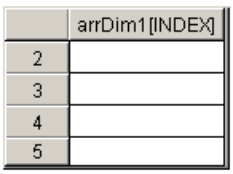
If the dimension of the assigned variable gets modified afterwards, for example by changing the array borders or by adding or removing structure components, the **table can be updated** by entering a <Return> in any of the table fields. This will not change the configuration of components which have remained vaild.

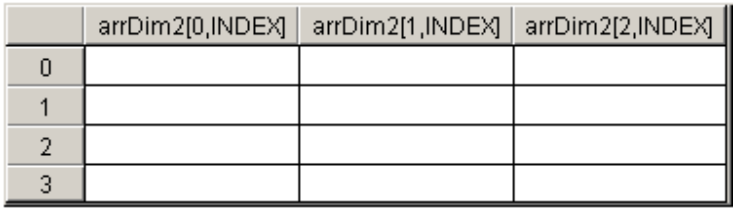
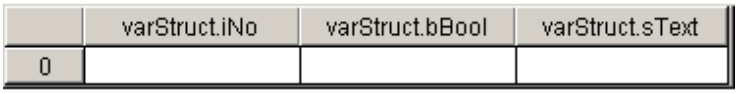
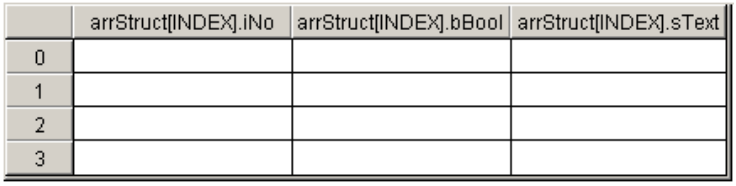
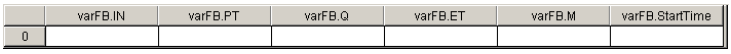
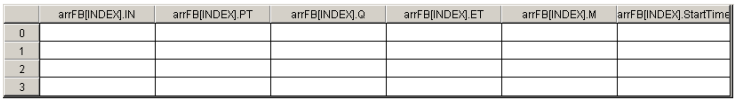
A table **column can be duplicated** by pressing <Ctrl> and - while keeping pressed this key - a mouseclick on the header of the column to be duplicated.

A table **column can be deleted** by pressing <Ctrl>+<Shift> and - while keeping pressed these keys - a mouseclick on the header of the column to be deleted.

Note: If you change the **data array** property of a table element, the other property settings for the element will be reset to default!

Take a look to the following examples:

Declaration of variable	Structure of associated table element
arrDim1 : ARRAY[2..5] OF INT;	 Note that the indices in the row header are equal to the indices determining the first dimension of the array; so the numeration starts with "2" in this example!

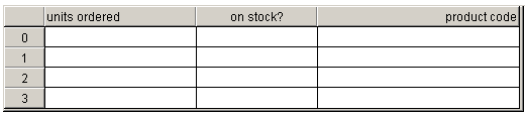
Declaration of variable	Structure of associated table element
arrDim2 : ARRAY[0..2, 0..3] OF INT;	
TYPE MyStruct : STRUCT iNo: INT; bBool : BOOL; sText:STRING; END_STRUCT END_TYPE varStruct: MyStruct;	
arrStruct : ARRAY[0..3] OF MyStruct;	
varFB: TON;	
arrFB : ARRAY[0..3] OF TON;	

Beside some basic properties like for example size, position or line width of the table you may adjust a multitude of table specific properties a selection of which shall be studied enlarging the example arrStruct. The effect of the different properties are shown step by step in the table below.

First of all we change the standard entries in the column headers into more significant titles, adjust the width of the columns and determine the alignment of the titles in the header; for example for the first column

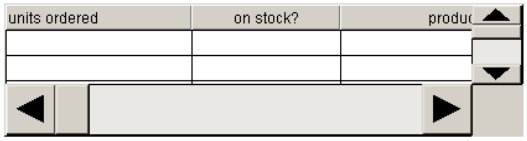
Columns		
Column		
[0]		☑
	Column header	units ordered
	Column width	150
	Text in headline	LEFT

There are two ways to change the column width: 1. In the element properties enter the desired value for"Column width", or 2. Directly In the header line of the table element perform a mouse-click on the dvider between two columns and draw it to the desired position.



We set the height of the rows to 20 pixels and switch off the row header, so that the first column displaying the indices of the array is missing now. The picture at right also shows the enlarged width ("Scrollbar size") of the scrollbar that appears as soon as the size of the table element is shrunk down under the sum of row heights resp. column widths.

Columns	
Column	
Row header	☐
Column header	☑
Row height	20
Row header width	15
Scrollbar size	40



We adjust the size of the table so that the scrollbars disappear again: the height equals the row height multiplied by the number of rows (+ 1 in case of column header switched on), the width equals the sum of the particular column widths and the row header width (in case of row header switched on). In general a small bonus (of for example 1) has to be added to these numbers.

Position	
X	21
Y	221
Width	486
Height	101
Text properties	
HorizontalAlignment	HCENTER
VerticalAlignment	VCENTER
Font	Tahoma; 12px

We change the style of lettering by selecting size, color and typeface in the dialog box opening on a click on ... in the field entitled "Font". Note that the setting for the horizontal alignment within the text properties is valid only for the row header; the text in the table fields are aligned as prescribed by the column settings.

The online representation of the table has now been given a new look:

	units ordered	on stock?	product code
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Note that the alignment of the text in the table fields follows the specific settings made for the columns.

The table fields can be edited column by column: For this purpose for a selected column activate the checkbox "Use template", whereon a folder entitled "Template" will expand within the column properties:

Use template	☑
Use text alignment in headline	☐
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; 12px; style=Bold
Text variables	
Text Variable	PLC_PRG.arrStruct[INDEX].iNo

Three types of template elements are available: the rectangle is entered by default, what may be changed (into a rounded rectangle or a circle) by a click in this field. The related configuration possibilities reflect the element properties of the selected template element:as usual you can specify fill and frame color of the element in normal and alarm state that may also be toggled by an assigned variable. The alignment of the text within the column cells (except for the headers) will now follow the settings of the template.

Note: Instead of the array element entered per default in the field "Text variables", you may enter any other variable contained in the project whose value you want to be displayed. So, in particular, it is possible to change the ordering of the displayed array elements within the table.

Changing the columns can also be done in the table element itself. In the title line perform a mouse-click on the center of a column and while keeping the mouse-button pressed draw the column to the desired position.

In the example below a template has been used for the first and third column. The template of the first column corresponds to the settings demonstrated at left:

	units ordered	on stock?	product code
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

The alarm state may not only be caused by a variable value, but also by selecting a table cell by a mouse click in online mode. The behavior of the selected region is controlled by the section "Selection":

Selection	
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 selection color chosen herein is valid for all selected cells that have no alarm state color prescribed by a template. So, in our example, a selected cell of the second column will be displayed in HotPink, whereas a selected cell of the first column will be displayed in the alarm color of the template (LightSalmon).
If you press the button **Apply**, each particular setting for the alarm state color in the column templates will be replaced within the element properties by the selection color. In consequence the associated entries for the template of the first column in our example now looks like this:

Template	
Type of element	VISU_ST_RECTANGLE
Colors	
Normalstate	
Framecolor	Coral
Fillcolor	Bisque
Alarmstate	
Framecolor	OrangeRed
Fillcolor	HotPink

The field "Selection type" prescribes what elements shall be selected by clicking on one table cell: as we have chosen the selection type SEL_ROW all cells belonging to the same row as the one the click has been performed on will be classified as selected and included by a frame.

If no columns are excluded from being displayed, the position of the selected cell within the table represents the indices belonging to the entry of the visualized array. These indices may then be written to the variables entered in the last three lines of the section "Selection". The third and fourth variable are of type boolean and will be set to TRUE if the selection is valid concerning line resp. column.

Again the table element visualized in online mode: after selecting the cell with the entry "FALSE" by a mouse click all cells in the second row are classified as selected and therefore displayed in alarm state. Only the cell in the second column shows the specified selection color, as the colors indicating an alarm state are prescribed by the templates for the other cells:

	units ordered	on stock?	product code
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Having applied the selection color to the templates also the display will change into the following:

	units ordered	on stock?	product code
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

The associated variables are set to the array indices corresponding to the selected cell:

CoDeSys_SP_Win_V3.Application.PLC_PRG

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)

8.21 Keyboard Usage in Online Mode

For operating a visualization in online mode via keyboard there are some standard keys supported by each device. Additionally there might be device-specifically supported keys.

Standard keys:

Key(s)	Action
Tab	Select next element according to the chronological order of insertion; within a table element each particular cell will be selected; when a frame is selected the selection will be forwarded to the particular subelements
Shift + Tab	Select previous element; inverse direction as via Tab
Enter	Execute input action on the selected element
Arrow keys	Select nearest element in the direction given by the arrow key.

Note: For the integrated visualization the keyboard usage can be explicitly activated and deactivated by command Activate keyboard usage. This might be useful, because as long as the visualization keyboard operation is activated, any other commands given via key shortcuts will not be executed.

Device-specific keyboard usage:

Besides the standard shortcuts described above it depends on the device which further keys might be used for the visualizations running on that device. For details see the Default Hotkeys configuration in the Visualization Manager, valid for all visualizations below an application, resp. the Hotkeys Configuration in the Visualization Editor for a particular visualization.

Note: There is the possibility to handle key events within the application code

8.22 Noticing events and input actions

Methods provided by the visualization libraries might be used in the application to get notified on certain events caused by an user input:

Notice closing of edit controls

Notice key actions

Notice Closing of Edit Controls

The application can notice when an edit control is closed in an assigned visualization, that means when a **variable gets written by an user input in an edit control element**. This might be useful, if an additional action should be executed after an edit control has been closed.

To get notified about such events, a function block instance (implementing the interface VisuElems.IEditBoxInputHandler, provided by VisuElemBase.library) can be attached to the global instance VisuElems.Visu_Globals.g_VisuEventManager, using method **SetEditBoxEventHandler** of this instance like shown in the example below.

VisuElems.IEditBoxInputHandler requires method **VariableWritten**, which will be called after a variable has been written by an 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 a variable") where the user can enter a value for variable i or j.

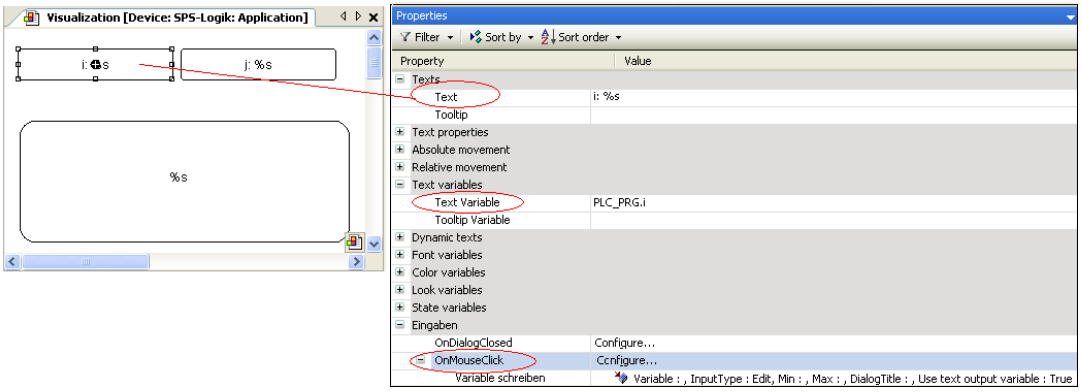


Abb. 65 Input elements

2. Another rectangle for displaying a string (Text Variable stInfo) with some information when one of the edit fields in the upper two rectangles gets closed, that means when a value gets written.

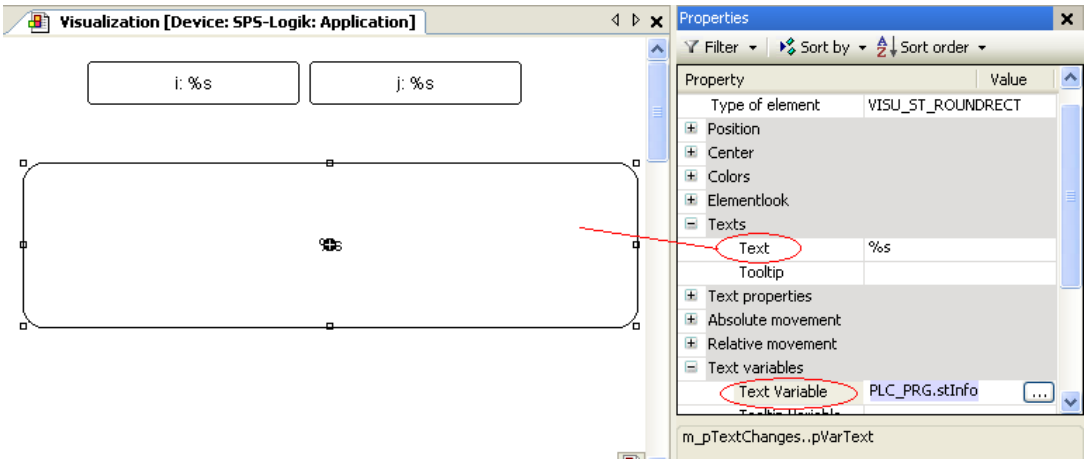
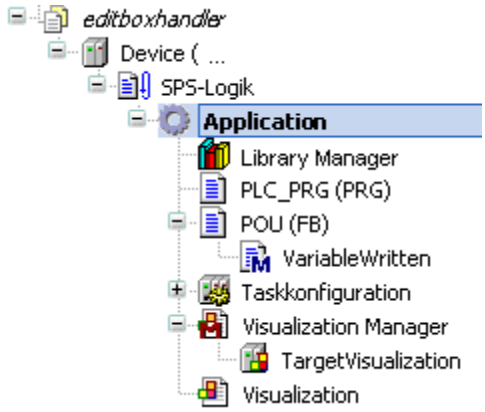


Abb. 66 String display element

The application contains the following POUs:



Program PLC_PRG:

```
PROGRAM PLC_PRG
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_VisuEventManager.SetEditBoxEventHandler(inst); (* Call of method VariableWritten *)
END_IF
```

Function block POU:

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IEditBoxInputHandler
(no further declarations, no implementation code)
```

```
Method VariableWritten, assigned to POU:
METHOD VariableWritten : BOOL   (* provides some information always when an edit control field is closed in the visualization, i.e. 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'));
```

Notice key events

The application can notice when a key event is released in an assigned visualization, that means when the **user operates and releases a key** when the visualization is active.

Note: Regard the options of configuring keyboard operation of visualizations.

To get notified about such events, a function block (implementing the interface VisuElems.IVisuUserEventManager, provided by VisuElemBase.library) can be attached to the global instance VisuElems.Visu_Globals.g_VisuEventManager, using the method **SetKeyEventHandler** of this instance like in the following example:

Example:

There is a visualization with a visualization element that displays the value of PLC_PRG.stInfo as a text string.

That means that in online mode in this element the following information will be shown on the currently released key action: "KeyEvent up: <dwKey>, key: <key id>, modifier: <dwModifiers>, by: <pClient^.globaldata.clienttype.....>".

Example output: "KeyEvent up: TRUE, key: 75, modifier: 0, by: targetvisu".

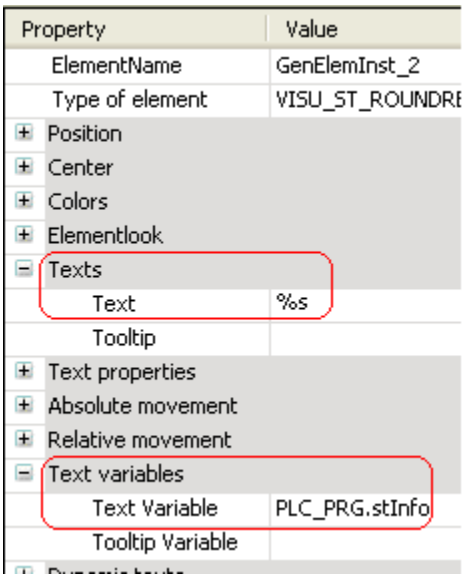
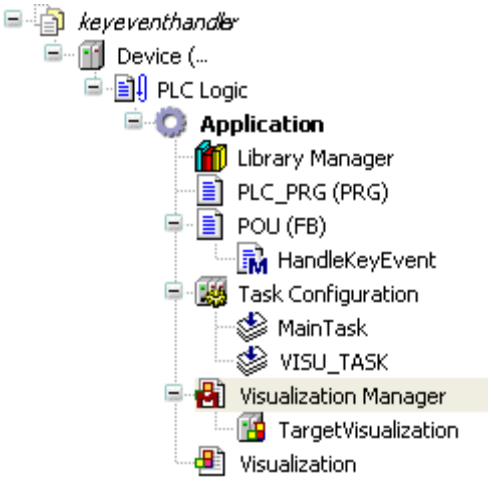


Abb. 67 Text configuration of the visualization element

The application contains the following POUs:



Program PLC_PRG:

```
PROGRAM PLC_PRG
VAR_INPUT
stInfo : STRING;
END_VAR
VAR
inst : POU;
bFirst : BOOL := TRUE;
END_VAR

IF bFirst THEN
bFirst := FALSE;
VisuElems.Visu_Globals.g_VisuEventManager.SetKeyEventHandler(inst);
END_IF
```

This allows to get notification when a key event is released.

Function block POU:

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IKeyEventHandler
(no further declarations, no implementation code)
```

Method HandleKeyEvent, assigned to POU:

The interface VisuElems.IVisuUserEventManager requires a method with the following signature that will be called with all available information:

```
/// 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;
/// Modifiers. Possible values:
/// VISU_KEYMOD_SHIFT : DWORD := 1;
/// VISU_KEYMOD_ALT : DWORD := 2;
/// VISU_KEYMOD_CTRL : DWORD := 4;
dwModifiers : DWORD;
/// Pointer to the client structure were 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, 'other visu', 'targetvisu'));
```

8.23 Input Dialogs

A visualization can be designed as and declared in its object properties as a "dialog". Dialog visualizations then can be used in other visualizations in order to provide an user input mask. They will be offered when you are configuring for example an **OpenDialog** or **OnDialogClosed** property of a visualization element.

Besides of possibly self-designed dialogs the standard login dialog visualization is available when the **VisuDialogs.library** is included in the project.

The parameters defined in the **interface** (Interface Editor) of a dialog visualization automatically get written to a structure named **<dialogname>_VISU_STRUCT**. ("dialogname" = name of the visualization). This structure then might be accessed by the application, for example via a function, which is configured to be called in case of a user input on a visualization element. This allows to program the opening of a dialog in a visualization and also the reaction on any user inputs on this dialog.

DialogManager

All visualizations declared as "dialogs" will be instantiated automatically and managed by the internal **DialogManager**. This manager can be accessed via the also internal **VisuManager** by calling method **GetDialogManager()**. The DialogManager provides some methods for handling a visualization dialog by the application.

GetDialog(STRING stName) : Provides the dialog, which is currently concerned by an event, as IVisualisationDialog:

METHOD **GetDialog**

Parameters:

Name	Type	Comment
GetDialog	VAR_OUTPUT	
stName	VAR_INPUT	

GetClientInterface() : Returns a pointer on the dialog structure

METHOD **GetClientInterface**

Parameters:

Name	Type	Comment
GetClientInterface	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

OpenDialog() : Opens the dialog for the given client

METHOD **OpenDialog**

Parameters:

Name	Type	Comment
OpenDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	
bModal	VAR_INPUT	
pRect	VAR_INPUT	

CloseDialog() : Closes the dialog for the given client

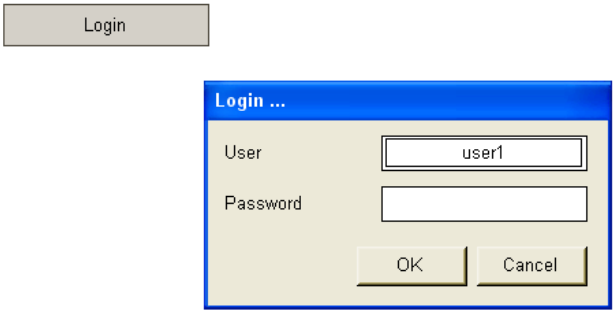
METHOD **CloseDialog**

Parameters:

Name	Type	Comment
CloseDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

Example of using the Login-Dialog provided by VisuDialog.library:

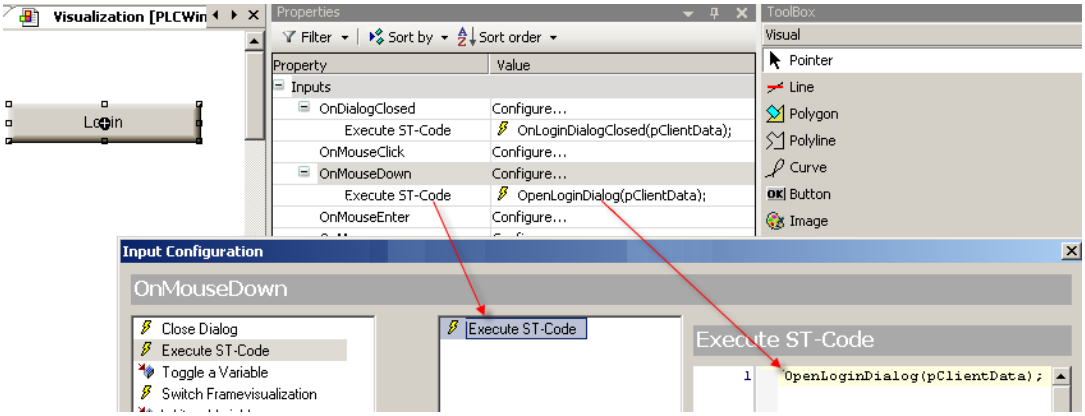
In this example a button element in a visualization is configured in a way that on a mouse-click on this button a login dialog gets opened, where a user name and password can be entered. A default login dialog is used for this purpose, provided by the VisuDialog.library. However, self-created dialogs could be handled in the same way:



Configuration of the "Login" button:

A button element is inserted in a visualization. It gets configured with a text property "Login" and the following two input properties:

1. OnMouseDown / Execute ST-Code: call function OpenLoginDialog(pClientData). Variable pClientData must be passed to an appropriate function OpenLoginDialog available in the project.



2. OnDialogClosed / Close Dialog / Execute ST-Code: call function OnLoginDialogClosed(pClientData)

The respective functions OpenLoginDialog and OnLoginDialogClosed are available as POUs in the project. See in the following a possible implementation:

Implementation of functions OpenLoginDialog and OnLoginDialogClosed:

1. Function OpenLoginDialog, getting called when the dialog gets opened, that is on a mousedown-action on the button element.

This function reads the parameters of the opened dialog.

```
FUNCTION OpenLoginDialog : bool
VAR_INPUT
pClientData : POINTER TO VisuStructClientData;
END_VAR
VAR
dialogMan : IDialogManager;
loginDialog : IVisualisationDialog;
pLoginInfo : POINTER TO Login_VISU_STRUCT; // Login_VISU_STRUCT contains the parameters defined in the interface of visualization "Login"
result : Visu_DialogResult;
stTitle : STRING := 'Login ...';
stPasswordLabelText: STRING;
stUserLabelText: STRING;
stUsername: STRING;
END_VAR

dialogMan := g_VisuManager.GetDialogManager();    // The DialogManager is provided via the implicitly available VisuManager
IF dialogMan <> 0 AND pClientData <> 0 THEN
loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); // dialog to be opened is specified
IF loginDialog <> 0 THEN
pLoginInfo := dialogMan.GetClientInterface(loginDialog, pClientData);
IF pLoginInfo <> 0 THEN     // in the following the parameters of the login dialog in the Login_VISU_STRUCT will be read
pLoginInfo^.stTitle := stTitle;
pLoginInfo^.stPasswordLabelTxt := stPasswordLabelText;
pLoginInfo^.stUserLabelTxt := stUserLabelText;
dialogMan.OpenDialog(loginDialog, pClientData, TRUE, 0);
END_IF
END_IF
END_IF
```

2. Function OnLoginDialogClosed, defining the reaction on a closing of the 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 VisuManager
IF dialogMan <> 0 AND pVisuClient <> 0 THEN
loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); // gets the Login dialog
IF loginDialog <> 0 THEN
result := loginDialog.GetResult(); // gets the result (OK, Cancel) of the dialog
IF result = Visu_DialogResult.OK THEN
loginDialog.SetResult(Visu_DialogResult.None); // reset to default (none)
pLoginInfo := dialogMan.GetClientInterface(loginDialog, pVisuClient); // structure
Login_VISU_STRUCT gets read; in the following the structure parameters can be set
IF pLoginInfo <> 0 THEN
stPassword := pLoginInfo^.stPassword;
pLoginInfo^.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
```

9 Symbol configuration

The Symbol configuration is used to create **symbols**, provided with specific **access rights**, via which project variables can be accessed from external, for example by an OPC Server. The description of the symbols (symbol information) will be available in a xml-file (symbol file) in the project directory, and it will be downloaded to the PLC together with the application.

Creating a symbol configuration:

In order to get the symbol configuration functionality available, the **Symbol configuration** object must be added to the **application** in the device tree via the Add Object command. This will automatically include the **IECVarAccess.library** in the Library Manager.

The variables to be exported as symbols can be defined either in the Symbol configuration editor or via pragmas, which are to be added at the declaration of the variables. Another possibility is provided for example by the SFC Editor: Here in the element properties the implicitly created element flags can be defined to get exported to the Symbol configuration.

The **name of a symbol** created by the symbol configuration is composed according to the following syntax:

<application name>.<POU name>.<variable name>

for example "MyApplication.PLC_PRG.a" or "MyApplication.GVL.a"

For the purpose of **accessing** the variable, the symbol name must be specified completely.

Symbol information:

The symbols defined for an application get exported to a xml-file in the project directory (**symbol file**). This file is named according to the following syntax:

<project name>.<device name>.<application name>.xml.

for example proj_xy.PLC1.application.xml

Note: The symbol file will be created, when the "Generate code" command is executed in the "Build" menu.

Further on the symbol information gets **downloaded** to the PLC with the application. Depending on the device description it will be included in the application or a separate child application will be generated, which then also will be listed in the Applications dialog of the PLC device editor. The syntax of a separate symbol application:

<application name>._Symbols

for example if application "application", for which a symbol configuration has been defined, gets downloaded, the symbol information will be downloaded as "**<application name>.__Symbols**":

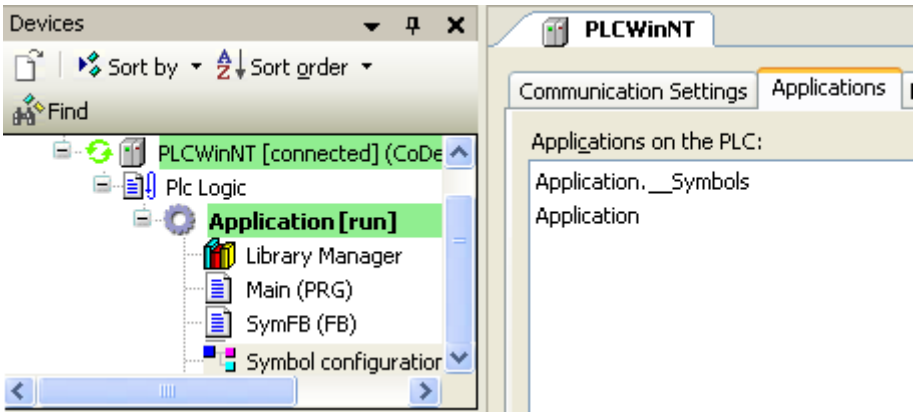


Abb. 68 Example, Application and "symbol application"

If the symbol configuration has been modified in online mode, it can be explicitly re-loaded to the controller using button Download in the editor window..

For example concerning the maximum number of applications on a PLC the symbol application must be noticeed like a "normal" application.

10 Source Control

10.1 Source Control via ENI

General

The objects of a PLC Designer project can be managed via the ENI (Engineering Interface) in an **external data base**. The usage of such an data base guarantees the consistency of the data, which then can be shared by several users, projects and programs. Also it extends the PLC Designer functionality by making possible the following items:

- **Revision control** for project objects: If an object has been checked out from the data base, modified and checked in again, then in the data base a new version of the object will be created, but the older versions will be kept also and can be called again on demand. For each object and for a whole project the version history will be logged. Two versions can be checked for differences.
- **Multi-User Operation**: The latest version of a sample of objects, for example of POUs of a project, can be made accessible for a group of users. Then the objects which are currently checked out by one of the users will be marked as "in the works" and not editable by the other users. Thus several users can work in parallel on the same project without risking to overwrite versions mutually.
- **Access by external tools**: Besides the PLC Designer programming system also other tools, which also have an ENI, can access the common data base. Those might be for example external visualizations, ECAD systems etc., which need the data created in PLC Designer or which also produce data which are needed by other programs.

ENI (Engineering Interface)

The ENI is an interface component which allows to connect the PLC Designer programming system to an external data base.

The ENI is composed of a **client** and a **server** part. So it is possible to hold the data base on a remote computer, which is required for multi-user operation. The PLC Designer programming system is a client of the independent ENI Server Process as well as another application, which needs access to the data base, might be such a client.

The ENI supports data base systems like for example 'Visual SourceSafe 6.0', as well as a local file system. For the currently supported data base systems as well as for detailed information on the ENI Server please see the **separate ENI Server documentation**. Objects can be stored in different locations of the data base. They can be checked out for editing and thereby will get locked for other users. The latest version of an object can be called from the data base. Further on in parallel you can store any objects just locally in the project as usual for projects without source control.

Working with an ENI project data base

Note the differences to PLC Designer V2.3.

Note: For a guide concerning installation and usage of the *ENI Server* provided by Lenze please see the separate server documentation resp. online help. There you will also find a quickstart guide.
Also consider the possibility of using the *ENI Explorer* which allows to perform data base actions independently from the currently used data base system.

Preconditions:

- the communication between PLC Designer and an ENI Server requires **TCP/IP**, because the ENI Server uses the HTTP protocol.
- an **ENI Server** (ENI Server Suite) must be installed and started locally or on a remote computer. A license is required to run it with one of the standard database drivers which has been installed with the server. Just the driver for a local file system can be used with a non-licensed ENI Server version.
- in the ENI Server service control tool (ENI Control) the **connection to the desired data base** (see the following list item) must be configured correctly (section 'Data base'). You will automatically be asked to do this during installation, but you can modify the settings later in ENI Control.
- a **project data base** for which an ENI-supported driver is available, must be installed. It is reasonable to do this on that computer, where also the ENI Server is running. Alternatively a local file system can be used, for which an appropriate driver is provided by default.
- in the data base administration possibly the user (Client) as well as the ENI Server must be registered as **valid users** with access rights. Anyway this is required for the 'Visual SourceSafe' , if you use another data base system, please see the corresponding documentation for information on the user configuration.
- for the current PLC Designer **project** the use of the ENI must be **activated** and the **connection** to the ENI Server must be configured correctly; this by default is to be done in the PLC Designer dialog 'Project' 'Source Control' 'Change Source Control...'.
- the **user must log in to the ENI Server** with user name and password; this is to be done in the Login-dialog, which can be opened explicitly by the command 'Project' 'Source Control' 'Login...' resp. which will be opened automatically in case you try to access the data base without having logged in to the ENI Server before. Notice that in contrast to PLC Designer V2.3 the user only must log in once to the ENI Server in order to work on several projects in the data base and has not explicitly to log in with each project separately.

Handling PLC Designer project objects under source control:

- The **'Source Control' commands** (Get Latest Version,Check Out, Check In, Version History, Label Version etc.) which are used for managing the project objects in the ENI project data base, will be available in the current PLC Designer project as soon as the connection to the data base has been activated and configured correctly.
- The current assignment ("binding") of an object to a **data base location** is shown in the Object Properties,category 'Source Control', and can be modified there. An overview on all objects locations within the data base and also the possibility of editing is given in the 'Advanced Settings' in the 'Change Source Control...' dialog.
- The **general settings** for source control actions, for example whether always the latest version of the objects should be called from the data base before a project gets built, can be modified in the Source Control Options dialog.

10.2 Differences to PLC Designer V2.3

The Source Control via ENI is simplified in use compared to PLC Designer V2.3:

- **No object categories, but "bindings" of objects to database locations**:
  The "Shared Object" and "Compile files" object categories do not exist any longer. All objects quasi are "Project objects", but can be assigned to any database locations different from the default project objects location. Objects, which should be accessible also by other ENI database clients (recently declared as "Shared objects"), do not have to be put any longer in one definite "common" database folder, but one or several locations besides the default project location might be defined for such purposes.
- **Connection to the ENI Server just to be defined once**:
  The configuration of the database connection (ENI server) to the default project database location is now to be done in just one dialog (in PLC Designer V2.3 it was to be done in three different dialogs for Project objects, Shared objects and Compile files). Just if bindings of particular project objects to other database locations are desired, these must be defined explicitly for the respective objects. This however is possible in one and the same dialog for all objects.
- **Tree structure of project objects in database always corresponds to that in the project**:
  The project objects will always be stored in the default database location in a tree structure which corresponds to that of the POUS and Devices view in PLC Designer.
- **Authentication for database access only needed once, for the respective ENI Server**:
  The authentication for the database access is not any longer bound to a project or even to object categories, but to an ENI Server. If you have logged in to the respective ENI Server once and have activated the option for saving the credentials, this authentication will keep effective until you use the Login dialog again, even if PLC Designer has been re-started in the meantime. If the credentials have not been saved, you have to log-in again after a restart of PLC Designer.

11 Project Documentation

A documentation, that is a print of the current project can be created via the functions provided with the 'PrintEngine' plug-in.

Print sets of the complete project, including a title and a contents page as well as just prints of single objects are possible.

Commands of category 'Print' are available for getting access to the dialogs for defining the contents of a print set, for configuring the page layout and the printer setup.

12 Menu Commands sorted by Categories

12.1 Commands and Menus

Which commands are available in the »PLC Designer« user interface depends on the currently used profile, that is the currently installed plug-in components.

Where - in which menus - the commands are available, basically also depends on the current profile which provides a default menu configuration, but might be modified by the user in the respective Customization dialog.

Find the command descriptions in this online help sorted by categories as used in the customization dialog.

12.2 Bookmarks

The command category 'Bookmarks' provides commands for handling bookmarks in text editors.

12.2.1 Bookmarks commands

The possibility to set bookmarks in text editors is provided with the Bookmarks-Plugin. Bookmarks can be assigned to one or multiple lines in an editor to make navigating in long programs easier. Via the appropriate commands you can jump to the next or previous bookmark.

The respective commands for creating bookmarks and navigating via bookmarks by default are available in a submenu of the 'Edit'-menu. If necessary you can modify the menu structure via the Customize dialog.

12.2.2 Toggle Bookmark

Symbol:

This command (category Bookmarks) is used in a text editor to set a bookmark in the current line respectively to remove a set bookmark. A cyan-colored rectangle at the left margin will indicate that a bookmark is set.

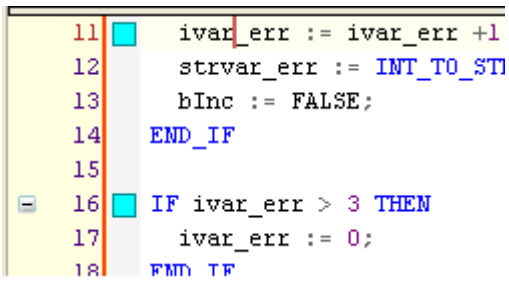


Abb. 69 Bookmarks in ST editor

12.2.3 Next Bookmark

Symbol:

This command (category Bookmarks) is used in a text editor to jump to the next bookmark.

12.2.4 Previous Bookmark

Symbol:

This command (category Bookmarks) is used in a text editor to jump to the previous bookmark.

12.2.5 Clear Bookmarks

Symbol:

This command (category Bookmarks) is used to clear all bookmarks in the current editor window.

12.3 Breakpoints

The command category 'Breakpoints' provides commands for handling breakpoints for debugging purposes.

12.3.1 Breakpoints commands

The breakpoint functionality for debugging purposes is provided with the 'Breakpoints' plug-in. Breakpoints can be defined to work as halt positions when processing an application program for example for debugging purposes. Further on there is the possibility to execute the program in defined steps (**stepping**).

The respective commands are of category 'Breakpoints'. By default they are available in the 'Online' menu. If you want to re-organize the menu structures, use the Customize dialog.

The following commands are available for handling breakpoints and for stepping through a program:

- Toggle Breakpoint
- New Breakpoint
- Breakpoint positions
- Step Into
- Step Over
- Step Out
- Set Next Statement
- Run to Cursor
- Show Current Statement

- Notice also the Breakpoints command which is available in the 'View' menu for opening the breakpoints dialog.

12.3.2 Toggle Breakpoint

Default Shortcut: <F9>

This command (category Breakpoints) basically toggles between status 'enabled' ( ) and 'disabled' ( ) of a breakpoint. However it also effects that a new breakpoint will be set if not yet one is already set at the current breakpoint position. (For defining a new breakpoint regard also the possibility of using the New Breakpoint dialog.)

Notice that an enabled breakpoint after logout and re-login automatically will be disabled.

Note: Each active breakpoint will get an "inactive" one, when you leave online mode and login again afterwards.

An active breakpoint is indicated by the symbol, an inactive one by the symbol.

Note: See also the corresponding functions in the 'Breakpoints' dialog.

12.3.3 New Breakpoint...

Symbol:

This command (category Breakpoints) is used to insert a new breakpoint in one of the project's POUs. It **does not care where the cursor currently is placed,** the 'New Breakpoint' dialog will open where in sub-dialog **Location** you can choose one of the possible breakpoint positions allover the project and in sub-dialog **Condition** can define some conditions for the new breakpoint.

Note: For setting a breakpoint at the current cursor position notice command Toggle Breakpoint. See also the corresponding functions in the 'Breakpoints' dialog.

Location

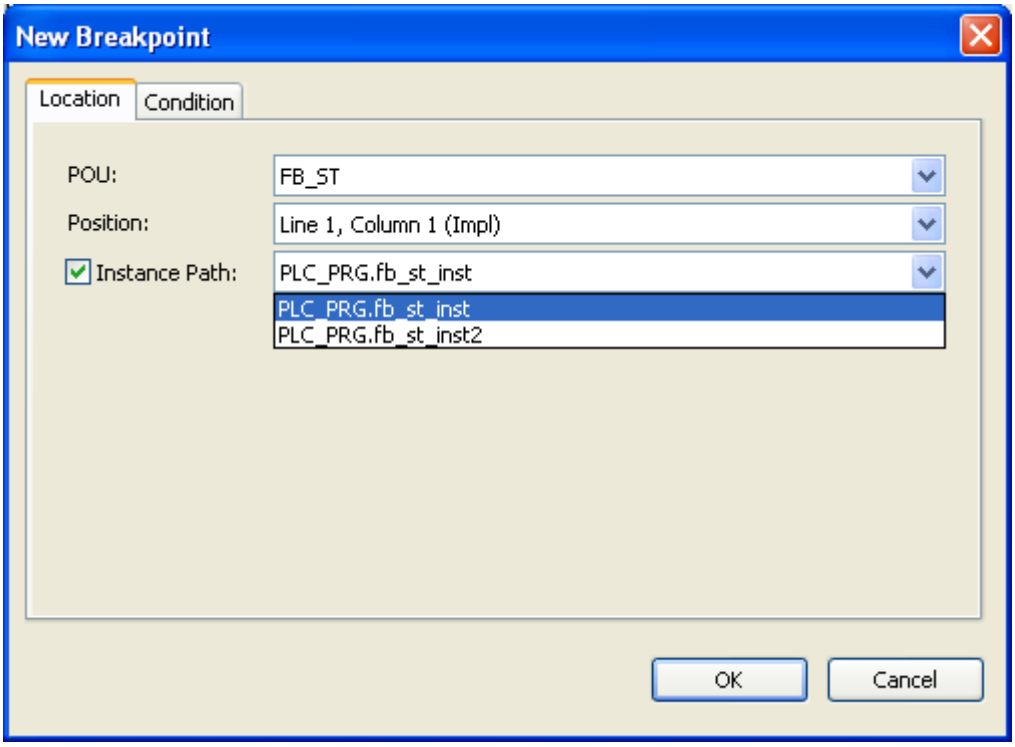


Abb. 70 New Breakpoint' dialog, Location

POU: The selection list offers all POUs currently available in the project. Select that one where you want to set a breakpoint

Position: The selection list offers all possible breakpoint positions of the currently selected POU. Depending on the editor type these positions are defined by Line+Column numbers (text editors) or as Network or Element numbers (graphic editors). In case of a function block additionally "(Impl)" will be added and you have to decide whether the breakpoint should be set in the **implementation** or in an **instance**. If it should be set in the implementation, leave option 'Instance Path' deactivated. If it should be set in an instance, activate option 'Instance Path' to select the desired instance; see the following:

Instance Path: If the currently selected POU is a function block and this option is deactivated, the breakpoint will be set in the implementation body of the POU (see above, Position). If you want to set the breakpoint in an instance, activate the option and select the desired instance.

Condition

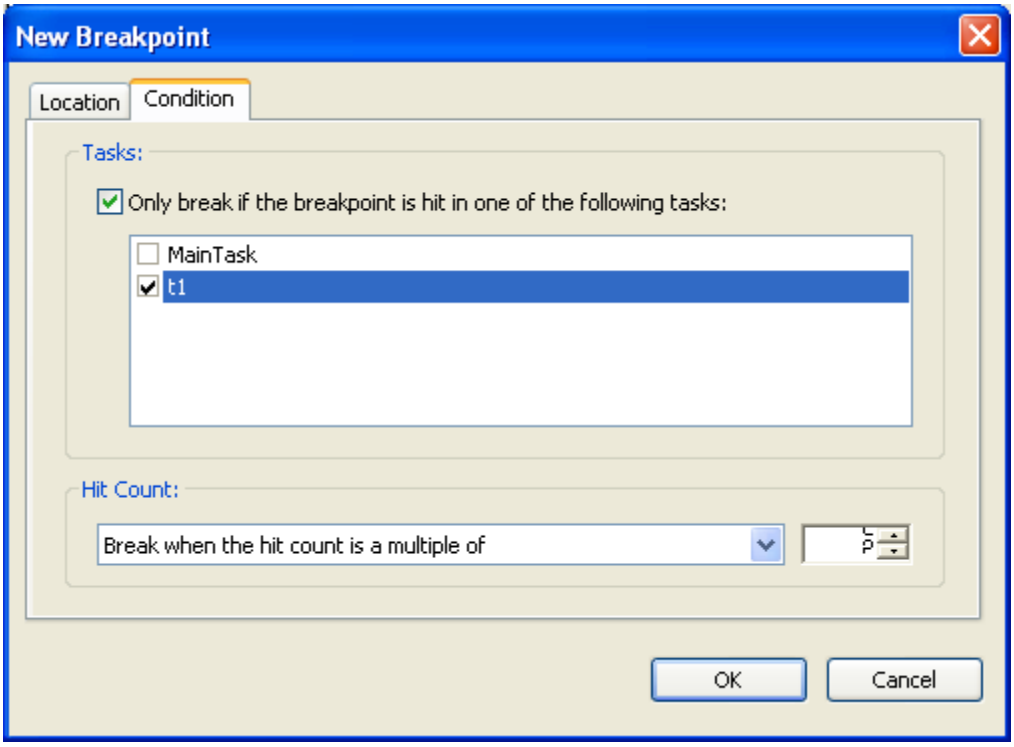


Abb. 71 New Breakpoint' dialog, Condition

Tasks:

Only break if breakpoint is hit in one of the following tasks is activated: Activate this option if the breakpoint only should be effective, if the POU where it is placed, is processed by particular tasks. All tasks currently defined in the project will be listed and the desired one(s) can be defined by setting tick(s) correspondingly.

Hit Count:

Break always: The program always will stop at the breakpoint.

The program will not stop at the breakpoint until the breakpoint has been hit the number of times defined (Enter the desired number resp. select it in the number field):

Break when the hit count is equal to

Break when the hit count is a multiple of

Break when the hit count is greater than or equal to

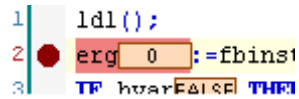
12.3.4 Breakpoint positions

The possible breakpoint positions depend on the editor. Basically they are those positions in a POU at which values of variables can change or at which the program flow branches out resp. another POU is called.

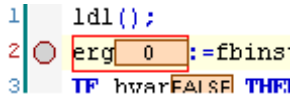
Note: Notice for breakpoints in methods: A breakpoint will be set automatically in all methods which might be called. So, if an interface-managed method is called, breakpoints will be set in all methods of function blocks implementing that interface and also in all derivative function blocks subscribing the method. If a method is called via a pointer on a function block, breakpoints will be set in the method of the function block and in all derivative function blocks which are subscribing the method.

Breakpoint symbols:

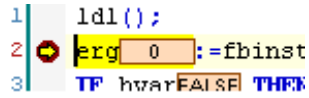
Breakpoint in online mode:



Disabled breakpoint:



Program stop at breakpoint:



12.3.5 Step Into

Symbol:

Default Shortcut: <F11>

This command (category Breakpoints) can be used for stepping through a program in online mode, for example for debugging purposes.

A single step will be executed. The program will stop before the next instruction. If necessary there will be a changeover to an open POU. If the present position is a call-up of a function or of a function block, then the command will proceed on to the first instruction in the called POU.

In all other situations, the command will act like 'Step Over'.

The possible halt positions during stepping depend on the editor. The current position is indicated by a yellow shading.

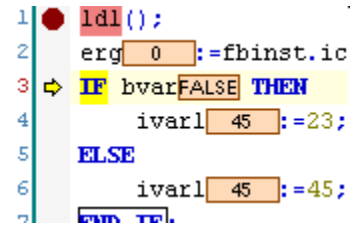


Abb. 72 Step Into, example

12.3.6 Step Over

Symbol:

Default Shortcut: <F10>

This command (category Breakpoints) can be used for stepping through a program in online mode, for example for debugging purposes.

A single step will be executed. With one-level instructions this is equal to stepping with 'Step In' (i.e. from instruction to instruction). If however a POU call is reached, then Step Over effects that this POU will be executed completely within the current step. In SFC a complete action will be executed.

If you would like to jump to the first instruction of a called POU, you had to use the 'Step Into' command.

Notice also: Step Out

12.3.7 Step Out

Symbol:

Default Shortcut: <Ctrl>+<F11>

This command (category Breakpoints) can be used when stepping through a program in online mode, for example for debugging purposes.

If the application program does not contain any calls, Step Out will effect a jump back to the start of the application. If however you previously have stepped into a called POU, "Step Out" will effect a jump back just to the calling instruction (new!). So, in case of nested calls, "Step Out" will lead you back through the hierarchy of callers step by step. This for examples allows to go back and to step in to another called POU.

Notice also: Step Into, Step Over

12.3.8 Set Next Statement

Symbol:

This command (category Breakpoints) can be used when stepping through a program in online mode (e.g. for debugging purposes).

It defines the next instruction (statement) to be executed. For this purpose place the cursor in the desired statement and perform the command.

12.3.9 Run to Cursor

Symbol:

This command (category Breakpoints) can be used (e.g. for debugging purposes) in order to execute the program up to a temporarily definable position.

This corresponds to setting a breakpoint at the desired next stop position and stepping there.

Place the cursor at the desired "next break" position and perform the command. The instructions between the current break position and the cursor position will be executed.

12.3.10 Show Current Statement

Symbol:

This command (category Breakpoints) can be used in online mode to get back to the current execution position.

This might be useful, if for any reason have left the window where you are currently debugging, and have put the cursor somewhere else in the programming system. The window showing the respective POU will be put again in foreground and the cursor will be placed again at the current position of execution.

12.4 Browse Project

The command category 'Browse Project' provides commands for evaluating the call stack of an identifier or POU and for getting opened the place where it is defined.

12.4.1 Go To Definition

Browse Project commands - Go To Definition

The 'BrowseProject' plug-in provides commands (category 'Browse Project') for browsing the project in order to get information about the dependencies resp. the "call tree" etc. of a POU or a variable currently selected in an editor window.

By default these commands are available in the 'Edit' menu, submenu 'Browse'. To re-organize the menu structure please use the Customize dialog.

Go To Definition

Symbol:

This command (category Browse Project) can be used when the cursor is currently positioned on an identifier in an editor window. It will browse the project for the line or POU, which contains the "definition" of the corresponding POU or variable and will open the respective POU in an editor window.

The command corresponds to the "Zoom" (to POU) command available in »PLC Designer« V2.3 and can be used in offline and in online mode. If applicable, the 'Select Online State' dialog will be opened, where you can decide whether to open the online or offline view of the POU and - in case of function block instances - whether you want to view the instance or implementation.

Examples:

The following POU contains 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;
```

If you put the cursor on "prog_y", the command will open program "prog_y" in its editor window.

If you put the cursor on "fbinst", the command will set the focus up to the declaration window to the line "fbinst:fb1;"

If you put the cursor on "out", the command will open function block "fb1" in its editor window.

12.4.2 Cross References List

Browse Project commands - Cross Reference View

The 'Cross Reference View' plug-in provides commands concerning the cross references list of a variable.

By default these commands are available in the Cross Reference View window.

The commands:

Browse Cross References

Go To Definition

Browse Cross References

Symbol:

This command (category Browse Project) dumps the cross reference list, which is currently displayed in the Cross References List view, to the message view. This might be of interest when the automatic cross reference listing is activated.

The command is also available via a button in the Cross Reference List View.

Go To Definition

Symbol: Shortcut: <F2>

This command (category Browse Project) effects a jump to the location, where the variable, the reference of which is currently selected in the cross reference list, is declared. The corresponding declaration editor will be opened and the variable will be highlighted there.

12.5 Build

The command category 'Build' provides commands for handling the project file.

12.5.1 Build Commands

The commands listed below are provided with the 'Build Commands' plug-in for command category 'Build'. If not explicitly named differently, they always refer to the currently active application.

The "build" commands on the one hand serve to handle **syntactical checks**, either just on the changed objects or on all objects of the active application. On the other hand you can perform an **offline code generation** run in order to check for detected compile errors before downloading the code to the device.

The results each get displayed in the message window, category 'Build'.

The "clean" commands serve to delete the **compile information** which was created at the last download - including generation of compilation code - on the target system. This for example will concern Online Change.

By default the commands are part of the 'Build' menu. If necessary open the Customize dialog to view resp. modify the menu configuration. Notice category 'Build'.

Note: The currently used compiler version is defined in the Project Settings, category Compile options.

- Build
- Rebuild
- Generate code
- Clean
- Clean all
- Generate runtime system files
- (Check All Pool Objects)

12.5.2 Build

Symbol:

Default Shortcut: <F11>

This command (category 'Build') starts the build process of the currently active application. This means that all objects belonging to the application will be syntactically checked. Notice that **no compilation code** will be generated like it will be done when you are going to login resp. to download an application! The build process is done automatically before each log-in with changed application program.

The syntactical check being accomplished, any error messages or alerts (warnings) will be displayed within the **message view** in category "Build". If the number of errors/warnings exceeds 500, all further messages will be skipped and a special error message will inform about this situation. Commands of category "Message View" are available for navigating between messages and messages and source code.

If the program has not been changed since the last build-process, and no errors were detected, it will not be built again and the message "The application is up to date" will be displayed in the message window. If you nevertheless want to get the syntactical checks done again, please use command Rebuild.

Note: The currently used compiler version is defined in the Project Settings, category Compile options.

12.5.3 Rebuild

This command (category 'Build') starts the build process for the currently active application, even if it had been built before without detected errors.

Note: The currently used compiler version is defined in the Project Settings, category Compile options.

12.5.4 Generate code

This command, which by default is part of the 'Build' menu, allows to compile the currently active application just for test purposes. A code generation run will be done like by default is done when logging in with the application. However **no code will be downloaded** and no compile info file will be created in the project directory. Thus you can check for any detected compilation errors before going online with an possibly incorrect code.

12.5.5 Generate runtime system files

This command (category 'Build') can be used to prepare the creation of an external library file. From the current library project you can generate a C stub file and an interface file in M4-format, which then can be followed up outside of »PLC Designer«.

The command opens the dialog 'Generate Files for Runtime System':

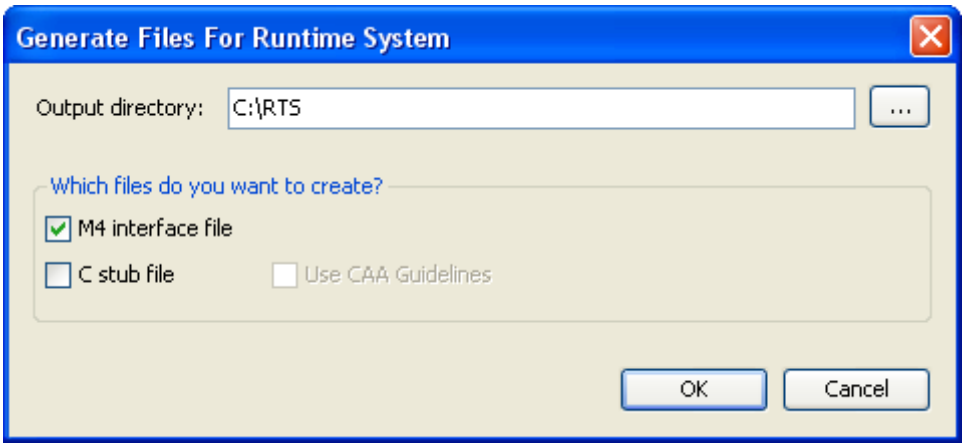


Abb. 73 Dialog 'Generate Files for Runtime System'

Output directory: Define the directory where the files should be created. For this purpose the standard dialog for browsing the file system might be opened via button ...

Which files do you want to create?: Set a tick at the desired file type(s):

M4 interface file: Interface file **<project name>>Itf.m4**, containing definitions.

Example:

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 Rea
 * *    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 (rea
 *
 */
typedef enum
{
        AM_READ = 0,
        AM_WRITE = 1,
        AM_APPEND = 2.
```

C stub file: Frame for further implementation of the library file in C.

Example:

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)
{

}
```

12.5.6 Check All Pool Objects

This command might be used for test purposes when creating a library project. It causes a "build" run of all objects currently available in the POUs window. This means that those will be checked for syntax errors. Notice that **no compilation code** will be generated.

12.5.7 Clean

This command (category 'Build') deletes the **compile information** for the currently active application. The compile information was created and stored in a file *.compileinfo in the project directory during the last download of the application.

After a clean process no Online Change will be possible for the respective application. The program first must be re-downloaded.

12.5.8 Clean all

This command, which by default is part of the 'Build' menu, deletes the compile information for **all** applications. The compile information was created and stored in a file *.compileinfo in the project directory during the last download of the application.

After a clean process no Online Change is possible for the respective application. The program first must be re-downloaded.

12.6 CAM

The command category 'CAM' provides commands for handling the project file.

12.6.1 CAM Commands

The commands of category 'CAM' are provided by the 'CAM Editor' plug-in, which is part of »PLC Designer« SoftMotion V3.

By default they can be found under the menu item "**CAM**". Else you may open the Customize dialog, category 'CAM', to view or modify the current menu configuration.

The commands:

- write CAM online file
- write CAM data into ASCII table
- read CAM online file
- read CAM data from ASCII table

12.6.2 read CAM online file

This command being part of menu item 'CAM' by default is used to read in data describing a CAM from an external file of proprietary type ".cam", which has been created before by use of the command write CAM online file.

12.6.3 read CAM data from ASCII table

This command being part of menu item 'CAM' by default is used to read in data describing a CAM from an ASCII file (".txt"). Reading the CAM data into a project you will be asked on the number of points that shall be taken into account for creating the CAM within the programming environment. By default the number of data points contained in the file is proposed, but restricting the amount of CAM data to a smaller set of equidistantly dispersed points may be reasonable: as the CAM will be interpolated by polynomials of degree 5, a high number of interpolation points can result in oscillations.

12.6.4 write CAM online file

This command being part of menu item 'CAM' by default is used to write the CAM data to an external file of proprietary type ".cam". The CAM data saved includes the description of the CAM as well as position and type of the tappets and can be read into another project by use of the command read CAM online file.

12.6.5 write CAM data into ASCII table

This command being part of menu item 'CAM' by default is used to write the data to an external file of ASCII type ".txt".

Saving the CAM data you will be asked on the number of points, that is the amount of data describing the CAM. These points will be dispersed equidistantly over the X range of the cam. The data saved includes the x- and y-position of these points. Be aware that no information on tappets will not be saved.

The CAM data saved can be read in to a project by use of the command read CAM data from ASCII table.

12.7 CFC

The command category 'CFC' provides commands for working in the CFC Editor.

12.7.1 CFC Editor commands

The commands of category 'CFC' are provided by the 'CFC Editor' plug-in and are available for programming in the CFC editor.

By default they are part of the **"CFC" menu**. If necessary, open the Customize dialog, category 'Menu', to view or modify the current menu configuration.

The commands:

- Edit Working Sheet
- Negate
- EN/ENO
- Set/Reset:
  - None
  - R - Reset
  - S - Set
- Execution Order:
  - Send To Front
  - Send To Back
  - Move Up
  - Move Down
  - Order By Data Flow
  - Order By Topology
  - Set Execution Order
- Connect Selected Pins
- Reset Pins
- Remove unused pins
- Edit Parameters...

12.7.2 Edit Working Sheet

This command (category CFC) opens the **Edit Working Sheet dialog** for modifying the size of the working area ("working sheet") of the current CFC.

The size of the "working sheet" is defined by the height and width of a rectangular area having its origin (X: 0, Y: 0) in the upper left corner of the editor window and including all existing CFC elements. Height and width are specified in number of **grid units** whereby the size of a grid unit is not changeable by the user. Height (Y): increasing positive values from top to down, width (X): increasing positive values from left to right.

The maximum size is 2048 grid units in width and height.

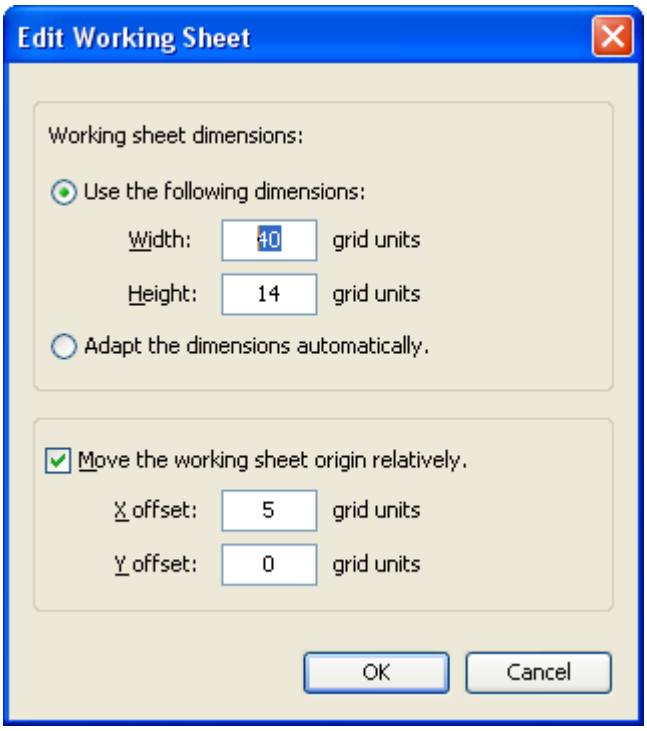


Abb. 74 Edit Working Sheet dialog

Working sheet dimensions:

Use the following dimensions: If this option is activated, the size of the worksheet will be determined by the following width and height values:

Width: Shows the current width in grid units. Can be edited, whereby it will not be possible to enter a width smaller than that which is actually required by the existing elements. Increasing the value (X) will enlarge the width horizontally to the right.

Height: Shows the current height in grid units. Can be edited, whereby it will not be possible to enter a height smaller than that which is actually required by the existing elements. Enlarging the value (Y) will enlarge the working sheet vertically downwards.

Adapt the dimensions automatically.: This option is activated by default. The size of the working sheet is defined by the bottommost (height) and the rightmost (width) element borders within the editor window. The origin (X=0, Y=0) is in the upper left corner.

Move the working sheet origin relatively.: If this option is activated, the working sheet can be shifted vertically and/or horizontally by the offset values given in the following. The restrictions: The shift might not lead to an upper left corner less than 0/0. If option 'Use the following dimensions' is activated in the upper part of the dialog, the shift may not exceed the width and height defined there. If option 'Adapt the dimensions automatically' is activated, the shift might exceed the current dimensions and the width and height values will be updated accordingly.

X offset: By default is 0. Entering a positive value shifts the chart to the right, thus possibly increasing the width of the working sheet. Entering a negative value shifts the chart to the left, thus only possible, if there is space between the leftmost element and the left window border.

Y offset: By default is 0. Entering a positive value shifts the chart downwards, thus possibly increasing the height of the working sheet. Entering a negative value shifts the chart upwards, thus only possible, if there is space between the uppermost element and the upper window border.

If you enter invalid sheet size values, an error message dialog will pop up, also listing the given restrictions.

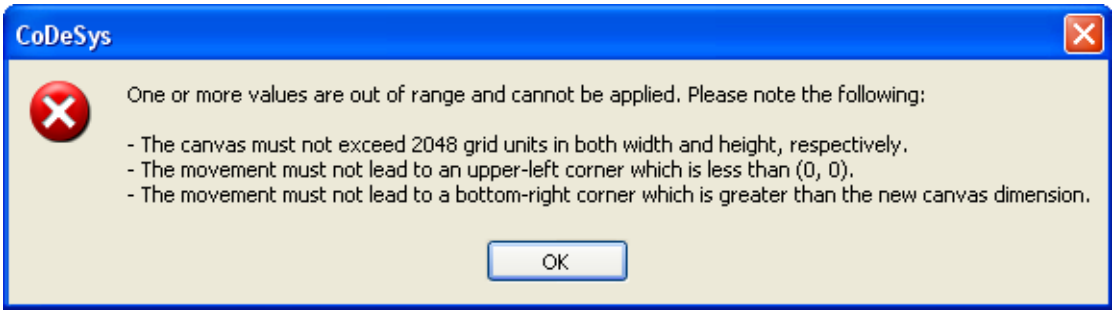


Abb. 75 Working Sheet size, error message dialog

12.7.3 Negate

Symbol:

This command (category CFC) is used to negate inputs, outputs, jumps or RETURN commands. The symbol for the negation is a small circle on the connection. To assign a negation select the respective input or output pins of the element and perform the command. See CFC Editor, Cursor Positions for the possible cursor positions for selection.

A negation can be deleted by negating again.

12.7.4 EN/ENO

Symbol: EN

This command (category CFC) is used to give a selected block (Cursor position 3.) an additional Boolean enable input EN and a Boolean output ENO (Enable Out).

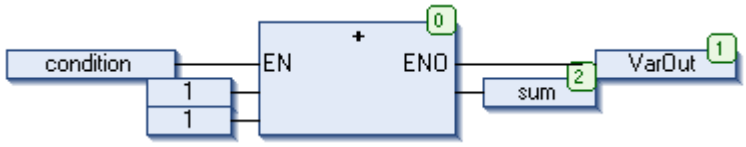


Abb. 76 ADD-box with EN/ENO

In this example ADD will only be executed if the boolean variable *condition* is TRUE. *VarOut* will be set to TRUE after the execution of ADD. Notice that if afterwards *condition* changes to FALSE, ADD will not be executed any longer and also *VarOut* will be set to FALSE.

The example below shows how the ENO value can be used for further blocks:

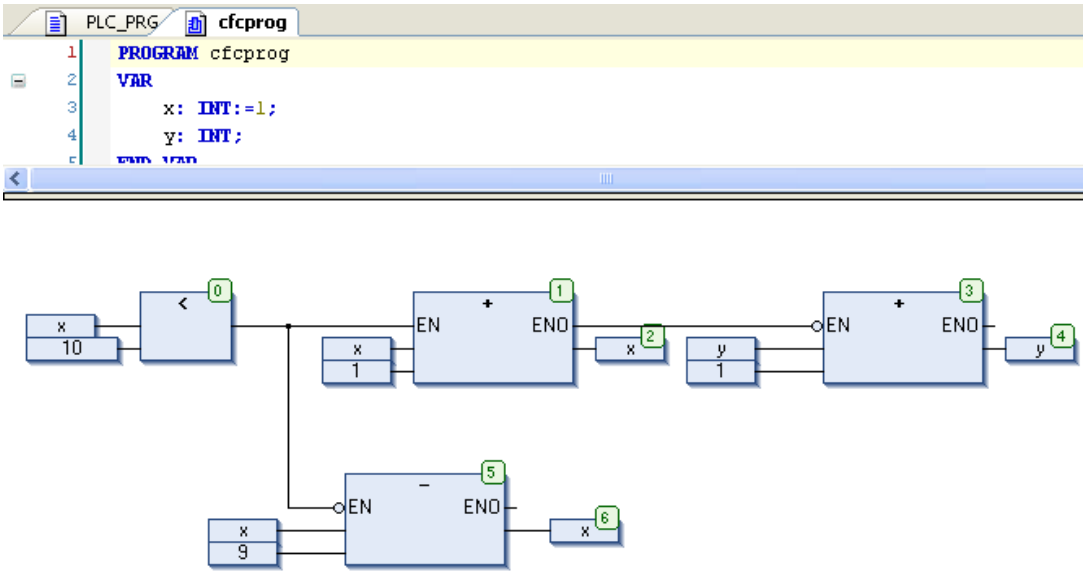


Abb. 77 Use of EN/ENO, example

For this example initialize *x* with "1". The numbers in the right corner of the boxes indicate the order in which the commands are executed.

As long as *x* is less than 10 (0), it will be increased by one (1). As soon as *x = 10* the output of LT(0) will deliver the value FALSE and SUB (6) and ADD (4) will be executed. *x* will be set back to the value 1 and *y* will get increased by 1. LT (0) will be executed again as long as *x* is less than 10. Thus *y* is counting how often *x* passes though the value range 1 to 10.

12.7.5 Set/Reset

None

Symbol:

This command (category CFC), per default part of the submenu 'Set/Reset' of the CFC menu, removes a "Set" or "Reset" from an output element. Select the input pin of the respective output and perform the command. The "S" or "R" symbol at the output element will disappear.

See CFC Editor, Cursor Positions for the possible cursor position for selection.

R - Reset

Symbol:

This command (category CFC), by default part of the submenu 'Set/Reset' of the CFC menu, assigns a "Reset" to a boolean output element, which means that the output will be reset by a TRUE of the input and keep this value even if the input becomes FALSE again.

Select the input pin of the respective output and perform the command.

See CFC Editor, Cursor Positions for the possible cursor position for selection.

The reset output will be indicated by a "R".

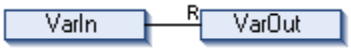


Abb. 78 Example

In this example *VarOut* will be set to FALSE, if *VarIn* delivers TRUE. *VarOut* retains this value, even when *VarIn* springs back to FALSE.

Alternative settings concerning the Set/Reset properties of an output are None, which means that there is no Set or Reset activity assigned, or S (Set).

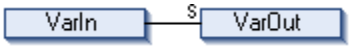
S - Set

Symbol:

This command (category CFC), by default part of the submenu 'Set/Reset' of the CFC menu, assigns a "Set" to a boolean output element, which means that the output gets set by a TRUE of the input and keeps this value even if the input becomes FALSE again.

Select the input pin of the respective output and perform the command.

See CFC Editor, Cursor Positions for the possible cursor position for selection.



In this example *VarOut* will be set to TRUE, if *VarIn* delivers TRUE. *VarOut* retains this value even when *VarIn* changes back to FALSE.

Alternative settings concerning the Set/Reset properties of an output are None, which means that there is no Set or Reset activity assigned, or R (Reset).

12.7.6 Execution Order

Send To Front

Symbol:

This command (category CFC) effects that all selected elements will be moved to the front of the execution order. Thereby the order within the group of selected as well as of the unselected elements is maintained.

By default the command is part of submenu "Execution Order" in the CFC menu.

Send To Back

Symbol:

This command (category CFC) effects that all selected elements will be moved to the end of the execution order. Thereby the order within the group of selected as well as of the unselected elements is maintained.

By default the command is part of submenu "Execution Order" in the CFC menu.

Move Up

Symbol:

The Move Up command (category CFC) effects that all selected elements - with the exception of the element which is at the beginning of the execution order - are moved one place forwards in the internal processing list.

By default the command is part of submenu "Execution Order" in the CFC menu.

Move Down

Symbol:

The Move Down command (category CFC) effects that all selected elements - with the exception of the element which is at the beginning of the execution order - are moved one place backwards in the internal processing list.

By default the command is part of submenu "Execution Order" in the CFC menu.

Order By Data Flow

This command (category CFC) per default is part of submenu "Execution Order" in the CFC menu. It effects that the execution order (indicated by the element numbers in the upper right corner of an element) in the CFC-Editor gets determined by the data flow of (all) elements and not by their position (topology).

The advantage of the order according to data flow is that an output box, which is connected to the output pin of a block, immediately will be processed after the block, which is not always so in case of a topological process flow. A topological order of processing might deliver another result in some cases than a processing by data flow. This can be recognized from the above described example.

Example

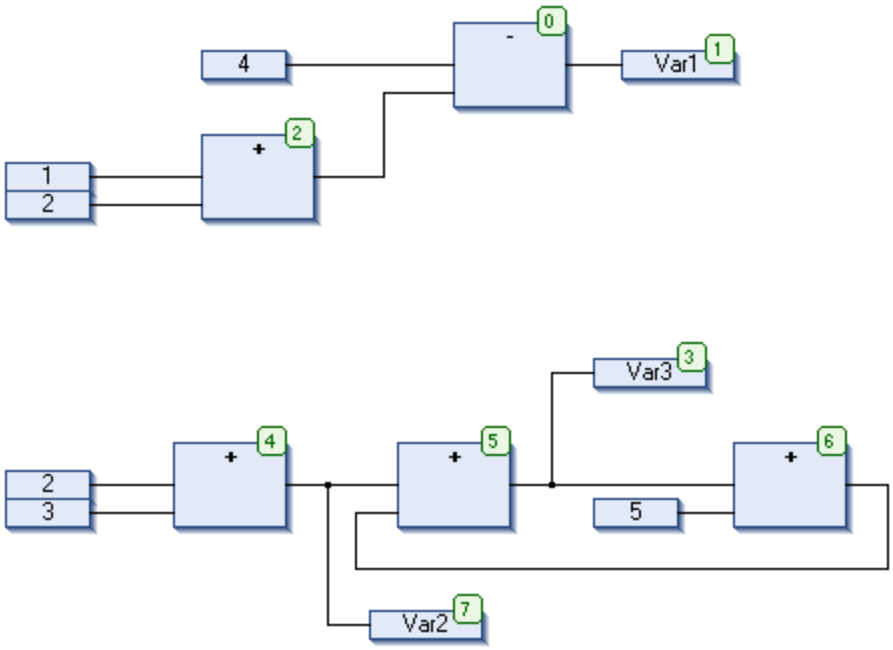


Abb. 79 Example, Before: Order by Topology

The following arrangement results after having used the command 'Order By Data Flow':

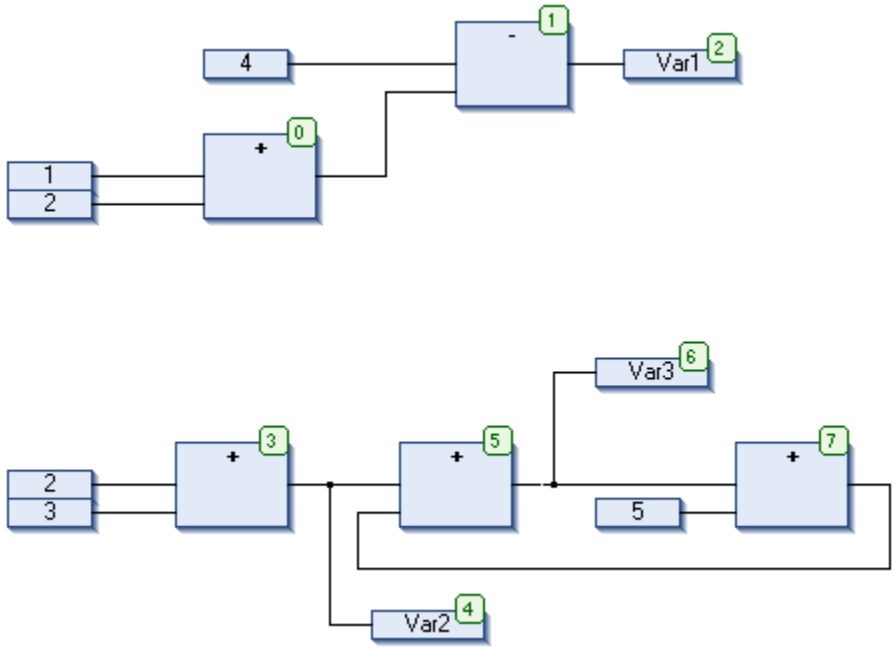


Abb. 80 Processing according to data flow

When the command gets executed, the following will happen internally: First the elements are ordered topographically. Then a new sequential processing list will be created. Based on the known values of the inputs, the computer calculates which of the not yet numbered elements can be processed next. In the above shown "network" e.g. the ADD block (0) could be processed immediately since the values at its inputs ("1" and "2") are known. Block SUB (1) can only be processed afterwards since the result from ADD must be known first, etc. Feedback paths get inserted last. So a sequencing by data flow will result.

Order By Topology

This command (category CFC) per default is part of submenu "Execution Order" in the CFC menu. It effects that the execution order in the CFC-Editor is determined by the topological order of the elements and not by the data flow.

Topological order means that the execution order, that is the **processing of the elements runs from left to right and from top to bottom**. The element numbers indicating the position of an element within the processing list, increase from left to right and from top to bottom. The position of the connection lines is not relevant, **only the location** of the elements is important.

When the command is executed, implicitly all currently selected elements get removed from the processing list and then re-inserted one by one in the remaining list **from bottom right through to upper left**. In doing so each selected element will be entered **before its topological successor** and the numbers of the remaining elements will be adapted.

Example, Topological arranging of selected elements

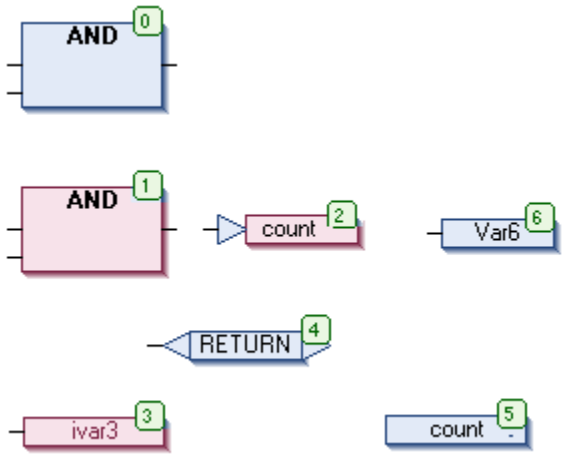


Abb. 81 Sequence before

The elements with numbers 1, 2 and 3 are selected. If now the command 'Order By Topology' is executed, the elements first will be taken out of the sequential processing list. The subsequent re-inserting will be done conversely:

First ivar will be inserted ahead of label count, thus getting number 4, which makes RETURN fall back to 3.

Then jump count gets inserted ahead of Var6 and thus gets number 5. This effects that label count (before then having 5), output ivar3 and RETURN each get numbered down by 1.

At last the AND box will be re-inserted ahead of jump count and thus will get number 4. This again effects a reducing of the numbers each for label count (before then having 4), output ivar3 and RETURN by 1. So the following new order of execution will arise:

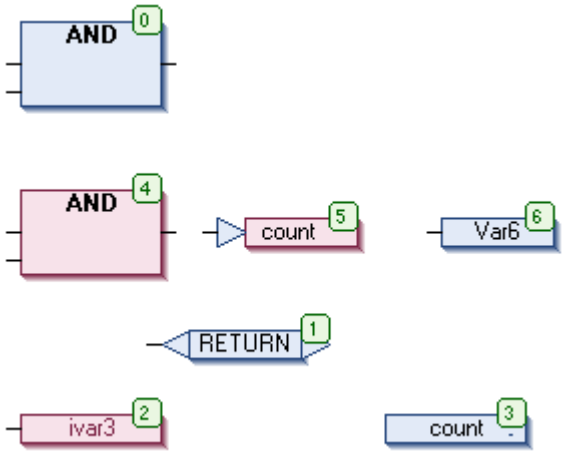


Abb. 82 Sequence afterwards

Also a **new element** always will be inserted in the sequential processing list ahead of its topological successor.

Set Execution Order

This command (category CFC) by default is part of submenu "Execution Order" in the CFC menu. It serves to redefine the element number of the currently selected element in order to change the position of this element within the execution order.

The command opens the dialog 'Set Execution Order'. The current element number is displayed in field **Current Execution Order** and you can enter the desired new one in **New Execution Order**. The possible values are displayed in brackets.

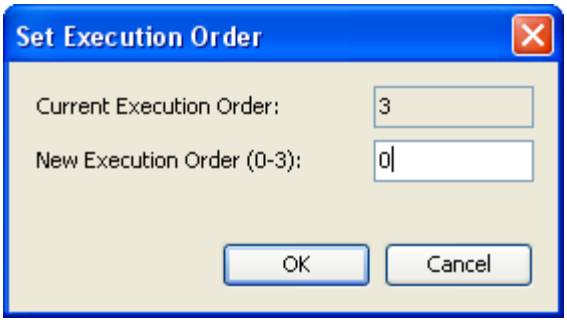


Abb. 83 Dialog 'Set Execution Order'

12.7.7 Edit Parameters...

Constant input parameters (VAR_INPUT CONSTANT) of functions and function blocks are not displayed directly in the CFC Editor.

Example:

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
fbin1:INT;
fbin2:DWORD:=24354333;
fbin3:STRING:='hallo';
END_VAR
```

To access those parameters use the 'Edit Parameters...' command (category CFC). It opens the 'Edit Parameters' dialog, where for each parameter the name (Parameter) the data type (Type) and the Initial value are displayed.

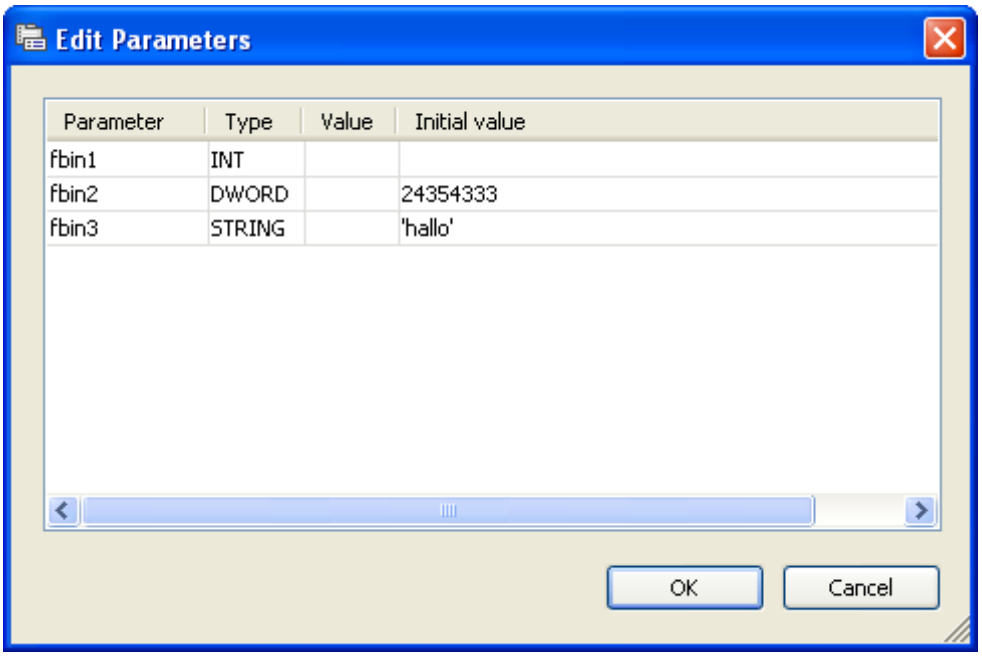


Abb. 84 Edit Parameters dialog

Here the values of a constant input parameter (VAR_INPUT CONSTANT) can be changed. For this purpose perform a mouse-click on the respective field of column **Value** to select it and a further click (or press <Enter>) to get an edit frame. After having modified the value use the <Return> key to confirm the modification. <Escape> will reject the modification. OK will store all modifications.

Note: This functionality and the associated declaration of variables with keyword "VAR_INPUT CONSTANT" is only of impact for the CFC editor. In the FBD editor all INPUT variables are displayed at a box, whether declared as VAR_INPUT or VAR_INPUT CONSTANT. For text editors this does not make any difference.

12.7.8 Connect Selected Pins

Symbol:

This command (category CFC) is only enabled when exactly one output pin and one or several input pins are selected. When executed, a connection between the output pin and the input pin(s) is established.

12.7.9 Reset Pins

Symbol:

If unused input or output pins have been removed from a box in the CFC Editor, for example because they are not used, or if the interface of the POU which is represented by the box has been changed, this command (category CFC) can be used to restore resp. update the display of the pins. It can also be used to display the parameters of type VAR_IN_OUT of a function block, which are hidden per default.

In the following example input fbin2 of a function block instance had been deleted because it is not used. By selecting the fb1 box and using command 'Restore Pins' all inputs and outputs of the function block, as defined in its implementation, can be displayed again.

```
1  FUNCTION_BLOCK fb1
2  VAR_INPUT
3      fbin1:INT;
4      fbin2:INT;
5  END_VAR
6  VAR_OUTPUT
7      fbout1:INT;
8  END_VAR
9  VAR
10     fbvar:INT;
```

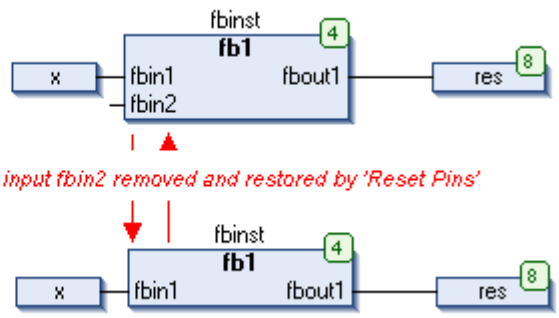


Abb. 85 Reset Pins, example

12.7.10 Remove unused pins

Symbol:

This command (category CFC) removes non-connected pins from program, function block or non-local action calls in the current editor selection. This will not be done for function, method or operator calls as this would lead to invalid syntax.

12.8 CNC

The command category 'CNC' provides commands for working in the CNC Editor.

12.8.1 CNC Commands

The commands of category 'CNC' are provided by the 'CNC Editor' plug-in, which is part of »PLC Designer« SoftMotion V3.

By default they can be found under the menu item 'CNC'. Else you may open the Customize dialog, category 'CNC', to view or modify the current menu configuration.

The commands:

- Show preprocessed path
- Show interpolation points
- Step Suppression
- Show grid
- Show end points
- Scroll path view
- Move program
- Scale program
- Rotate program
- Reverse direction
- Renumber CNC program
- Write program to ASCII file
- Load program from ASCII file
- Import from DXF file
- Analyze dynamics
- CNC Info
- Set Variables

12.8.2 Show preprocessed path

Button:

If this CNC option is activated, the preprocessed path will be displayed in the CNC editor. In difference to the preprocessed path the original path is displayed in the background in light gray and can be changed. The calculation of the preprocessing depends on the selected function blocks from the SM3_CNC library in the CNC settings dialog in tab **Path preprocessors**. There a set of function blocks can be chosen for computing the preprocessed path like SMC_ToolCorr, SMC_RoundPath, SMC_SmoothPath or SMC_AvoidLoop. Thus the effect of series connected path preprocessing elements can be simulated.

Note: This option has no effect on the code generation.

In case of code generation for SMC_OutQueue the path preprocessing is always calculated independent of the status of this option.

12.8.3 Show interpolation points

Button:

If this CNC option is activated, the preprocessed path is displayed with interpolation points. The interpolation time have to be defined in the CNC settings dialog in tab **Preinterpolation**. For identical behaviour with he later online execution, it should be conform with the task cycle time of the calling task. Default value is 20000 µs.

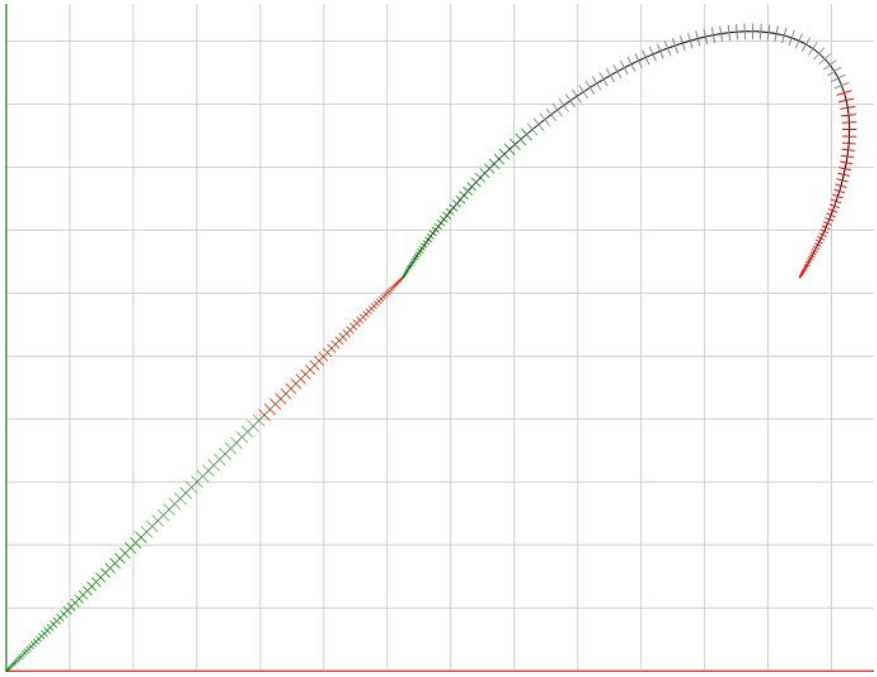


Abb. 86 Path with interpolation points

Every interpolation point is visualized with a line. The length of the line is proportional to the current path velocity. Thus a rough estimation of the velocity behaviour (fast = long distances and long line, slow = small distances and short line) is possible. If the interpolator is in acceleration, the line is green. In deceleration the line is red. Otherwise it is gray.

Depending in the option show preprocessed path the interpolation points are based on the preprocessed path, when the option is activated, or they are based on the original path, when not.

12.8.4 Step suppression

Button:

If this CNC option is activated, all lines of the text editor starting wiht "/" will be ignored. This command is not available in table view.

12.8.5 Show grid

Button:

If this CNC option is activated in menu 'CNC', a visible grid will be added to the graphic editor. The distance of the grid line depends on the zoom factor configuration. The grid's size depends on the extent of the path.

12.8.6 Show end points

If this CNC option is activated, all elements will be displayed with end points.

12.8.7 Scroll path view

If this CNC option is activated, the path view scrolls and zooms to the selected element, whenever a new line is selected in the CNC text editor or table editor. So the latest selected path element in the text editor is visible in the graphic editor.

12.8.8 Move program

This command being part of menu item 'CNC' opens the **Translation vector** dialog where a vector can be defined by which the CNC program will be shifted. The values for the following axes can be inserted: X, Y, Z, P, Q, U, V, W.

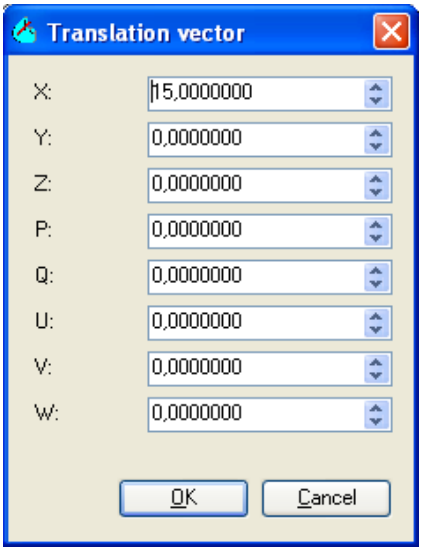


Abb. 87 Translation vector dialog

Before the first command of the following list (G1, G2, G3, G5, G6, G8, G9, G10, G11, G92) is executed a relative G0 command with the coordinates of the translation vector is inserted. If the first command is a G56 command, the translated program begins with a G56 code, too. The sentence numbers are assigned automatically.

It is not possible to move a CNC program including variables or using the axis A, B or C. IF the CNC program uses the shift coordinate commands (G54, G55, G56) the arguments of the commands were adapted and the following sentences are not changed.

12.8.9 Scale program

This command being part of menu item 'CNC' opens the **Scale program** dialog. Here the factor by which the CNC program should be stretched and the affected axis can be inserted. Only positive values equal or greater than 1e-3 are allowed.

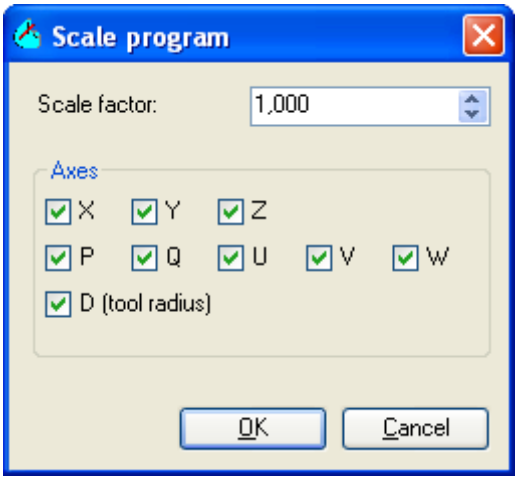


Abb. 88 Scale program

The sentence numbers are assigned automatically and it is not possible to scale a CNC program including variables or using the axis A, B or C.

12.8.10 Rotate program

This command being part of menu item 'CNC' opens the dialog **Rotation parameters** for rotating the current CNC program. There the desired angle can be defined. If the program path contains no 2D-elements or it contains only 3D-elements particularly no G5 splines, parabolas, ellipses or 2D-arcs, the rotation axis can be set. Default axis of rotation is Z-axis. Accordingly the program will be rotated counterclockwise when the angle is positive and clockwise when it is negative.

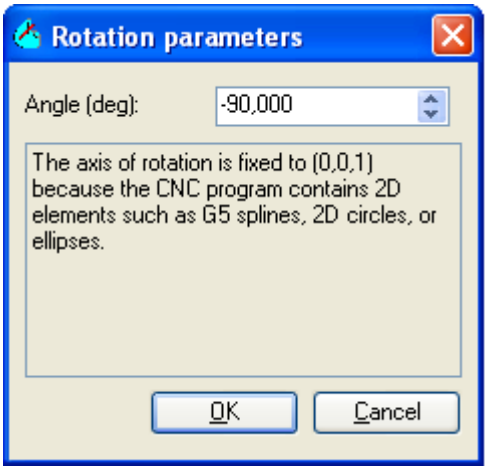


Abb. 89 Rotation parameters

Example

For rotation round the z-axis in clockwise direction with x-axis is horizontal and y-axis is vertical, put in a negative angle. It is not possible to rotate a CNC program including variables or using the A, B or C axis.

The sentence numbers are assigned automatically.

12.8.11 Reverse direction

This command being part of menu item 'CNC' can be used to reverse the programmed path. If the CNC program has got the following path elements, the reverse direction command is not possible:

- jumps (G20, G36 or G37).
- variables.
- step suppression.
- G56 or G75.
- axis A, B or C.

If the CNC program contains arcs defined with aperture angle (parameter K in 2D elements or parameter T in 3D) the command cannot generate an exact reverse arc. G41/G42 blocks will be exchanged in the reversed program. So G41 becomes G42 and G42 becomes G41 and the direction of the correction is exchanged.

The sentence numbers are assigned automatically.

12.8.12 Renumber CNC program

Button:

This command being part of menu item CNC' is used to trim the sentence numbers of the path program in the text editor (N<number>) automatically. So every sentence with no sentence number gets one starting with 0 and is incremented by ten. Jump targets (L-codes in G20 sentences) are corrected automatically,too. This command is not available in table view.

12.8.13 Load program from ASCII file

This command being part of menu item 'CNC' is used to load a CNC program which is stored before in ASCII format. The standard dialog for opening a file provides a list of files for selecting the desired one. Possible endings are ".txt" or ".cnc", default is ".gcode". With clicking button **Open** and acknowledging this the ASCII file is loaded into the CNC editor with replacing the latest CNC program.

12.8.14 Write program to ASCII file

This command being part of menu item 'CNC' is used to write the current CNC program to a file in ASCII format. Possible endings are ".txt", ".gcode" or ".cnc". If the defined file already exists , »PLC Designer« will ask for a confirmation.

12.8.15 Import DXF file

This command being part of menu item 'CNC' by default is used to import a DXF file into the CNC editor with the current CNC object name. The standard **Open** dialog opens where you can browse for the desired DXF file. Then the **Import DXF file** dialog created from the DXF file opens.

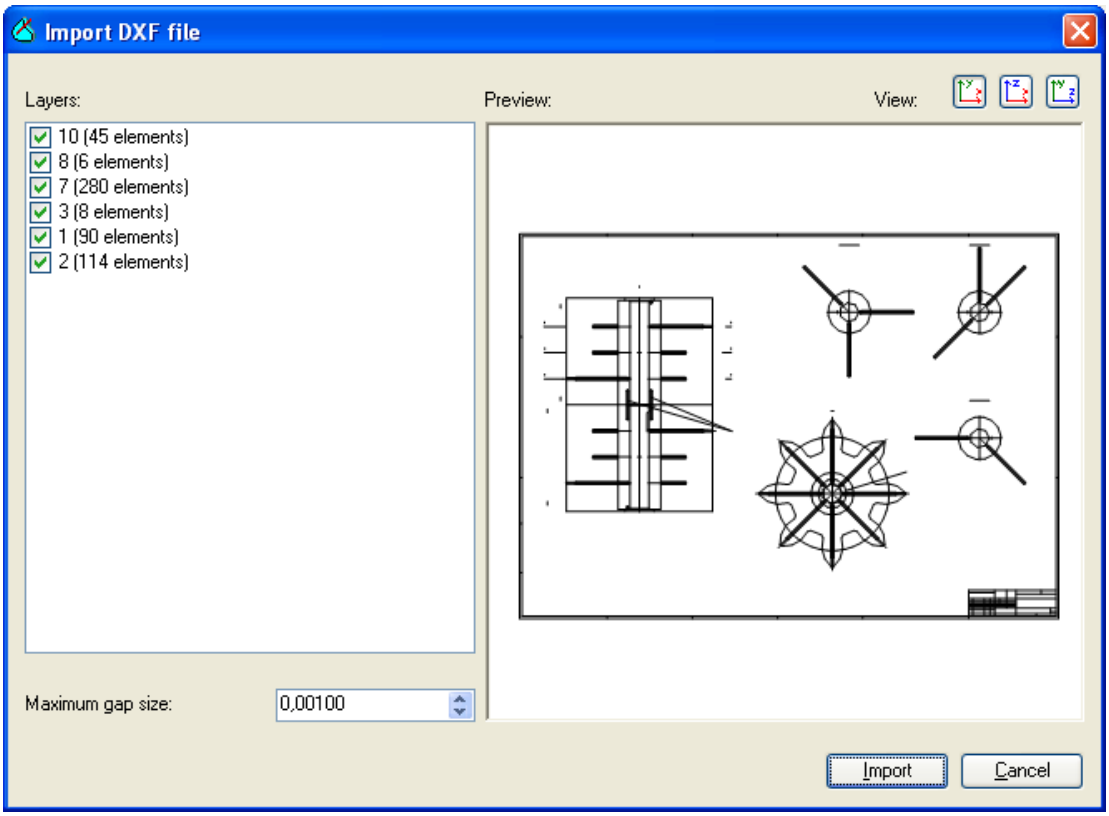


Abb. 90 Dialog Import DXF file

In the left part of the dialog a list of all layers with check-boxes of the imported file is shown. Here it is possible to select the layers which are determined for import. In the right part a preview of the CNC path visualize it. With the **View** buttons the view level can be changed between x/y, x/z and z/y plane. The **Maximum gap size** defines the distance of the end points of the elements in the DXF file up to which the points are considered to be connected. With clicking button **Import** and acknowledging this the DXF file is loaded into the CNC editor with replacing the latest CNC path. Please use undo when the import doesn't fit . The sentence numbers are assigned automatically.

The DXF standard is supported with following restrictions:

- Splines are not imported exact.
- No external references.
- Only ellipses layered in the x/y plane.
- Only arcs and circles with a scaling indifferent in all space axis.
- The scaling factor of block references has to be positive.
- No text, mtext or dimension elements.
- Visibility of elements, style, color, and line width is ignored.

12.8.16 Analyze dynamics

This command being part of menu item 'CNC' is used to analyze the dynamics or the associated CNC path. The dynamics of the path are visualized in four diagrams with a common time axis for the selected space axis: position, velocity, acceleration and jerk. With clicking the command the **Analyze dynamics** dialog opens after the path preprocessing is done without failure.

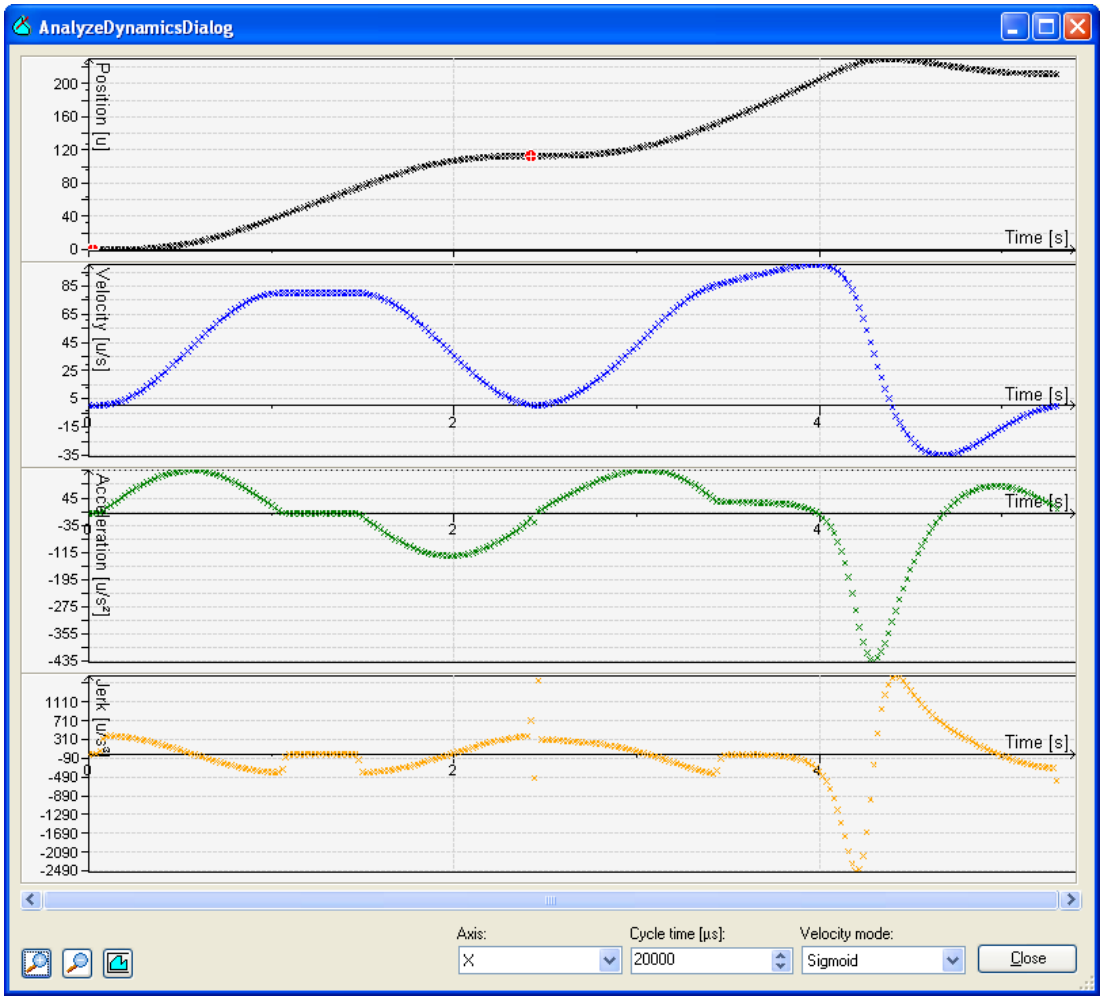


Abb. 91 Analyze dynamics

Default setting for axis is X axis, for cycle time the cycle time as defined in **CNC Settings** is used as initial value and velocity mode is trapezoid. The zoom factor is so adjusted, that the entire CNC path is visible. The changed settings are stored after closing the dialog, so by starting the Analyze dynamics dialog again the same settings will be selected. (Changing the CNC Settings then have no effect on the analyze.) Particularly the zoom factor, the scroll position, the selected axis, and the interpolation settings are stored.

Note: In general the values of the settings here are only used for the dynamics' analyze and they have no further effect to the CNC program or the CNC settings.

In all diagrams the set of data points are visualized with little crosses, so jumps are visible.

Position:

The top diagram shows the position of the selected axis over the time in distance units in black. A red circle in the diagram visualize the time stamp, where a path element starts. The tool tip of the red circle informs about the line number in the associated G-code or table.

Velocity:

The second diagram shows the velocity of the selected axis over the time in distance units per second in blue. It is conform to the one-dimensional path velocity.

Acceleration:

The third diagram shows the acceleration of the selected axis over the time in distance units per second squared in green.

Jerk:

The bottom diagram shows the jerk or the current axis over the time in distance units per second cubed in orange.

Zoom Buttons:

Use this buttons for changing the zoom factor so that the shared time axis is scaled up or down. Use the third button to reset the factor so that the entire CNC path is visible again. The same can be done with the **Scroll Wheel** while pressing the **Ctrl key**, or **key +** and **key -** while pressing the **Ctrl key**

Axis:

Use the pull-down menu to select a space axis. Select between X, Y, Z, A, B, C, A1(P), A2(Q), A3(U), A4(V), A5(W), A6 and Path. The special axis 'Path' will show the position on the path and the path velocity, acceleration and jerk.

Cycle time [µs]:

Change the cycle time with the pull-down menu or by editing. This value is used as input dwIpoTime of the function block SMC_INTERPOLATOR.

Velocity mode:

Select between trapezoid or sigmoid. This value is used as input iVelMode of the function block SMC_INTERPOLATOR.

12.8.17 Set variables

This command being part of menu item 'CNC' is used to set the values of CNC variables in offline mode. When using this command the **Offline values of variables** dialog opens. All variables used in the associated CNC path are listed in a table there. In the left row all used variables and in the right column the current values are listed. The values can be edited with clicking in the table entry. Then a inline editor opens. By clicking the **OK** button valid values were accepted. Variables displayed in gray cursive font are no longer used in the program.

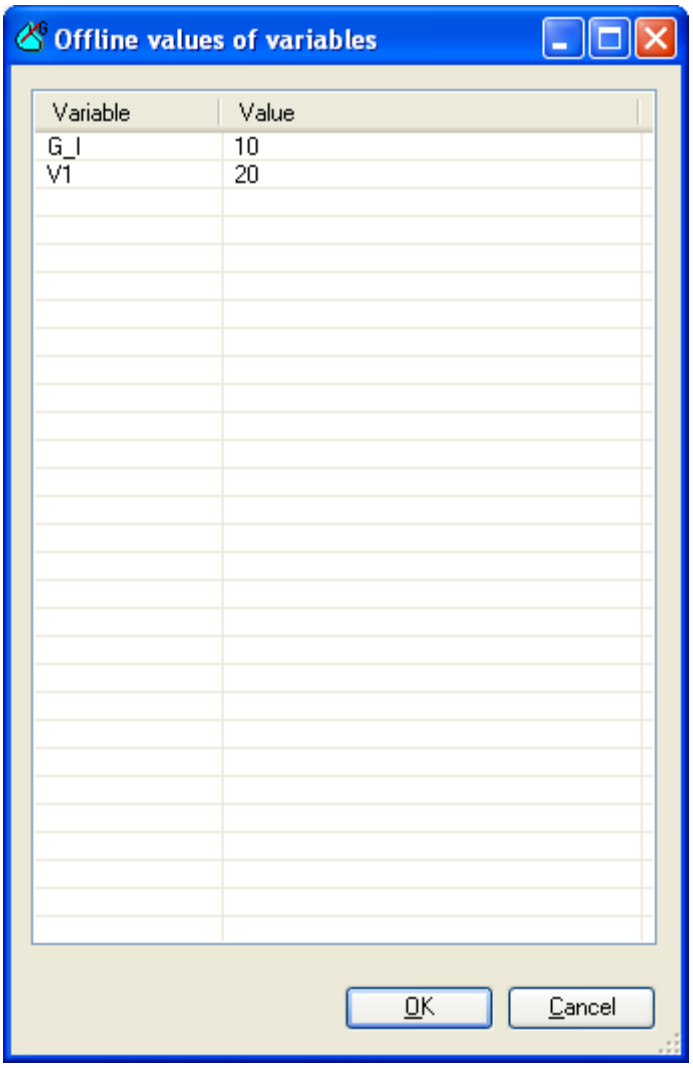


Abb. 92 Offline values of variables

12.9 Clipboard

The command category 'Clipboard' provides the common functions to handle contents between project and clipboard.

12.9.1 Clipboard commands - Edit commands

The 'Edit Commands' plug-in provides commands for command category 'Clipboard' which are used for moving and copying object entries resp. strings in editors or view trees.

By default these commands are part of the 'Edit' menu. If necessary, open the Customize dialog, category Menu, to view or modify the current menu configuration.

The commands:

- Cut
- Copy
- Paste
- Delete
- Select all

12.9.2 Cut

Symbol:

Default Shortcut: <Ctrl>+<X> or <Shift>+<Delete>

This command (category 'Clipboard') transfers the current selection (object entry, string) to the clipboard. The selection is removed from the editor resp. object tree.

In tree structures which are used to organize objects, as for example in the POUs view, this applies to the selected object. Multiple selection is possible.

Remember that not all editors support the 'Cut' command, and that its use can be limited in some editors.

The form of the selection depends upon the respective editor: For example it is a string or character in text editors resp. might be one or several elements surrounded by a selection frame in graphic editors.

In order to paste the content of the clipboard you use the Paste command.

In order to copy a selection onto the clipboard without deleting it, use the Copy command.

In order to remove a selected area without changing the clipboard, use the Delete command.

12.9.3 Copy

Symbol:

Default Shortcut: <Ctrl>+<C>

This command (category 'Clipboard') copies the current selection from the editor to the clipboard. This does not change the contents of the editor window.

In tree structures organizing objects as for example in the 'POUs' view, this applies to the selected object. Multiple selections are possible.

Remember that not all editors support copying and that it can be limited with some editors.

For the type of selection the same rules apply as with the Cut command.

In order to paste the content of the clipboard you use the Paste command.

In order to delete a selected area and simultaneously put it on the clipboard, use the Cut command.

12.9.4 Paste

Symbol:

Default Shortcut: <Ctrl>+<V>

This command (category 'Clipboard') pastes the content of the clipboard onto the current position in the editor window.

Pasting is not supported by all editors resp. its use might be limited. In graphic editors the command is only supported if a correct structure will result from the insertion.

In tree structures which are used to organize objects, as for example in the POUs or Devices view, the command applies to the selected object. Multiple selection is possible. Depending on the current position, for example in the devices tree, a dialog might be opened where you have to choose, whether the object from the clipboard should be entered below or above.

In order to copy a selection onto the clipboard without deleting it, use the Copy command.

In order to remove a selected area without changing the clipboard, use the Delete command.

12.9.5 Delete

Symbol:

Default Shortcut: <Entf>

This command (category 'Clipboard') deletes the selected area from the editor window. It does not change the contents of the clipboard.

In the Object Organizer the command applies to the selected object.

For the type of selection the same rules apply as with the Cut command.

In order to delete a selected area and simultaneously put it on the clipboard, use the Cut command.

12.9.6 Select all

This command (category 'Clipboard') can be applied in editors to select the complete content.

12.10 Declaration

The command category 'Declaration' provides commands for working in the tabular declaration editor.

12.10.1 Tabular Declaration Editor - Declaration

The 'TabularDeclarationEditor' plug-in provides the following commands of category 'Declaration' for working in the tabular declaration editor:

- Edit Declaration Header
- Insert
- Move down
- Move up

The commands by default are available in the context menu and in the toolbar of the editor. For detailed information please see the description of the tabular declaration editor.

12.10.2 Edit Declaration Header

This command (category 'Declaration') opens the editor for the declaration header. For detailed information please see the description of the tabular declaration editor.

12.10.3 Insert

This command (category 'Declaration') adds a new line for a variables' declaration. For detailed information please see the description of the tabular declaration editor.

12.10.4 Move down

This command (category 'Declaration') moves the currently selected declaration line one down. For detailed information please see the description of the tabular declaration editor.

12.10.5 Move up

This command (category 'Declaration') moves the currently selected declaration line one up. For detailed information please see the description of the tabular declaration editor.

12.11 Devices

The command category 'Devices' provides commands for installing/uninstalling and handling resource objects of type 'Device' in the Devices view window.

12.11.1 Devices commands

Devices commands

The functions for managing the devices tree in a project mainly belong to the command category 'Devices'.

By default the commands are in the 'Project' menu. For re-organizing the menu structure the Customize dialog can be used.

Available commands:

Add Device...

Insert Device...

Generate EtherCAT XML

Generate SercosIII SCI xml

Plug Device

Update Device

Disable Device

Add Device...

This command (category Devices) is used to add a device object, representing a hardware module, **indented below** the currently selected object in the device tree. Notice that in order to insert an object on the same level, command Insert Device... must be used.

The command opens the **Add Device dialog**. Depending on the currently selected position in the devices tree you will get a list of devices available for insertion. As a precondition the devices, basing on a device description file, **must be installed** on your system, which can be done by the Device Repository dialog.

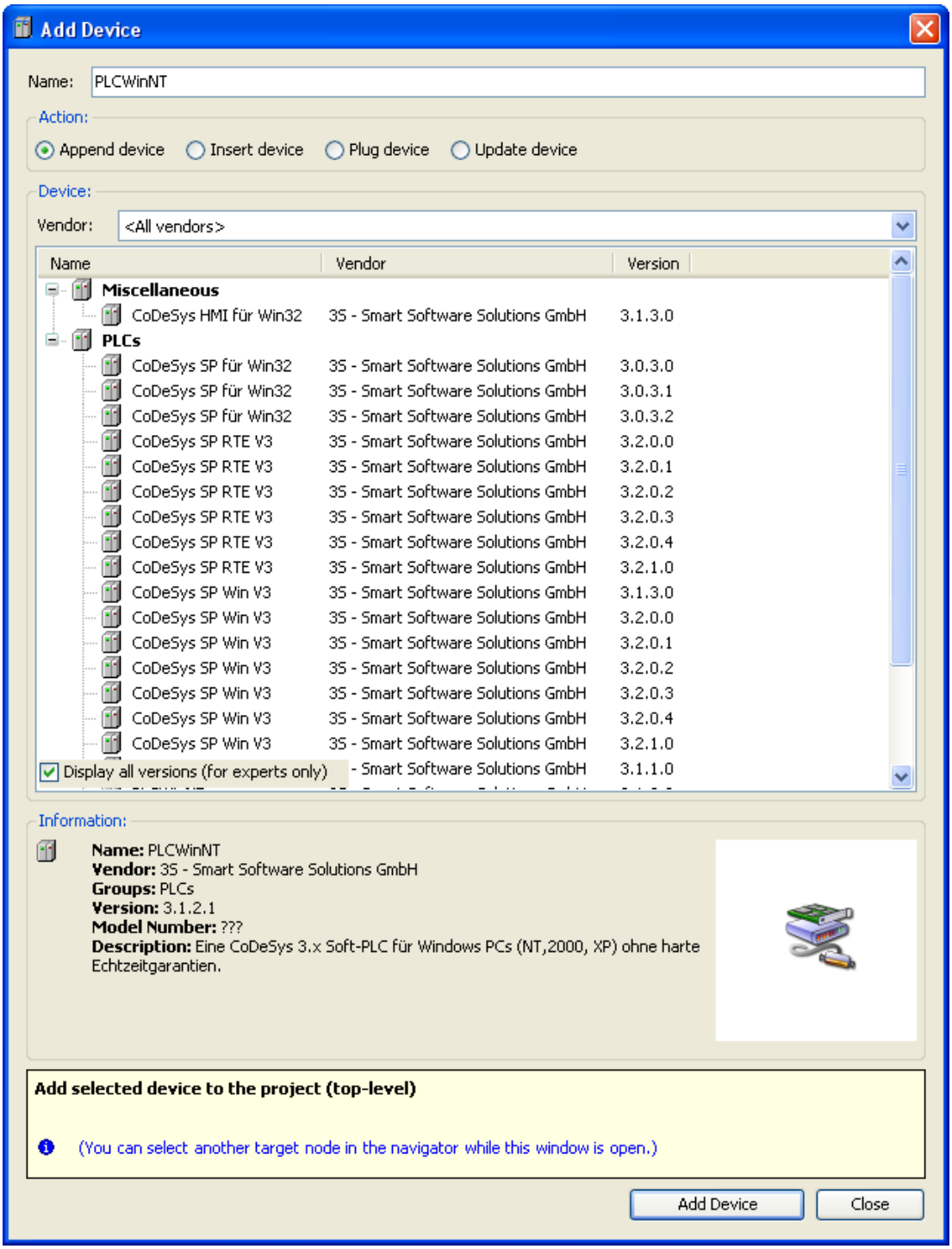


Abb. 93 Add Device dialog

In the **Name** field at top of the dialog define a name with which the device should be entered in the devices tree. This name must be a valid and unique IEC identifier.

Action:

By activating the corresponding item you may select which of the actions listed shall be executed: **Append device**, Insert device, Plug device or Update device. Your choice will effect the list of proposals in the sector "Device" as well as the labeling of the button placed at left of the button **Close** for closing the dialog.

Device:

The scrollable table in the 'Add Device' dialog provides a list of devices which can be inserted at the current position of the devices tree.

The devices are listed with **Name** and **Vendor**. By activating the checkbox **Display all versions (for experts only)** all available versions of the particular devices will be displayed as well. The devices might be grouped, that is the table entries (category names) indented in the column **name**, like for example "Miscellaneous" in the dialog image shown above. To open or close the groups use the preceding plus- resp. minus-signs.

By default initially all available devices will be displayed. If you want to restrict the display on a certain vendor, correspondingly modify the current selection (<All vendors>) in the **Vendor** list above the table.

Information:

For the currently selected device additional information - as provided by the device description file - might be displayed below the table: Device name, Vendor, Groups (might be defined for categorization purposes), Version, Order Number and a short Description. A device-specific image might be added also.

Depending on your action type chosen you have to confirm your selection by clicking on **Add device** / **Insert Device** / **Plug Device** or **Update Device**, whereon the corresponding action will be executed. Be aware that you are able to add/insert/plug/update further devices without reopening this dialog. Thereby you may select the appropriate node by a click in the device tree.

Having finished you may quit the dialog by pressing **Close**.

Notice: A device can also be added to the devices tree by the 'Add Object' dialog. Using the 'Add Device' dialog provides the advantage of getting displayed additional information on the device.

Insert Device...

This command (category Devices) is used to insert a device object, representing a hardware module, on the **same level and above of** the currently selected object in the devices tree. Note that in contrast via command Add Device... a device object can be inserted indented below the currently selected.

The command opens the Add Device dialog, where you can choose one of devices available for the current position.

Generate EtherCAT XML

This command (category Devices) is used to save the EtherCAT configuration of the current project in XML format. After issuing this command you will be asked, where you want to store the output .XML file.

Generate SercosIII SCI xml

This command (category Devices) is used to save the SercosIII SCI configuration of the current project in XML format. After issuing this command you will be asked, where you want to store the output .XML file.

Plug Device

This command (category Devices) is used to plug a device object, representing a hardware module, to the currently selected **slot** in the devices tree. An empty slot is identified by icon and entry "<Empty> (<Empty>)". An already occupied slot shows icon and the device name.

The command opens the Add Device dialog, where you can choose one of devices available for the current position. In case of an occupied slot the existing entry will be replaced by the new one.

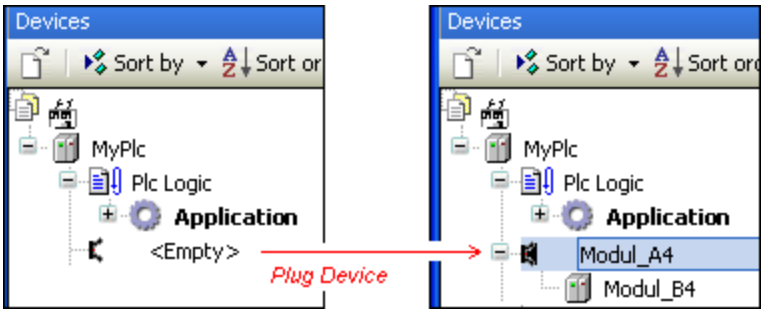


Abb. 94 Example, Plug Device

Update Device

This command (category Devices) is used to replace the device object currently selected in the device tree by another one. This can be used either to change to another version of the device or to replace it at all by another device type.

Like at normal inserting in the configuration tree, the Add Device dialog will be opened, where you can choose one of the installed devices available for the current position. The chosen device will be inserted instead of the previous one. The symbolic **device name**, used in the device tree, will be maintained, the description of the **device type** (in brackets behind the device name) will change if a different type has been chosen. Thus, if just the device version has got changed, the object entry will look like before.

The configuration tree indented below the respective device entry, that is submodules etc. will be maintained. The device configuration settings will be kept, if the device type has not been changed. Any mismatches in the existing configuration, caused by the device update, will be detected at the next build run of the application and appropriate error messages will be generated. This might also concern implicitly added libraries which will not be removed automatically and appropriately at a device update.

Disable Device

This command (category Devices) is used to activate resp. deactivate (toggle command) a device in the current PLC configuration on the bus system. A disabled, passive device will not be noticed and evaluated. Notice however, that the bus driver must support this function. Notice also that - depending on the bus system - disabling one of the slave devices might prevent the master device at all from working.

The command is only available in offline mode, by default in the context menu.

A disabled device entry is displayed in light-grey letters. When going online, the non-active devices additionally will be marked by red triangle symbols.

To re-enable a device, select the entry in offline mode and use the command once more.

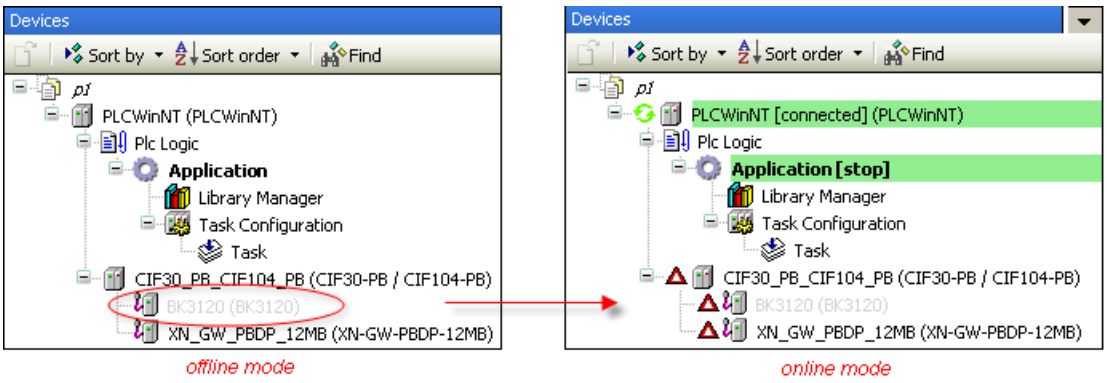


Abb. 95 Example, Profibus Slave disabled

12.11.2 Scan

Devices commands - Scan

The command Scan For Devices for scanning the hardware environment of the currently connected PLC is provided in command category 'Devices'.

By default it is part of the 'Project' menu. If necessary, the menu structure can be re-configured via the Customize dialog.

Scan For Devices

This command (category Devices) starts a scan of the hardware environment of the currently connected PLC. This means to **detect and view the hardware components** in a dialog and to enable the user to directly map them in the device tree in the project.

How the scan-functionality works, depends on the network type. The whole functionality might be implemented in the PLC, which then requires that you log in before executing the Scan command, or a library might be necessary which must be included in the project.

In each case a connection to the PLC will be established automatically before a scan function is executed, and afterwards will be closed again also automatically. Thus the gateway connection must be configured and the PLC must be running before a scan can be done. If the scan functionality completely is implemented in the PLC, a scan can be done without further preparations. If scanning needs library functions, at least once a login must have been done before the first scan, in order to get the library downloaded. If this has not been done, the user at his first trial to scan will be prompted to load the project (and library) to the PLC.

The command always refers to the controller currently selected in the device tree. For example an already inserted PROFINET IO Master module can be selected and via the scan command its slave devices and IO modules can be detected.

The command opens the **'Scan devices' dialog**:

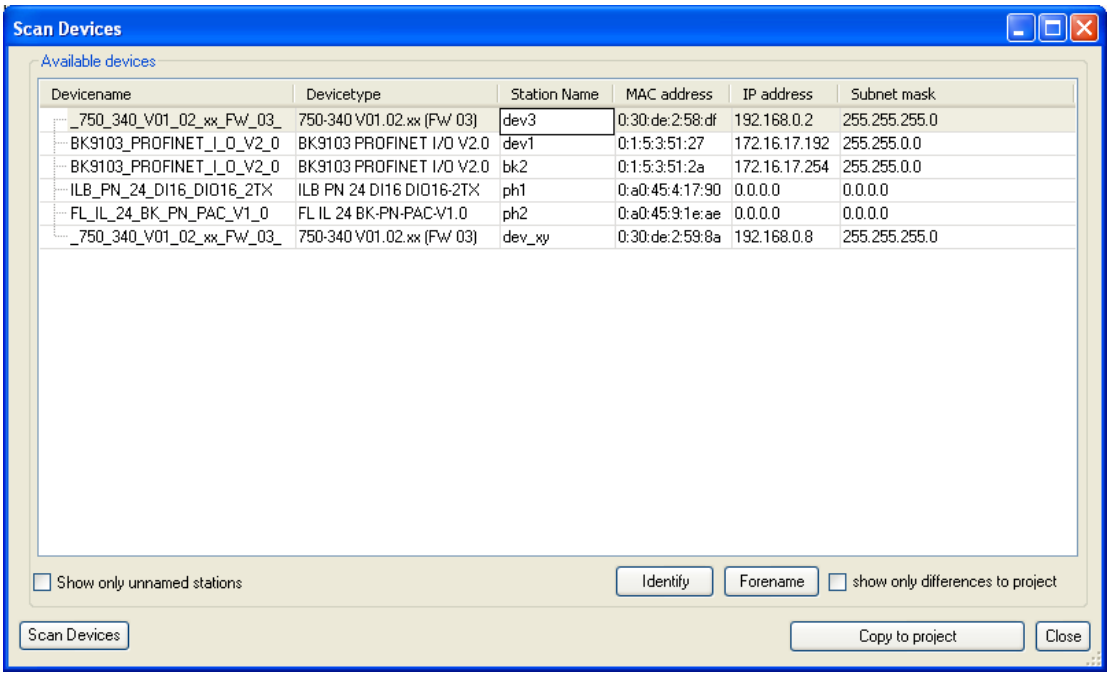


Abb. 96 Dialog "Scan Devices", example

The **Available devices** field might be still empty or contain a list of all devices and modules found during the last scan.

Which information is displayed for the particular devices and which options are available in the dialog, depends on the current network type.

In each case the following functions are available:

- In field "Available devices" resp. "Devices" the **Device name** and **Device type** of the found devices will be shown. Depending on the device type also the Station Address, Ident Nr and State can be displayed.
- Via button **Scan devices** a new scan can be started. If option **only show differences to project** is available and activated, only those devices will be listed, which are not yet represented in the device tree..
- Via button **Copy to project** the devices currently selected in the 'Available devices' field, resp. if nothing is selected, all listed devices will be mapped to the device tree in the project. Depending on the device type first the standard dialog for adding a device might be opened for selecting the device to be copied to the project (for example for Profibus devices)

Additional functions for PROFINET IO devices:

- For each device in further columns of the 'Available devices' list also the **Station name**, **Station Address**, **MAC address**, **IP address** and **subnet mask** are displayed.

Note: Each device must get a **station name**, because this is needed for certain functions during network operation.

- If the filter **only show unnamed stations** is activated, only those devices will be listed, for which not yet a station name is defined.
- The **station name** can be modified here in the dialog (open edit field by double-click on table cell). In order to get the new name used in the bus system, select the device entry and use button **Forename**. Thereupon appears a note that the bus must be switched off and a new bus configuration is required (by reset or application download) in order to get the change effective. The user will be responsible for this! The station name shown in the scan dialog also will be used in the device tree in the programming system.
- If the **device name should be used as station name**, use button **Forename**. You will be asked whether the device name should be used as station name, answer Yes. If you choose No, because you after all you have decided to define another name, then you have to enter this name manually in the respective field in column 'Available devices'. For this purpose via a mouse-click on this field an edit frame can be opened.
- Also the entries for device name, IP address and subnet mask can be **edited here in the dialog**, resp. can be defined here, before the device gets mapped to the configuration tree in the project If a device gets mapped without having defined addresses, those presettings will be applied, which are defined in the PNIO Master Parameter dialog of the PROFINET IO device editor for such cases.. The final definition of the addresses always can be still done in the configuration dialog of a device (PNIO identification).
- In order to identify a device which is listed in the scan-dialog at the hardware itself, select the respective entry and use button **Identify**. This should cause blink signal at the hardware device.
- The module **Ident number** will only be shown in the list, if (1) it is provided by the slave device, (2) if slave and master are part of the same logical network and (3) if IP address and network mask of the slave are set correctly. If the ident number is not available, a slave device can be "copied" to the project anyway as long as it can be identified uniquely by the available information.
- Regard the possibility of synchronizing the configuration settings of a PROFINET IO slave device defined within the project with those of the respective hardware device.

Additional functions for Profibus DP devices:

The Profibus scan shows a list of all available Profibus Slaves. To each slave information about station address, ident number, status (comparison with slaves of the device tree of the project) and the IO configuration defined by the slave will be displayed. The mapping of the scanned slaves and the slaves of the project configuration is based on the station address. If there is no slave in the device tree with the same station address of a scanned slave the state is set to "New". If a slave with the identical station address exists the scan also compares the module configuration. Any difference in the modul configuration of the corresponding slaves will be shown with the red error icon and an error message. If the modul configuration is identical the state is set to "OK". With the button Copy to project the selected device will be mapped directly to the device tree.

The information provided by the scanned slave (Id number, IO configuration of the module) do not allow a unique assignment to specific devices of a supplier. Therefor, when copying the devices to project, the "Add Device... " dialog will open where the scanned target devices can be assigned to the installed devices of the project manually.

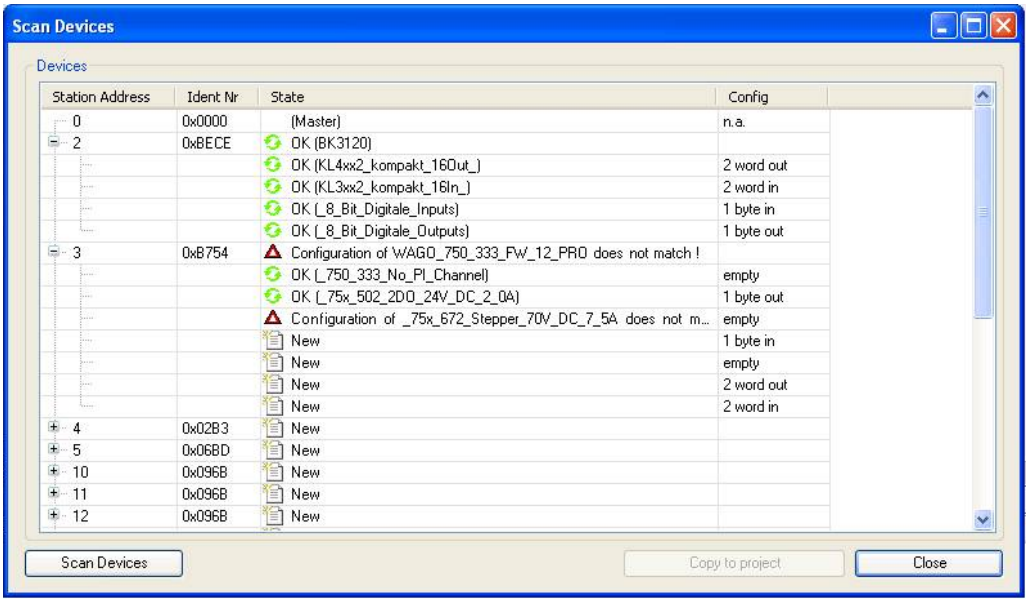


Abb. 97 Dialog "Scan Devices" for Profibus DP

Additional functions for EtherCAT devices:

If you have added a EtherCAT Master module and you scan for devices you will get a list of all available EtherCAT Slaves. Analog to device scans of other fieldbus devices you will get entries written in bold if there exist more than one device with the same device description. With a double click on this entry a list will open and the required device can be selected.

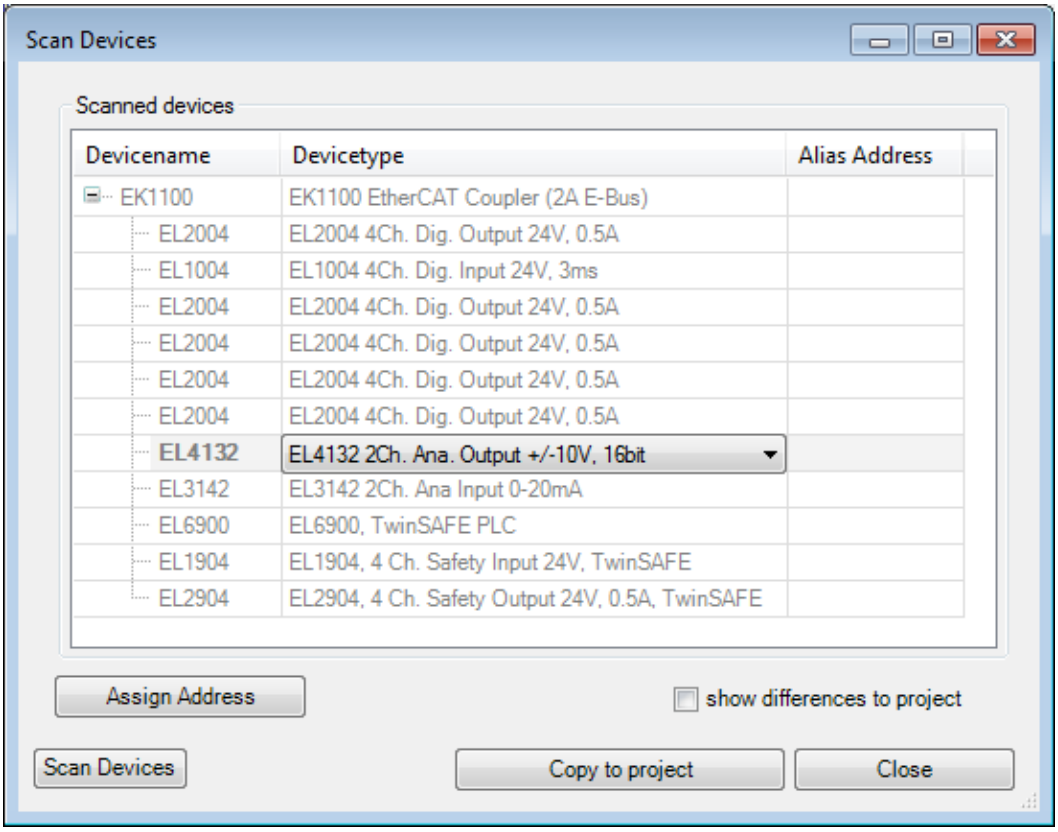


Abb. 98 Dialog "Scan Devices" for EtherCAT

Sync Config

Note that this functionality is only available for PROFINET IO bus systems!

The command (category Devices) provides a comfortable way to synchronize the configuration settings of a PROFINET slave device, defined within the project, with those defined on a corresponding device (same device id and vendor id) in the connected hardware. That means that the local settings will be overwritten with the hardware settings.

Perform the following steps: Set the appropriate communication path. Then in offline mode select the PROFINET slave in the device tree and use command "Sync Config" from the context menu. The **Scan Devices** dialog will open and a search for matching devices on the hardware will be executed. The found devices will be listed. You can fade out those, which are already used within the project (identified by their station name), by activating option **Hide used devices**. You can restart the scan process by button **Scan Devices**.

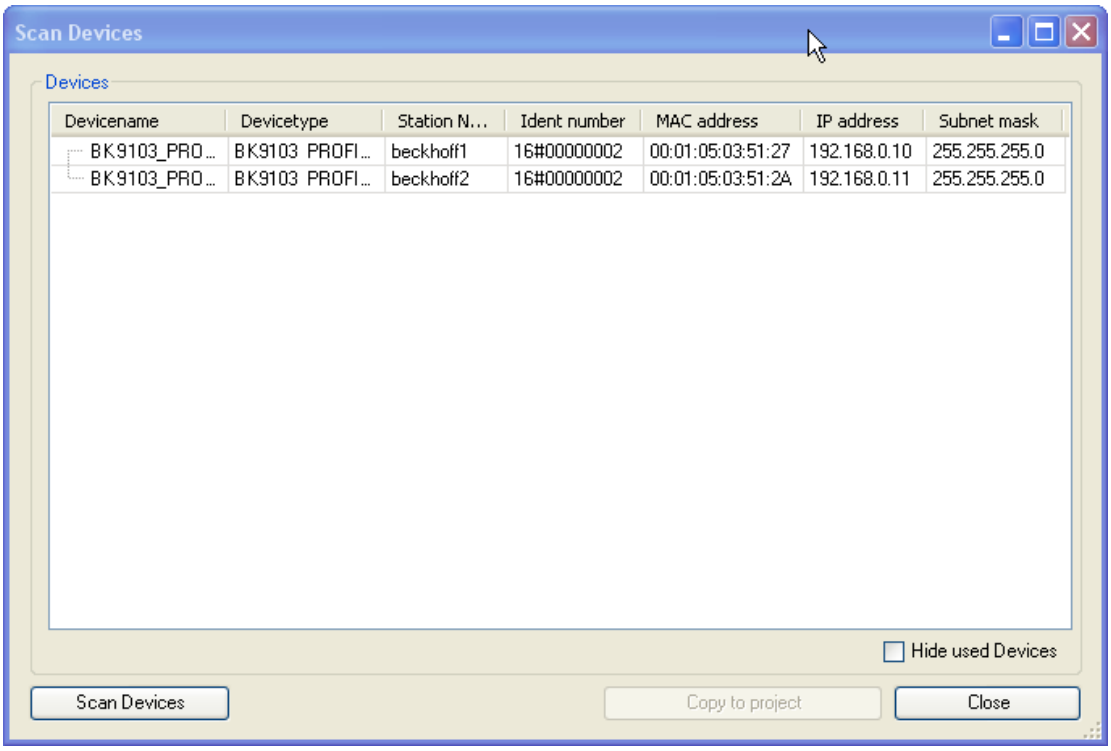


Abb. 99 Dialog "Scan Devices" for offline configuration synchronization, example

To overwrite the configuration settings of the device in the project with those from the hardware device, select the desired device in the list and use button **Copy to project**. Possibly added subdevices will not be removed by this action.

12.11.3 Device Repository

Devices commands - Device Repository

The functionality for managing the devices on your system and project is i.a. provided by the 'Device Repository' plug-in. It contains commands of category 'Devices' which by default are items of the 'Tools' menu. If you would like to re-organize the menu structure, use the Customize dialog.

The following commands are available:

- Device Repository...

Note: By default also the DeviceObject plug-in will be available, providing dialogs and functions for handling the devices in your project.

Device Repository...

Note: This functionality is only available if supported by the currently used feature set.

This command (category 'Devices') opens the dialog 'Device Repository'. A Device Repository is a database for the devices which have been installed on the local system for the purpose of being available in the »PLC Designer« development system. In the Device Repository you can add or remove such device installations:

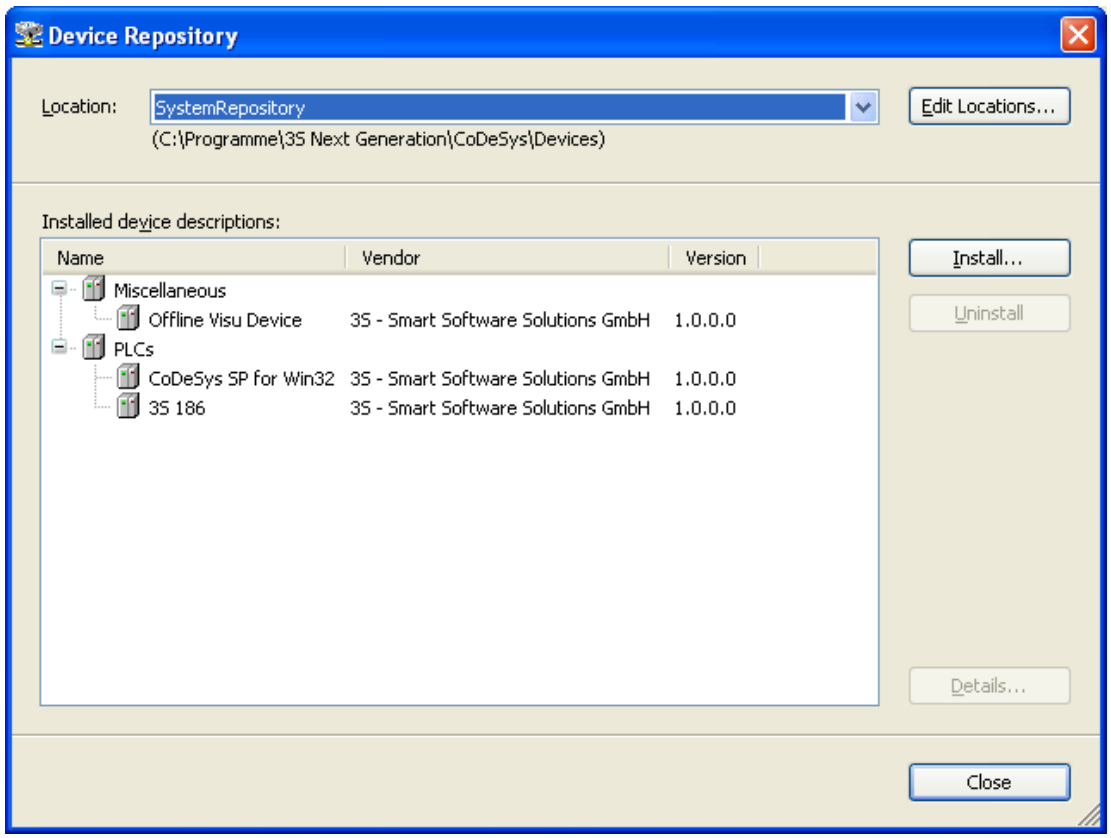


Abb. 100 Device Repository dialog

Location: Device repositories can be available on multiple locations on the system. The selection list offers the currently available locations. By default with the »PLC Designer« installation at least the "System Repository" is provided.

Installed device descriptions: The currently installed devices are listed in a tree structure, each showing the **Name**, **Vendor** name and **Version** of the device. The devices tree may be structured by categories like for example "PLCs" and "Miscellaneous" (see above, picture of Device Repository dialog). If applicable open or close the tree entries via the plus- and minus-"buttons".

Details...: This button opens for the currently selected device the 'Details'-dialog showing additional information as given by the device description file: Device name, Vendor name, Categories, Version, Order Number, Description.

Install...:Use this button to get a device installed for being available in the programming system. The dialog 'Install Device Description' will open where you can browse your system for the respective device description. For the standard devices the file filter is to be set to "*.**devdesc.xml**". But also description files provided by the manufacturer, like for example *.gsd-files for Profibus DP modules, *.eds- and dcf-files for CAN devices, can be selected by setting the respective filter.
As soon as you confirm the selection with OK, the dialog will close and the new device will be added to the devices tree in the 'Device Repository' dialog.

Note: During installation the device description files and all additional files referenced by that description will be copied to an internal location. Altering the original files will have no further effects on the installed devices. In order for this changes to take effect, the devices will have to be reinstalled. It is considered good practice to change the internal version number of a device description after it has been changed.

Attention: The internal device repository must never be altered manually. Do not copy files to or from there! Always use the Device Repository dialog to reinstall, add or remove devices.

Uninstall: This command will remove, that is uninstall the currently selected device. It will be removed from the device repository and not be available for use in the programming system any longer.

The current list of installed devices will be offered when you are going to add a device object via Add Device or Add Object. Device Repository...

12.12 Device Communication

The command category 'Device Communication' provides commands for working in the "Communication Settings" dialog of the generic Device Editor.

12.12.1 Device Communication commands

The commands of category "Device Communication" by default are available in the context menu when working in the Communication Settings dialog of the Device Editor. This dialog serves to define the communication via gateway to the target device. For some of the commands there is also a button available in the dialog.

Available commands:

- Add Device...
- Add Gateway...
- Change Device Name...
- Connect to local Gateway...
- Delete selected Device
- Edit Gateway...
- Scan for Device by Name
- Scan for Device by Address
- Scan Network
- Set Active Path

12.12.2 Add Device

Symbol:

This command (category Device Communication), for which there is also a button in the Communication Settings dialog, opens a dialog titled **Add Device**, where you can manually define a device to be added to the currently selected gateway entry (Regard also the Scan network functionality).

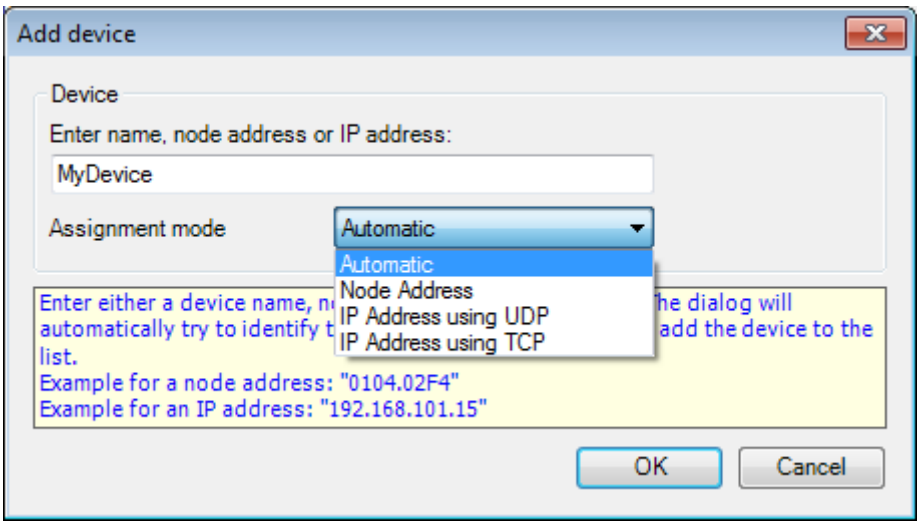


Abb. 101 Add Device" dialog

You can specify the device to be added by its **name** (example:"mynewdevice"), **node address** (example: "0104.02F4") or **IP address** (example "192.168.101.15").

Depending on whether you specify a name or an address, you have to set the appropriate **Assignment mode**, that is how the system should process to add the desired device. In most cases your specification will be unambiguous, so that the "Automatic" mode will work. The other two modes are only needed, when you have to precise the type of your entry, because it is not evidently a name or an address. Example: You want to add a device with hex node address "AFFE". "AFFE" also could be a target name, so, to make the specification clear, set option "Node Address".

- Automatic: The dialog will automatically search the correct mechanism to find and add the device to the list.
- Node Address: If this option is selected, an unique node address (example: 02F4") must be specified. The network then will be checked for the availability of a device with that address and will add it to the list.
- IP address via UDP: If this option is selected, an unique IP address must be specified (example: "192.168.101.15"). The network then will be scanned for the respective device in order to get its node address and to add it to the list. Optionally the UDP port for the device can be appended (example: "192.168.192.15:1749").
- IP address via TCP: If this option is selected, an unique IP address must be specified (example: "192.168.101.15"). The network then will be scanned for the respective device in order to get its node address and to add it to the list. Optionally the TCP port for the device can be appended (example: "192.168.192.15:1749").

If you specify a device, which is not available in the network, the device entry anyway will be added to the list with a default name ("NewDevice") resp. address ("FFFF.FFFF.FFFF"). You can use the "Resolve Address" command to identify the device when it is connected later on. Please note that any network scan will delete this device when it is not available at the time of the scan.

12.12.3 Add Gateway

Symbol:

This command (category Device Communication), for which there is also a button in the Communication Settings dialog, opens the **Gateway** dialog, where you can define a gateway to be added to the current configuration. Enter a symbolic **Name**, the **Driver** type and driver specific parameters (eg. **IP Address** and **Port**) for that gateway. To enter the parameter values perform a double-click on the respective column field to open an edit frame. After having closed the dialog by OK the new gateway entry will be added in the configuration tree in the Communication Settings dialog.

This dialog can be used to edit the gateway entry later (see Edit Gateway...).

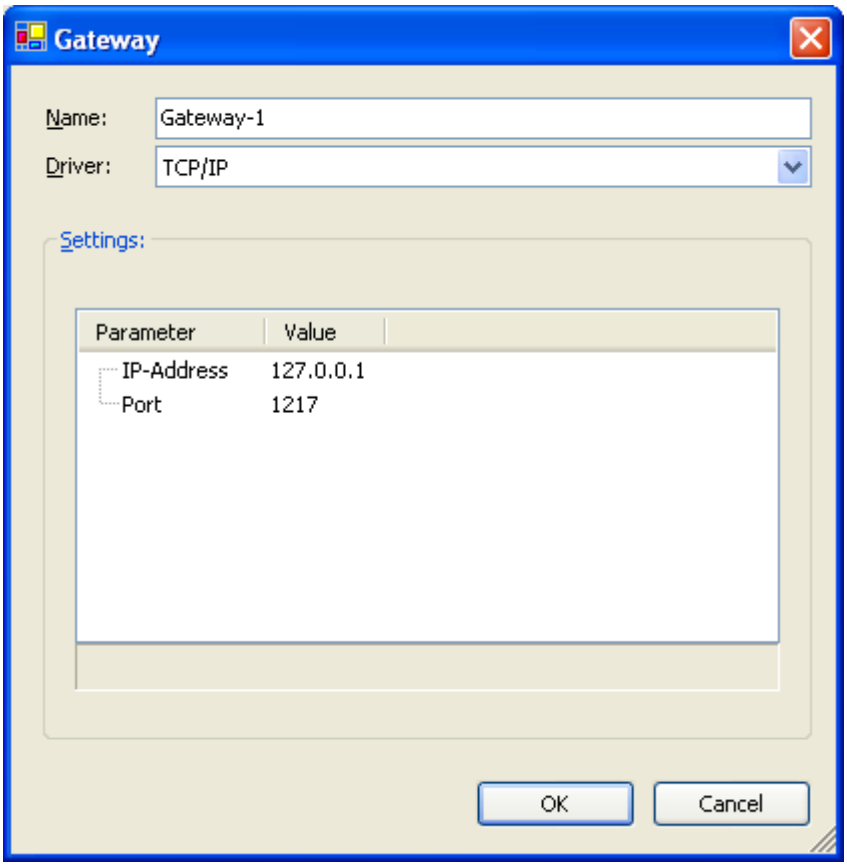


Abb. 102 Gateway dialog

12.12.4 Change Device Name...

This command (category Device Communication) opens the same-named dialog for changing the (**Current**) name of the device as used in the configuration tree in the Communication Settings dialog. The new name can be entered manually in the field behind **New:**., or the standard option "<devicename>" (name of the device as used in the device tree of the project) can be chosen from the selection list.

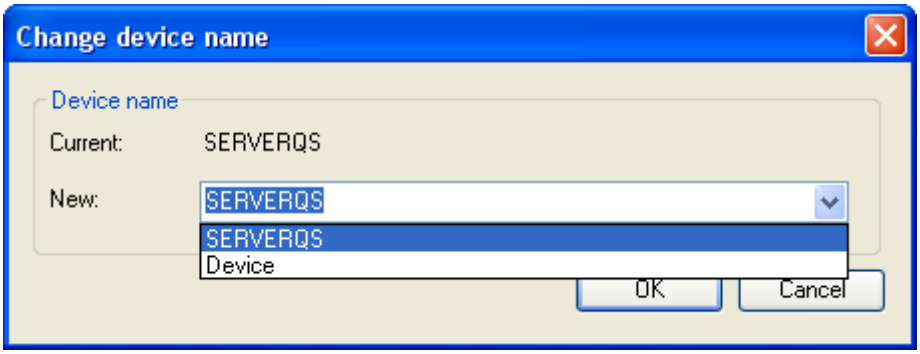


Abb. 103 "Change device name" dialog

12.12.5 Connect to local Gateway...

This command (category Device Communication), for which there is also a button in the Communication Settings dialog of the device editor, will open a dialog containing two tabs that assist with configuring a local gateway and therefore provide an alternative to manual editing the file "Gateway.cfg".

Attention: The proper configuration of the gateway requires detailed knowledge. In case of doubt leave unchanged the default configuration settings!

Routers:

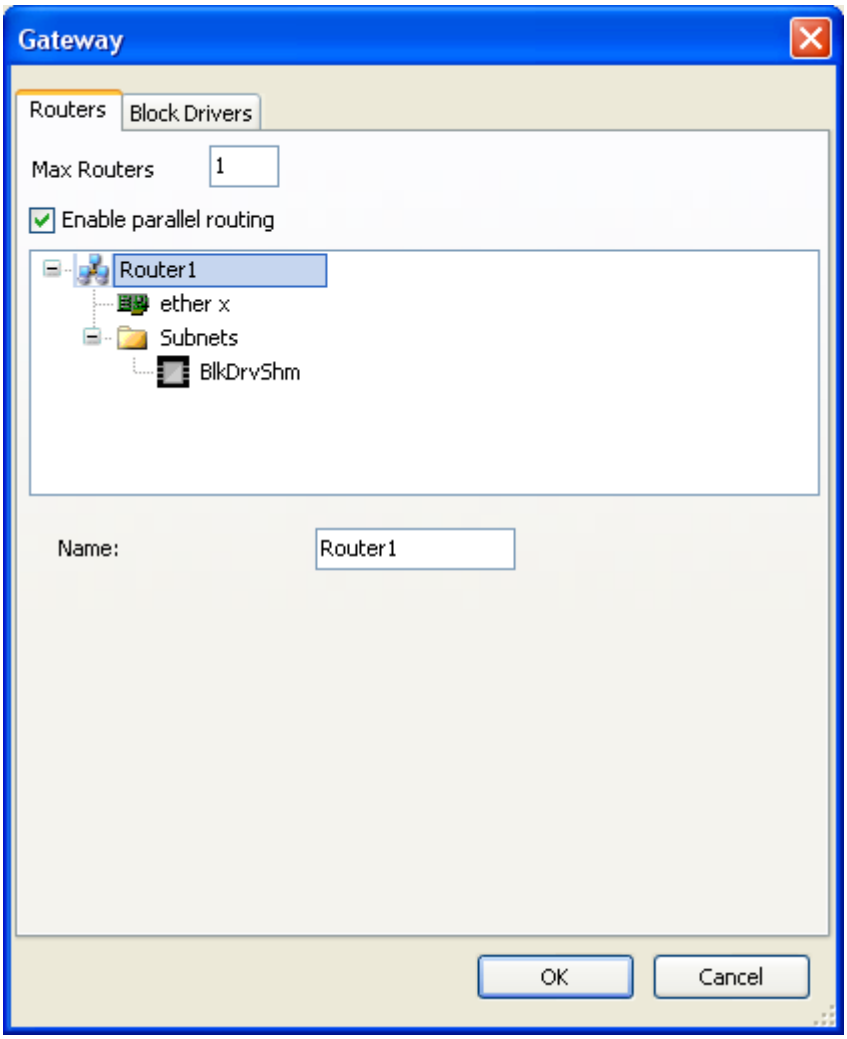
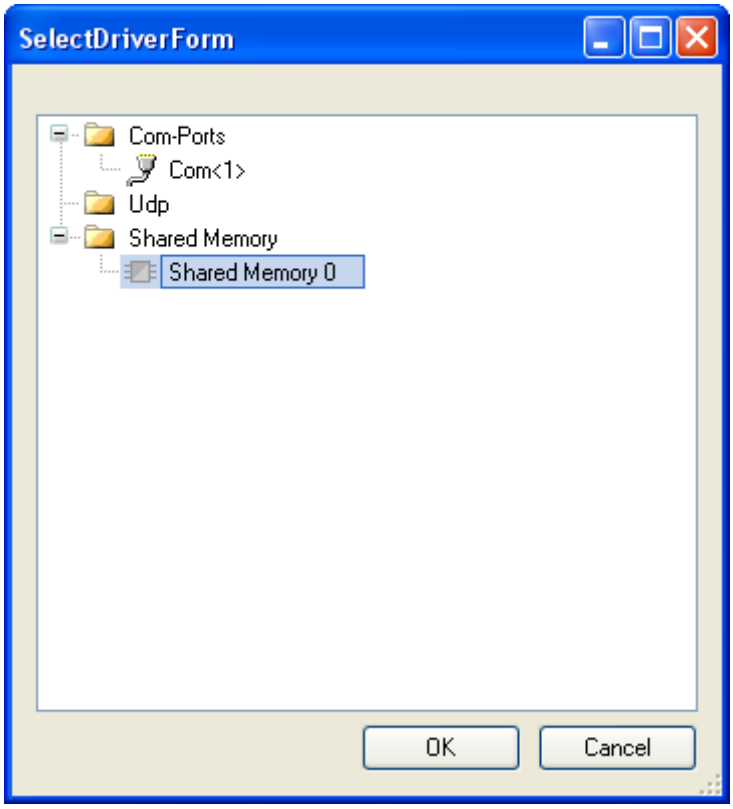


Abb. 104 Dialog Gateway - Routers

In the edit field at right of **Max Routers** you may enter an upper limit for the number of routers. Activating the checkbox **Enable parallel routing** allows the direct communication of two different networks sharing a common node, that, however, is represented twice and bears a different address in each network. While sending a message from one network two the other one via the common node the addressing will be mapped from one to the other configuration when passing this node.

In the window below all routers together with their subnets and interfaces are displayed. Basically each interface is put under control of an own **Router** that is part of the **Main Net**. However, to save resources, it is possible to add (up to 15) further interfaces to one router, not at the same level though, but within **subnets** that are displayed in an own folder below the router.

You can select each entry of the list by a click. The associated context menu will provide an assortment of the following commands:

Add	Serves to add a new subnet to the selected folder. Opens a dialog for selecting an appropriate driver, wherein all block drivers configured so far are listed:  *Abb. 105 Dialog - Select Driver Form* You may select an entry by an click and confirm your selection by OK
Delete	Serves to delete the selected router together with all attachments
New Router	Serves to add a new router
Remove	Serves to delete the selected interface

Set Main Net	Serves to set the interface of the Main Net Executing this command will open a dialog for selecting an appropriate driver, wherein all block drivers configures so far are listed:

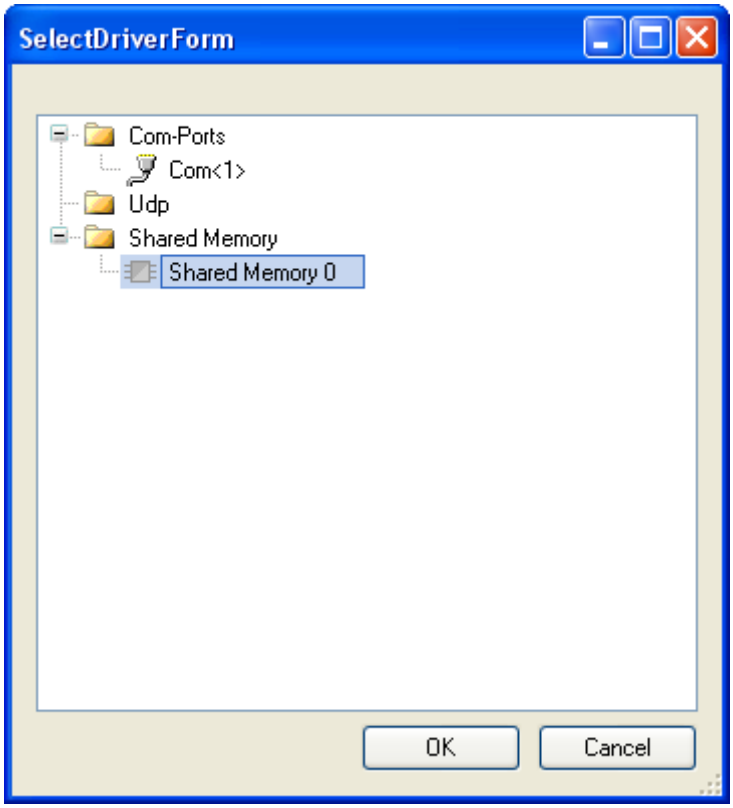


Abb. 106 Dialog - Select Driver Form

You may select an entry by an click and confirm your selection by **OK** or abort the operation with **Cancel**.

The bottom part contains the characteristic settings of the selected entry. For a driver these parameters correspond to the configuration done in tab block drivers. For a router you may edit here its **Name**.

Block drivers:

The upper window displays the folders of Com-Ports, Udp-Ports and Shared Memory, where all interfaces being part of the gateway are arranged according to their type. You may select an entry by a click . The associated context menu will provide one of the following commands:

	Serves to delete the selected block driver
	Serves to add a Com-Port to the Com-Ports folder
	Serves to add a Udp block driver to the Udp folder
	Serves to add shared memory to the Shared Memory folder

Attention: It is strongly recommended not to modify the contain of the folder Shared Memory!

Attention: If you add a block driver for Com-Port for the first time in the current configuration, the configuration file **gateway.cfg** must be extended manually. For this purpose open the file, which per default is available in directory "GatewayPLC" in the »PLC Designer« installation directory (e.g. C:\Programs\3S »PLC Designer«\GatewayPLC), in a text editor and in section [ComponentManager] add the following entry:
Component.xy=CmpBlkDrvSimpleCom
whereby xy must match the continuous component numbering.
If this entry is missing, the block driver will not be loaded when the gateway is started!

In the bottom part the configuration data of the selected entry will be displayed.

In case of a Com-Port you may edit its **Name**, its subindex within the gateway (**Com Port**) and its **Baudrate**. Activation of the checkbox **Enable Auto Adressing** will resolve ambiguous addresses automatically. If unique and not specified otherwise by the field **Local Address**, the default for the local address equals the value of Com Port.

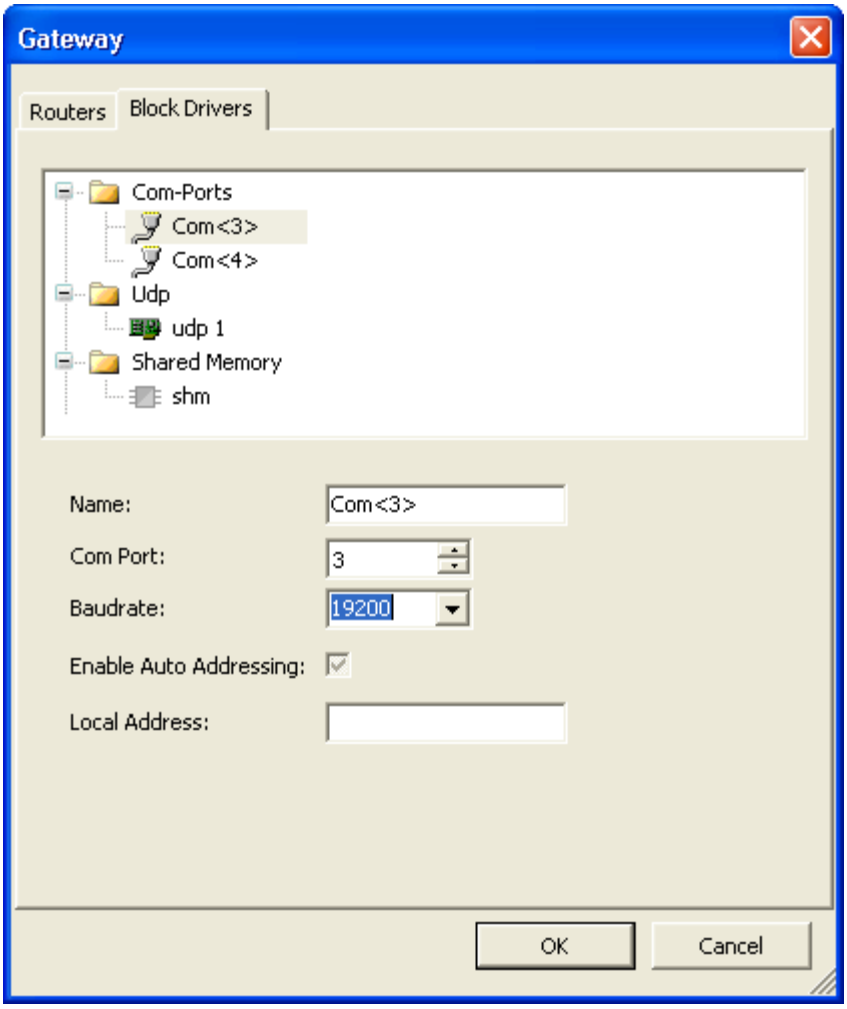


Abb. 107 Dialog Gateway - Block drivers (Com Port)

In case of an Udp block driver you may edit its **Name** and specify the **IP Address**, if you want to change the default router. Further you may indicate the **Network Mask,** the **Port Index** (for selecting the port number of the »PLC Designer« gateway; the Port Index ranges from 0 to 3, where the index 0 corresponds to the port number 1740 and the Index 3 to 1743). Entering a **Ppp Remote Address** establishes a logical point to point connection between the Udp interface and the node indicated by this address. The consequence is that the Udp interface will communicate with this node exclusively and no more broadcasts will be sent within this network.

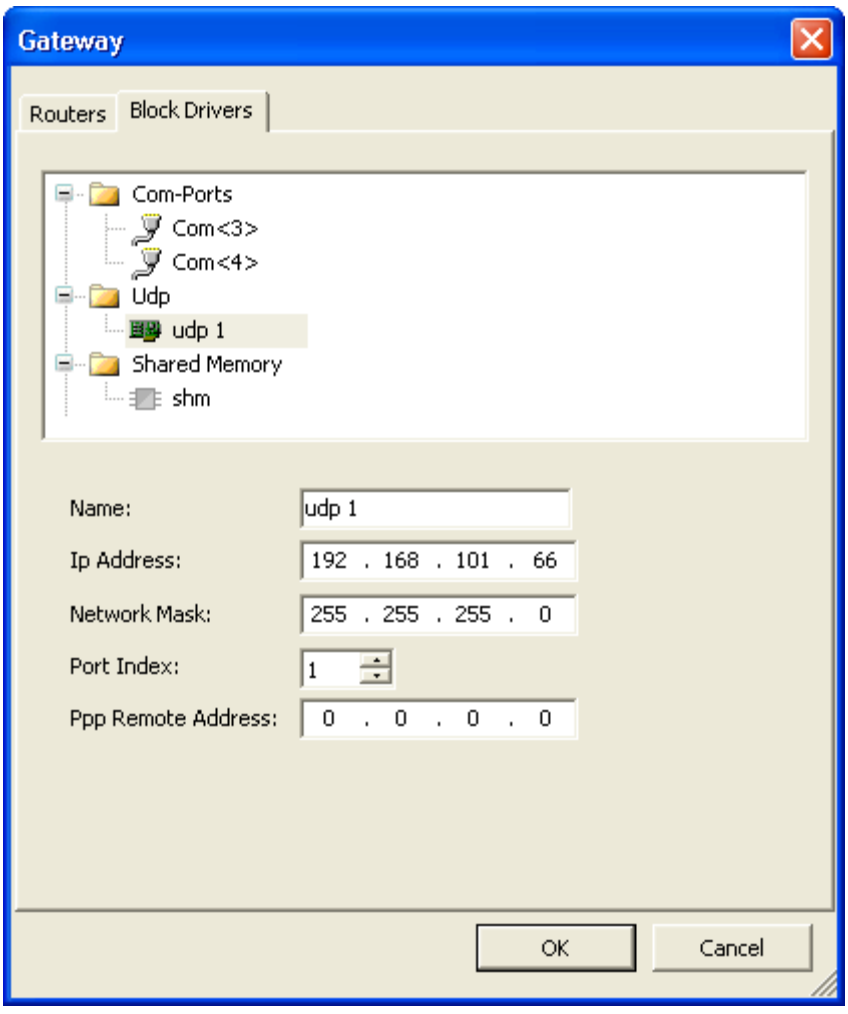


Abb. 108 Dialog Gateway - Block drivers (Udp)

In case of Shared Memory you may edit its **Name**. If you want to fix the local address of the shared memory, you may enter this address in the edit field **Forced Address**.

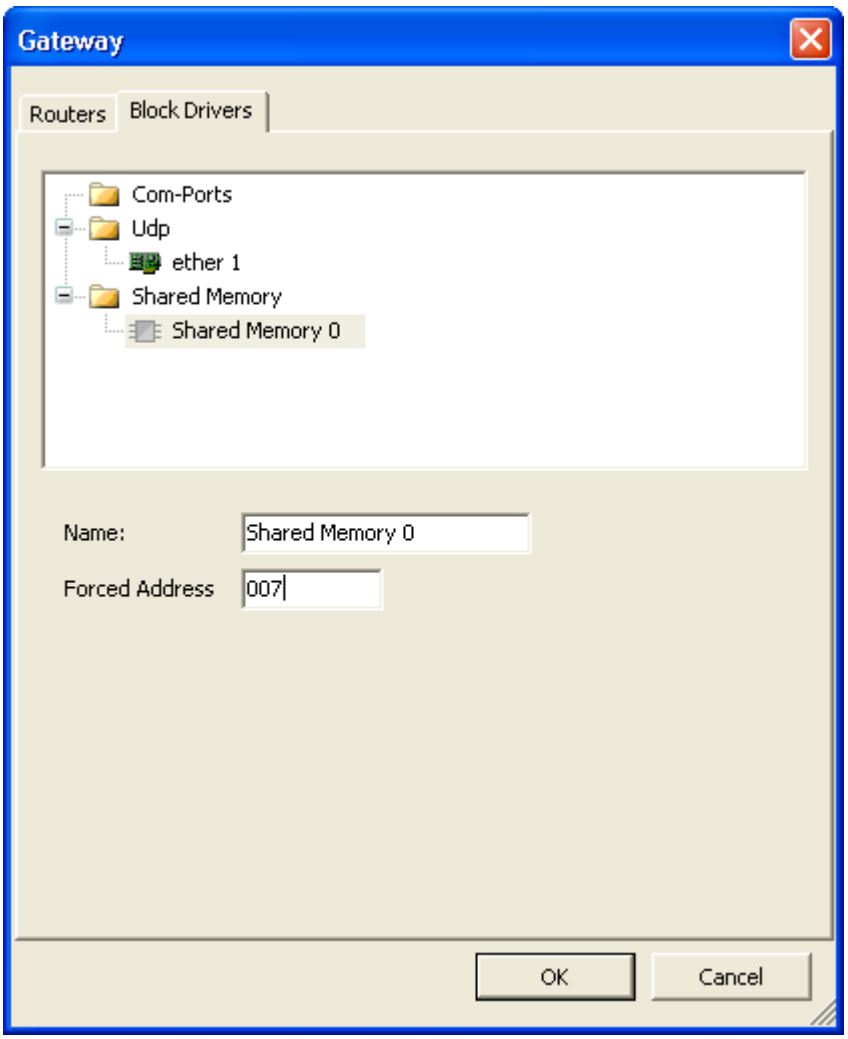


Abb. 109 Dialog Gateway - Block drivers (Shared Memory)

Attention: After having finished the configuration in the Gateway dialog (closed with OK) and a possible modification of the gateway.cfg file the **gateway must be stopped and restarted**. Otherwise the changes will not be effective!

12.12.6 Delete selected Device

This command (category Device Communication) removes the currently selected device from the list in the Communication Settings dialog.

12.12.7 Edit Gateway...

This command (category Device Communication) opens the Gateway dialog, which in the Communication Settings is used for adding a gateway resp. for editing the settings of the currently selected gateway.

12.12.8 Scan for Device by Name

Symbol:

This command (category "Device Communication") serves to scan the network for devices of the same name as specified in the "Communication Settings" devices list below the corresponding gateway entry (case sensitive search!). If any are found, each of them will be added to the list with its specific node address, i.e. the entry will be displayed with name + node address.

Example

USER1 [057B]

12.12.9 Scan for Device by Address

Symbol:

This Command (category "Device Communication") serves to scan the network for devices with the same node address as specified in the "Communication Settings" devices list below the corresponding gateway entry. If any are found, each of them will be added to the list with its name, i. e. the entry will be displayed with name + node address.

Example

USER1 [057B]

12.12.10 Scan Network

Symbol:

This command (category Device Communication) for which there is also a button in the Communication Settings dialog, starts a search for available devices in your local network and the configuration tree will be updated accordingly. The command will also be executed on a doubleclick on the gateway entry.

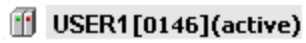
The found devices will be listed below that gateway entry, in the tree of which an entry has currently been selected when calling the command.

12.12.11 Set Active Path

Symbol:

This command (category Device Communication) for which there is also a button in the Communication Settings dialog, sets the communication channel, currently selected in the tree below the gateway, as the active path to the controller. This means that all actions concerning communication will exactly refer to this channel. The command will also be executed on a doubleclick on the channel entry.

The currently active path will be displayed bold in the list, and "(active)" will be appended:



If a device has been set as the active path, but was not reachable during the last scan network, it will be drawn in italics.

Note: The iInformation on which device was recently set as "active path" will be stored on the local system and can be automatically restored at a Login with not yet configured communication settings.

12.13 FBD/LD/IL

The command category 'FBD/LD/IL' provides commands for working in the combined editor for FBD, LD and IL.

12.13.1 FBD/LD/IL Editor commands

The 'Network List Editor' plug-in provides for command category 'FBD/LD/IL' the following commands for the FBD/LD/IL-Editor, which is a collective editor for the languages FBD (Function Block Diagram), LD (Ladder Diagram) and IL (Instruction List).

By default these commands are part of the 'FBD/LD/IL' menu. If necessary, open the Customize dialog, category 'Menu', to view or modify the current menu configuration.

Note: For inserting elements in the FBD and LD editor view regard also the possibility to drag them directly from the toolbox or from another position within the editor.

12.13.2 Insert Network

Symbol:

Shortcut: <Ctrl> + <I>

This command (category 'FBD / LD / IL') is used to insert a network in the FBD, LD or IL editor.

If the cursor currently is placed within an existing network, the new network will be inserted immediately **above** that network. If the cursor currently is placed in the editor window but not in a network, the new network will be added at the end of the current network list. The network numbering will be updated automatically.

Notice that in contrast to V2.3 the network element is also used in the IL editor.

Note: The display of the elements in a LD/FBD/IL network is defined in the 'FBD/LD and IL editor Options'.

12.13.3 Insert Network below

Symbol:

Shortcut: <Ctrl> + <T>

This command (category 'FBD / LD / IL') is used to insert a network in the FBD, LD or IL editor.

If the cursor currently is placed within an existing network, the new network will be inserted immediately **below** that network. If the cursor currently is placed in the editor window but not in a network, the new network will be added at the end of the current network list. The network numbering will be updated automatically.

Notice that in contrast to V2.3 the network element is also used in the IL editor.

Note: The display of the elements in a LD/FBD/IL network is defined in the 'FBD/LD and IL editor Options'.

12.13.4 Toggle network comment state

Symbol:

Shortcut: <Ctrl> + <O>

This command (category 'FBD / LD / IL') can be used in FBD, LD or IL editor to comment out a network resp. to set it back from comment to normal state. The command will effect the network in which the cursor is currently positioned.

A commented-out network will be displayed according to the options set for comments and will not be noticed in program processing.

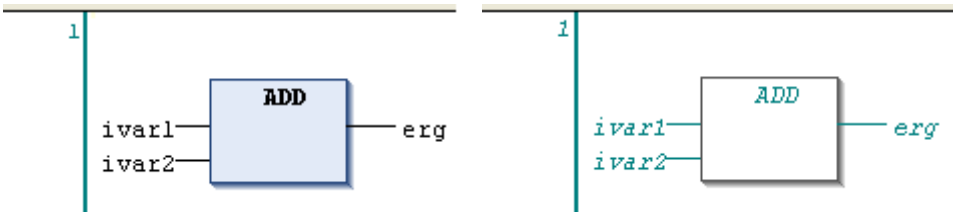


Abb. 110 Example: network, normal state and comment state

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.5 Insert Assignment

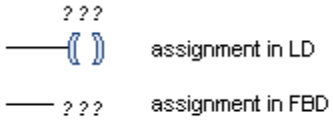
Symbol:

Shortcut: <Ctrl> + <A>

This command (category 'FBD / LD / IL') is used to place an assignment in a LD, FBD or IL editor.

Depending on the selected position insertion takes place directly in front of a selected input (Cursor Position 2), directly after a selected output (Cursor Position 4) or - if a whole network or sub-network is selected - at the end of the network or sub-network (Cursor Position 6 resp. 11).

In FBD the assignment is inserted as a line followed by "???", in LD it is represented by a coil and "???":



To define the assignment select the "???" and replace it by the name of the variable that is to be assigned. The Input Assistant can be used for this purpose.

In IL an assignment is programmed via the "LD" and "ST" operators. Example:

```
PROGRAM ILInsertAssignment
VAR
    iVar: INT;
    bResult: BOOL;
END_VAR
```

```
LD      iVar
ADD     1,
        4
GT      12
ST      bResult
```

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.6 Insert Box

Symbol:

Shortcut: <Ctrl> + <B>

This command (category 'FBD / LD / IL') is used to insert a box element into a network for the purpose of calling an operator, a program, a function block, a function or an interface. In the IL editor the corresponding instructions will be inserted.

As soon as you choose the command, the Input Assistant dialog will open, providing the appropriate categories of POUs. Select one and confirm with OK to insert the box at the currently selected position in a network resp. to create the corresponding IL instructions.

Alternatively you can choose command Insert Empty Box so that you can enter the desired box type directly. A very comfortable way to add a box is to drag it directly from the ToolBox or from another position within the editor. Concerning drag&drop of elements see: Working in the FBD and LD editor view.

See in the following the editor-specific characteristics for inserting a box element:

FBD or **LD**:

- Boxes of type **program** or **function block** always are inserted in line, that is the processing line will be connected to the uppermost input and output of the inserted POU.

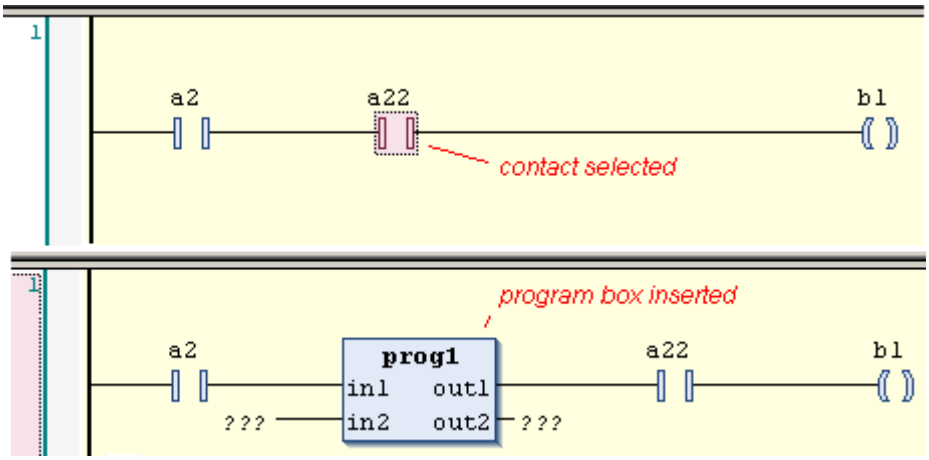


Abb. 111 Example: Insert function block in LD

- The text within the box shows the **box type** (for example F_TRIG) and is editable. By replacing this text by the type name of another valid module you can **replace** the box by another one. An existing box also can be replaced by inserting another one at the same position. Notice that **if already inputs have been defined** for the previously used box, these will be kept, except the new box has a lower maximum number of inputs. In this case the last inputs will be deleted accordingly.
- If provided with the respective module and if option Show box icon is activated, an icon will be displayed within the box.

- Within parallel connections in a LD network no insert positions will be offered when dragging a box element from the ToolBox. Reason: A POU call (box) needs a direct connection to the power rail.

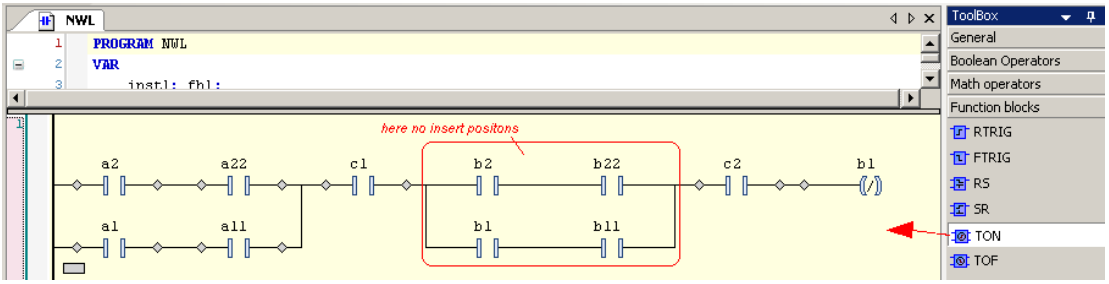


Abb. 112 Example: Insert positions for box element in LD network

- In the LD editor boxes for the call of certain **operators** automatically are inserted with **EN and ENO** resp. only EN in- and outputs. EN and ENO connections get those which have a non-boolean output (for example ADD, SEL, BOOL_TO_INT), only an EN input get those which have a boolean output (for example EQ, GE, GT). At a box with an ENO output it is not possible to insert a further box at the other outputs of this box.
- **VAR_IN_OUT** parameters of an inserted POU box are marked with a bidirectional arrow ↔.

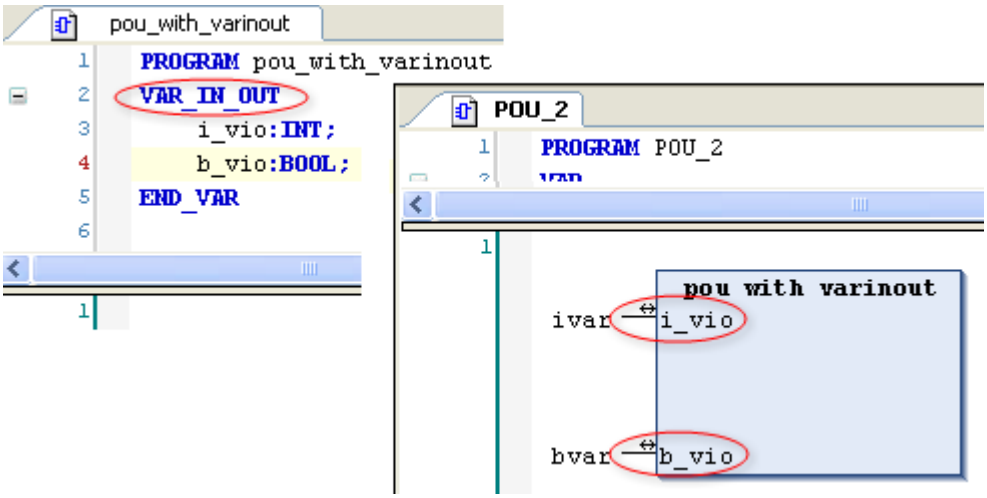


Abb. 113 Display of VAR_IN_OUT parameters

- **Function block** boxes have an editable field above the box where you have to enter the name of the instance variable. If a box representing the instance of a function block instance gets replaced by inserting another function block type, the instance has to be redefined also.
- In functions and function blocks, the **formal names of the in- and outputs** are displayed. The main output of the function (return value) however is displayed without name.
- In case the **box interface** has been changed, you can update the box parameters (for example modified number of outputs) by command Update parameters. **The update will not be done automatically** as it is in »PLC Designer« V2.3!

- I**nsert positions**: The most recently inserted POU will be inserted at the currently selected position:
- If an input is currently selected (cursor position 2), the box will be inserted before this input. Its first input and - if applicable- its first output will be linked within the existing branch.
- If an output is currently selected (cursor position 4), the box will be inserted after this output. Its first input and - if applicable- its first output will be linked within the existing branch.
- If a box is currently selected (cursor position 3), the old element will be replaced by the new POU. As far as possible, the connections will remain as they were before the replacement. If the old element had more inputs than the new one, then the unattachable branches will be deleted. The same holds true for the outputs.
- If a jump or a return is selected (cursor position 3), the box will be inserted before this jump or return. Its first input and - if applicable- its first output will be linked within the existing branch.
- If a complete network or sub-network is selected (cursor position 6 resp. 11), then the box will be inserted following the last element of the network or sub-network and be linked with its first input.
- All box inputs that could not be linked will receive the text "???". This must be selected and replaced by the name of the desired constant or variable.
- If there is already a branch to the right of an inserted box, this branch will be assigned to the first output of the box. Otherwise the outputs remain unassigned.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

IL:

In IL also a "box" can be inserted at any desired line. If the Input Assistant is used with option 'Insert with arguments', the chosen POU will be displayed in form of a CAL instruction and the respective input and output parameters of the chosen box element.

Example:

```
1   CAL          TONinst(
            IN:= ???,
            PT:= ???,
            ET=> ???)
```

Abb. 114 Box inserted in IL:

Box element "TON" has been inserted; instance TONinst has been defined locally; input parameters: IN, PT; output parameter: ET.

12.13.7 Insert Empty Box

Symbol:

Shortcut: <Ctrl> + <Shift> + <B>

This command (category 'FBD / LD / IL') is used to insert an empty box element into a network.

See Insert Box for detailed information on inserting boxes in the FBD/LD/IL editor. In contrast to this command, when inserting an empty box not automatically the input assistant will open, but the instance field above the box. Then decide, which type of box you need:

- If the box should represent a **function block**, then enter the desired instance variable name and close the input with <Return>. You might use the input assistant ... for entering the name of an already existing instance. After inserting an already declared instance variable the function block immediately will be displayed accordingly. If you entered a not yet known instance name, you also must specify the name of the desired function block. For this purpose after closing the instance name input, the input focus automatically changes to the box type edit field within the box. In this case the input assistant will only offer function blocks.
- If the box should represent an **operator**, **program**, **function** or **interface**, then when the cursor is placed in the instance field, just press the down arrow key. The input focus will change to the box type field (within the box), where you can enter the respective operator, program, function or interface name, directly or via the input assistance. After having terminated this input, the box will be displayed accordingly.

In the IL editor the corresponding instructions will be inserted.

12.13.8 Insert Jump

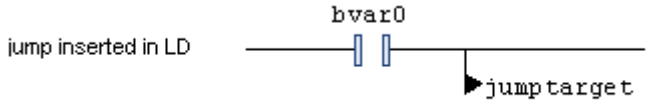
Symbol:

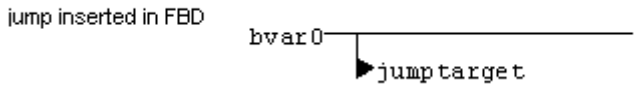
Shortcut: <Ctrl> + <L>

This command (category 'FBD / LD / IL') inserts a jump. The target of a jump is another network, that is the label of that network.

In **FBD** or **LD**, depending on the selected cursor position, insertion takes place directly in front of a selected input (Cursor Position 2), directly after a selected output (Cursor Position 4) or - if a whole network or sub-network is selected - at the end of the network or sub-network (Cursor Position 6 resp. 11).

For an inserted jump, a selection can be made accompanying the entered text "???", and the jump can be replaced by the name of the label to which it is to be assigned.





In **IL** a jump is programmed via the JMP operator.

Example:

```
1   PROGRAM IL
2   VAR
3       bVar: BOOL;
4       bVar2: BOOL;
5   END_VAR
```

```
1   LD          bVar
    JUMPC       jumptarget
2    jumptarget:
    LD          bVar2
```

If a JMP operator, that has been inserted in **IL** without preceding LD is converted to **LD**, a dummy operator "???" will be inserted.

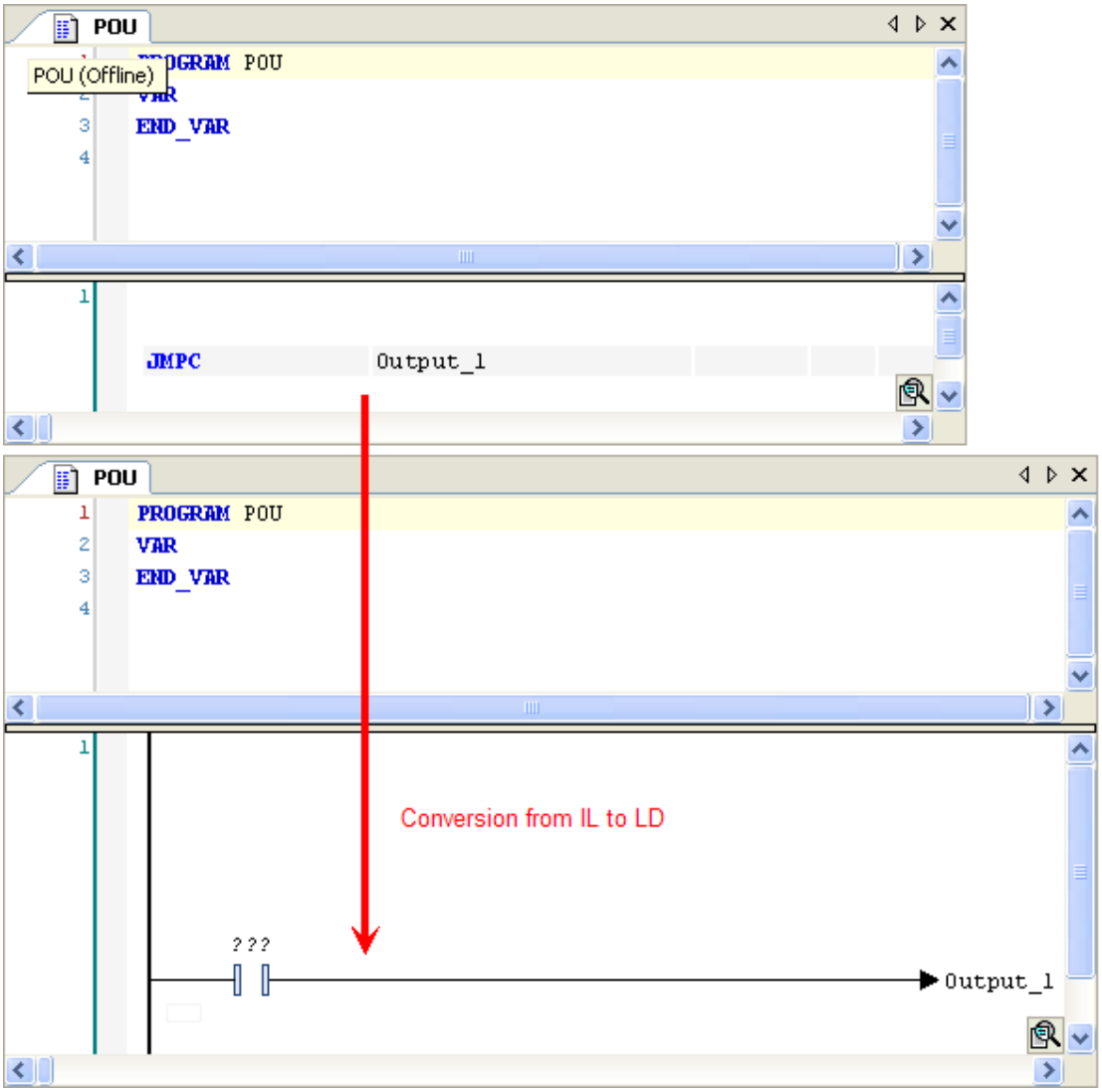


Abb. 115 Example: Conversion of JMP operator

Note: Concerning the view options for the components of a FBD/LD/IL network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.9 Insert label

Symbol:

This command (category 'FBD / LD / IL') inserts a label field into the currently selected network. The default text string "Label:" can be modified according to your demands:

- You may enter the name of a label providing a target for a jump instruction.
- You may enter a pragma instruction. See "Pragma instructions" for a detailed description.

If you want to make use of both of the alternatives, enter the pragma first followed by the name of the label.

12.13.10 Insert Return

Symbol:

This command (category 'FBD / LD / IL') inserts a RETURN instruction.

FBD or **LD**:

Depending on the selected position insertion takes place directly in front of a selected input (cursor position 2),directly after a selected output (cursor position 4), directly before a selected line cross (cursor position 5) or at the end of a network or sub-network (cursor position 6 resp. 11).

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

IL:

In IL a Return instruction will be inserted.

12.13.11 Insert Input

Symbol:

Shortcut: <Ctrl> + <Q>

This command (category 'FBD / LD / IL') inserts an additional input at an extensible box (AND, OR, ADD, MUL, SEL) in FBD or LD editor. It is not available in the IL editor.

The maximum number of inputs depends on the box type (for example ADD can have 2 or more inputs.)

In order to extend a box by an input at a certain position, select the input above which you wish to insert an additional input (cursor position 1).

In order to extend a box by an input at the lowest (last) position, select the box body (cursor position 3).

The new input primarily is allocated with the text "???". Replace this text by the name of a desired constant or variable. The Input Assistant might be used for this purpose.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.12 Insert Coil

Symbol:

Shortcut: <Ctrl> + <A>

This command (category 'FBD / LD / IL') is used to insert a coil in parallel to the previous coils.

If the marked position is a connection between the contacts and the coils, then the new coil will be inserted as the last. If the marked position is a coil, then the new coil will be inserted directly above it.

The coil is given the text "???" as a default setting. You can click on this text and change it to the desired variable or you can use the Input Assistant instead.

For the possibility of entering addresses, of linebreaks for variable names and of comments per coil please see the description of the Ladder Options dialog.

12.13.13 Insert Set Coil

Symbol:

This command (category 'FBD / LD / IL') is used to insert a set coil. Alternatively the combination of the commands 'Insert' 'Coil' and' Extras' 'Set/Reset' can be used to insert a set coil.

12.13.14 Insert Reset Coil

Symbol:

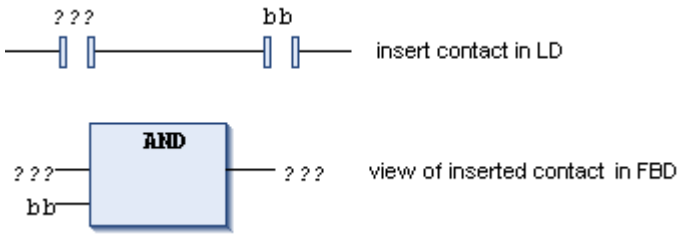
This command (category 'FBD / LD / IL') is used to insert a reset coil. Alternatively a combination of the commands 'Insert' 'Coil' and' Extras' 'Set/Reset' can be used to insert a reset coil.

12.13.15 Insert Contact

Symbol:

Shortcut: <Ctrl> + <K>

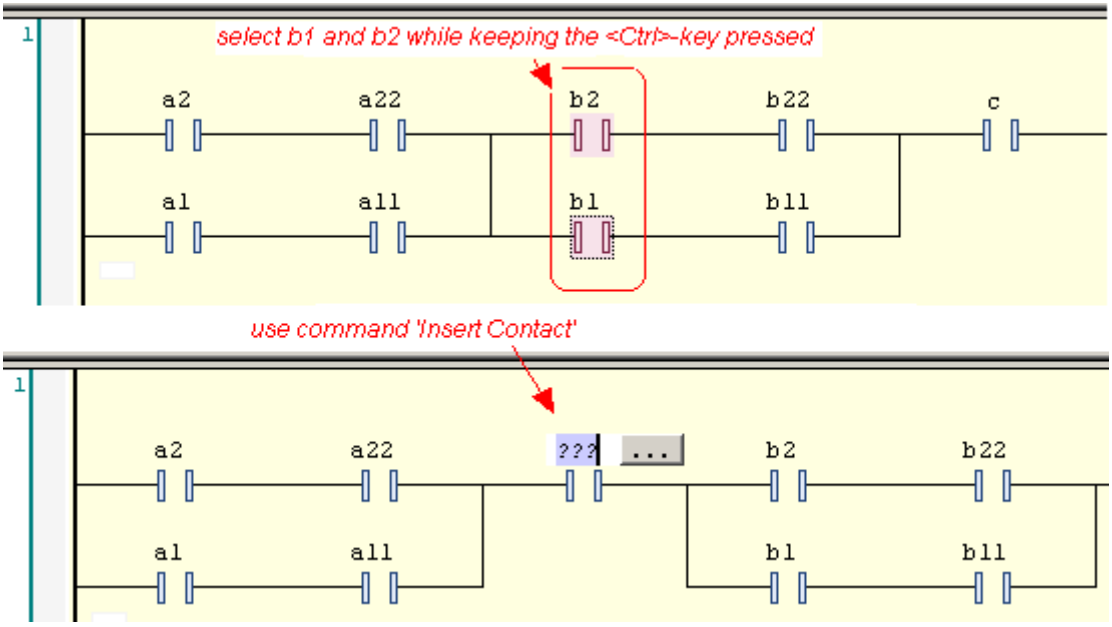
This command (category 'FBD / LD / IL') inserts a contact in a LD network. It is not available in the FBD and IL editor but will be converted appropriately when switching views.



The new contact will be inserted in line **left to** the currently selected contact or box. If the cursor position is within an existing parallel connection, the new contact element also will be inserted within.

Note: Alternatively a contact element can be inserted by dragging from the ToolBox or from another position within the editor. If however it should not be inserted within, but before, behind or between existing parallel connections, this only will be possible by using the menu command. For this purpose first in each of the branches of the parallel connection select one of the contact elements (multiselection while keeping pressed the <Ctrl>-key) and then use the command.

Example



Notice also the commands Insert Contact right resp. Insert Contact Parallel above, Insert Contact Parallel below.

A new contact element is preset with the text "???". You can select this text by a mouse-click and replace it by the name or address (depending on the current settings in the FBD/LD and IL Options dialog) of the desired variable or the desired constant. The Input Assistant can be used for this purpose.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.16 Insert Negated Contact

Symbol:

This command (category 'FBD / LD / IL') is used to insert a negated contact. Alternatively a combination of the commands 'Insert Contact' and 'Negation' can be used to insert a negated contact.

12.13.17 Insert Contact right

Symbol:

Shortcut: <Ctrl> + <D>

This command (category 'FBD / LD / IL') inserts a contact in a LD network. The same is valid as for command Insert Contact, except that the new element will not be inserted left but right to the current cursor position in line.

12.13.18 Insert Contact Parallel below

Symbol:

Shortcut: <Ctrl> + <R>

This command (category 'FBD / LD / IL') inserts a contact parallel to the marked position in the network. The same is valid as for command Insert Contact Parallel (above), except that the new contact will be inserted **below** the selected position.

12.13.19 Insert Negated Contact Parallel below

Symbol:

This command (category 'FBD / LD / IL') is used to insert a negated parallel contact. Alternatively a combination of the commands 'Insert Contact Parallel (below)' and 'Negation' can be used to insert a negated parallel contact.

12.13.20 Insert Contact Parallel above

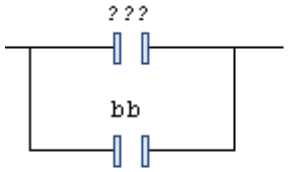
Symbol:

Shortcut: <Ctrl> + <P>

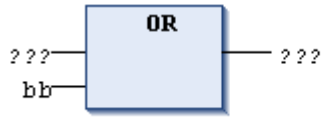
This command (category 'FBD / LD / IL') inserts a parallel contact above the currently marked position in the network (parallel connection). The command is not available in the FBD and IL editor but will be converted appropriately when switching views.

Notice that also multiple elements might be selected in order to get the new contact inserted parallel to those.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. The Input Assistant can be used for this purpose.



insert parallel contact (selected position was "bb" contact) in LD



view of inserted "parallel contact" in FBD

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.21 Paste contacts: Paste below

Shortcut: <Ctrl> + <F>

This command (category 'FBD / LD / IL') is only available in LD editor. It pastes the elements resp. the section of the network which has been put to the clipboard before by a "copy" or "cut" command, **below** the currently selected contact element in the network. This corresponds to the common "paste" command.

See Working in the FBD and LD editor view for a description on selecting in networks.

12.13.22 Paste contacts: Paste right after

Shortcut: <Ctrl> + <G>

This command (category 'FBD / LD / IL') is only available in LD editor. It pastes the elements resp. the section of the network which has been put to the clipboard before by a "copy" or "cut" command, **right to** the currently selected contact element in the network.

See Working in the FBD and LD editor view for a description on selecting in networks.

12.13.23 Paste contacts: Paste above

Shortcut: <Ctrl> + <H>

This command (category 'FBD / LD / IL') is only available in LD editor. It pastes the elements resp. the section of the network which has been put to the clipboard before by a "copy" or "cut" command, **above** the currently selected contact element in the network.

See Working in the FBD and LD editor view for a description on selecting in networks.

12.13.24 Insert IL line below

Symbol:

This command (category 'FBD / LD / IL') is only available in the Instruction List editor. It is used to insert a further instruction line below the currently selected one.

12.13.25 Delete IL line

Symbol:

Shortcut: <Ctrl> + <Del>

This command (category 'FBD / LD / IL') is only available in the Instruction List editor. It is used to delete the line where currently the cursor is positioned.

12.13.26 Negation

Symbol: (FBD), (LD)

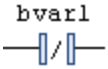
Shortcut: <Ctrl> + <N>

This command (category 'FBD / LD / IL') is used in FBD or LD to toggle the negation of an input, an output, a jump or a RETURN instruction. It is not available in the IL editor, where the corresponding modifiers have to be used appropriately.

Attention: If view is switched from FBD or LD to IL view and back, the negations of some constructs might be set back because an un-ambiguous conversion is not possible.

At boxes, jumps or returns the symbol for the negation is a small circle at the respective input or output connection.

A negated contact in LD is indicated by a slash in the contact symbol:



To negate a contact or coil select the element (cursor positions 8 resp.9) and use the command. Notice that the ToolBox in category 'Ladder elements' provides negated contact elements for inserting by drag&drop.

To negate the an input or output put the cursor according to cursor positions 2 or 4.

To negate a jump or RETURN instruction select the last preceding output (cursor position 4).

To cancel the negation of an element use the same cursor positions and also perform the Negate command.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.27 Edge Detection

Symbol: (FBD), (LD)

Shortcut: <Ctrl> + <E>

This command (category 'FBD / LD / IL') is used in FBD or LD to insert an edge detection element at a boolean input. This corresponds to inserting a R_TRIG function block for detecting a rising edge (FALSE -> TRUE) respectively a F_TRIG function block for detecting a falling edge (TRUE -> FALSE).

When the command is performed repeatedly at the same insert position, the inserted element will toggle between rising edge detection ( ), falling edge detection ( ) and none.

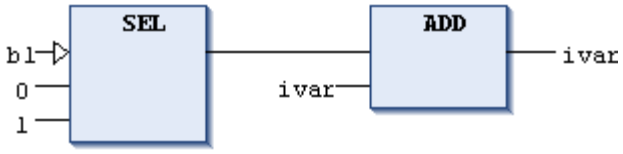


Abb. 116 Example, Edge Detection at SEL operator

In this example a Edge Detection element has been inserted when the first input (b1) of the SEL box was selected. The SEL operator will have output "1" each time a rising edge is detected at its input.

The command is not available in IL editor. A network containing an edge detection will be kept unmodified after conversion from FBD/LD to IL.

Note: Concerning the display options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.28 Set/Reset

Symbol:

Shortcut: <Ctrl> + <M>

This command (category 'FBD / LD / IL') is used in FBD or LD to define boolean outputs resp. coils as "Set" or "Reset" outputs resp. coils. It is not available in the IL editor, where the corresponding modifiers have to be used appropriately.

With multiple executions of the command, the output will alternate between set, reset, and normal output.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.29 Set output connection

Symbol:

Shortcut: <Ctrl> + <W>

This command (category 'FBD / LD / IL') can be used in FBD or LD with boxes which have multiple outputs to determine that output which should be connected to the network processing line.

This is an **extension compared to »PLC Designer« V2.3** where only the uppermost output can be connected to the processing line.

Notice the shift of the output assignments in case of changing the output connection.

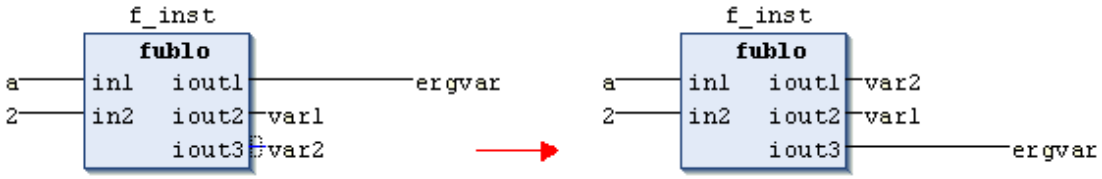


Abb. 117 Example: set output connection at o3

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.30 Insert Branch

Symbol:

Shortcut: <Ctrl> + <V>

This command (category 'FBD / LD / IL') branches the current execution line within a network in FBD/LD.

The current line will be split into two "subnetworks": An additional line will be drawn "below" the existing one. In online mode the two subnetworks after a branching point will be executed one after the other from up to down.

If you drag the "branch" element from the toolbox or another position within the network, you will get indicated all possible insert positions by grey position markers.

A branch can be inserted at the input connectors of boxes which are not positioned in a subnetwork, at the output connectors (cursor position 4) of a box if that is not connected (also indirectly) to the input of another box within a subnetwork, at the connector between contacts and coils (cursor position 10), or at a contact (cursor position 8). A branch cannot be inserted inside of "OR"ed contacts groups and inside of multiple assignment groups.

Each subnetwork gets an own "marker", an upstanding rectangle symbol, which serves as a possible cursor position (11) for selecting the subnetwork.

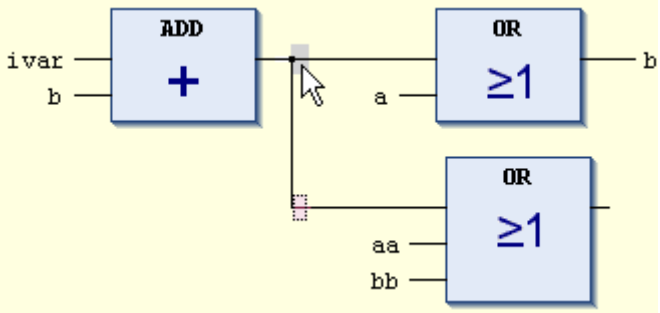


Abb. 118 Subnetwork markers in FBD network

Each subnetwork can get another branching, thus multiple branches and thus a widely ramified construction of "subnetworks" is possible within the main network. For adding branches to branches see the commands: Insert Branch above and Insert Branch below.

For general information on branches see Branches in FBD/LD/IL.

The command is not available in IL editor. Networks with branch elements cannot be converted to IL.

Note: Concerning the display options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.31 Insert Branch above

Symbol:

This command (category 'FBD / LD / IL') is available for inserting another branch within an existing one above the current cursor position.

12.13.32 Insert Branch below

Symbol:

This command (category 'FBD / LD / IL') is available for inserting another branch within an existing one below the current cursor position.

12.13.33 Update parameters

Shortcut: <Ctrl> + <U>

This command (category 'FBD / LD / IL') can be used in FBD, LD or IL editor to update the parameters (inputs, outputs) of a box, which is already inserted in a network, after having changed its interface for example by adding an output.

The already defined connections of inputs and outputs remain unchanged, resp. if an input or output gets added, this will get the "???" and can be assigned.

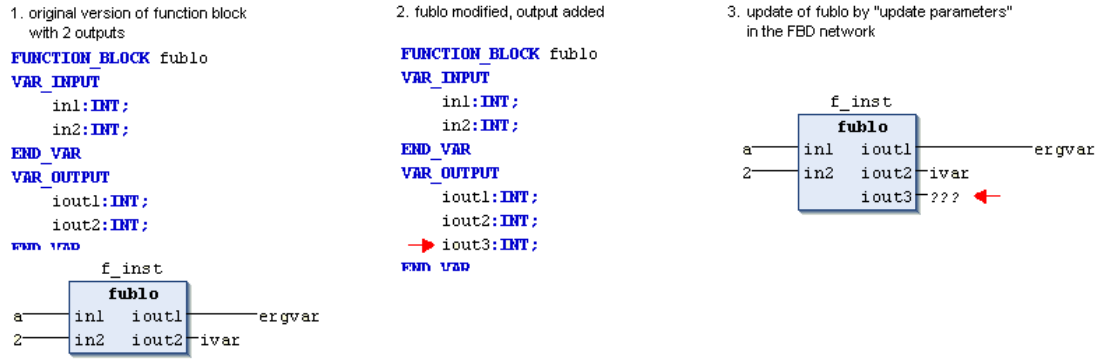
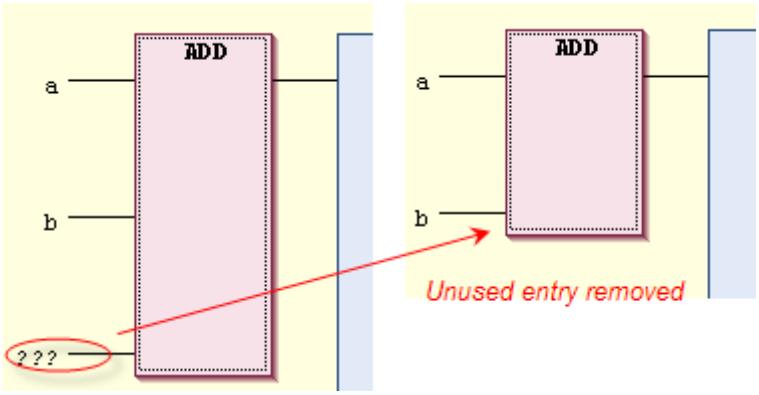


Abb. 119 Example, update parameters

12.13.34 Remove Unused FB Call Parameters

Symbol:

This command (category 'FBD / LD / IL') is available only for implementation in FBD. Its execution will remove all unassigned entries or exits of the FB box focused, that is, all entries or exits whose assignments are empty or marked by '???'. However, the minimum number of necessary in- or outputs of the box will be maintained.



12.13.35 View as function block diagram

Shortcut: <Ctrl> + <1>

This command (category 'FBD / LD / IL') is available if you currently are working in the Ladder Diagram (LD) or Instruction List (IL) view of the POU. It can be used in offline and online (online not available with the current version) mode.

The LD networks respectively the instruction list will be converted appropriately to FBD networks. Notice anyway, that there are some special elements, which **cannot get converted** and thus will only be available in the appropriate editor view. Further on some constructs might not be converted un-ambiguously.

To switch back to the LD view use command View as ladder logic .

To switch back to the IL view use command View as instruction list .

Attention: A proper conversion presumes syntactically correct code. Otherwise parts of the implementation can get lost.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.36 View as ladder logic

Shortcut: <Ctrl> + <2>

This command (category 'FBD / LD / IL') is available, if you currently are working in the Function Block Diagram (FBD) or Instruction List (IL) view of the POU. It can be used in offline and online (online not available with the current version) mode.

Notice anyway, that there are some special elements, which **cannot get converted** and thus will only be available in the appropriate editor view. Further on some constructs might not be converted un-ambiguously:

FBD elements which cannot be displayed as LD elements (for example XOR) will be displayed as FBD boxes within a LD network.

To switch back to the FBD view use command View as function block diagram.

To switch back to the IL view use command View as instruction list .

Attention: A proper conversion presumes syntactically correct code. Otherwise parts of the implementation can get lost.

Note: Concerning the view options for the components of a LD / FBD network notice the settings in the 'FBD/LD and IL editor Options'.

12.13.37 View as instruction list

Shortcut: <Ctrl> + <3>

This command (category 'FBD / LD / IL') is available if you currently are working in the Function Block Diagram (FBD) or Ladder (LD) view of the POU. It can be used in offline and online (online not available with the current version) mode.

The FBD resp. LD networks will be converted to an instruction list.

Note:
Notice for conversions:
There are some elements, which cannot get converted and thus the respective network will remain in the original editor view. Example: operators CALC, CALCN. In case of syntax errors also no conversion is possible; a corresponding error message will be generated.
Some constructs might not be convertible unambiguously and thus will be "normalized" after a conversion from IL to FBD/LD and back. This concerns negations and explicit/implicit assignment of function block in- and outputs.

To switch back to the FBD view use command View as function block diagram.

To switch back to the LD view use command View as ladder logic .

Attention: A proper conversion presumes syntactically correct code. Otherwise parts of the implementation can get lost.

Note: Concerning the view options for the components of an Instruction List notice the settings in the FBD/LD and IL editor options'.

12.14 Find/Replace

The command category 'Find/Replace' provides commands for searching the project for certain strings.

12.14.1 Find/Replace commands

The 'Find/Replace' plug-in provides for command category 'Find/Replace' commands which can be used to perform a find and/or replace action concerning strings resp. characters in the editable places of the whole project.

By default the commands are contained in the 'Find & Replace' submenu within the 'Edit' menu. If necessary, open the Customize dialog, category Menu, to view resp. modify the current menu configuration (notice category 'Find/Replace').

The commands:

- Find
- Replace
- Find Next
- Find Next (Selected)
- Find Previous
- Find Previous (Selected)

12.14.2 Find

Symbol: 🔭

Default Shortcut: <Ctrl> + <F>

Use this command (category Find/Replace) to search the project for a certain string. All editable places within the project objects will be searched.

The 'Find' dialog will be opened, where you define which string should be searched according to certain rules, where it should be searched and whether it the found locations should be displayed one after the other or all on a whole. Also you can switch to the Replace dialog.

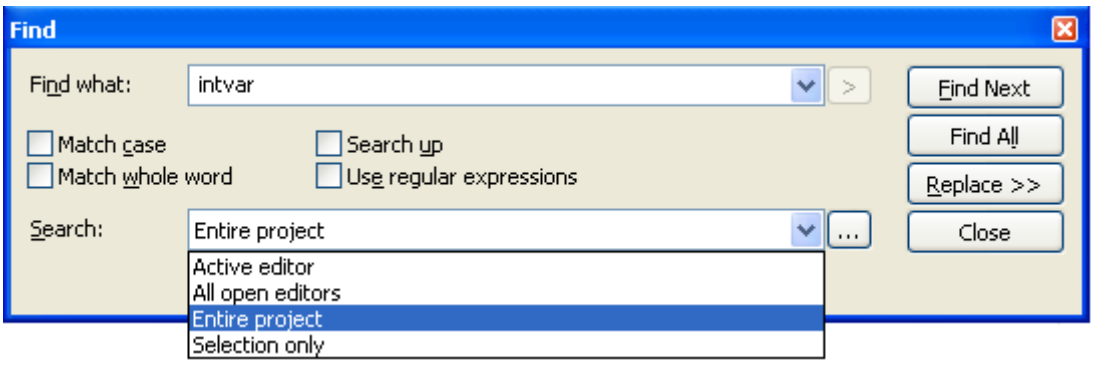


Abb. 120 Find dialog

Find what: Insert here the string to be searched. The selection list available via button will be filled with strings which have been already searched since the last start of »PLC Designer«.

Activate the desired search options:

Match case: The search is case-sensitive referring to the search string.

Match whole word: Only strings which match the whole search string will be found.

Search up: The specified search area will be passed upwards. Deactivate the option to search downwards again.

Use regular expressions: Regular Expressions (RegExp), the pattern matching standard for string parsing and replacement) is supported concerning the most commonly used expressions. Use the button to get assistance for entering the desired combination of those expressions in order to define, which strings resp. characters should be found. The available expressions are sorted in the following submenus: Special characters, Repetitions, Alternatives, Groups and Others.

Search: Specify here in which objects should be searched for the given string. For this purpose either choose one of the options offered in the selection list via the button or open the Search dialog via button ...:

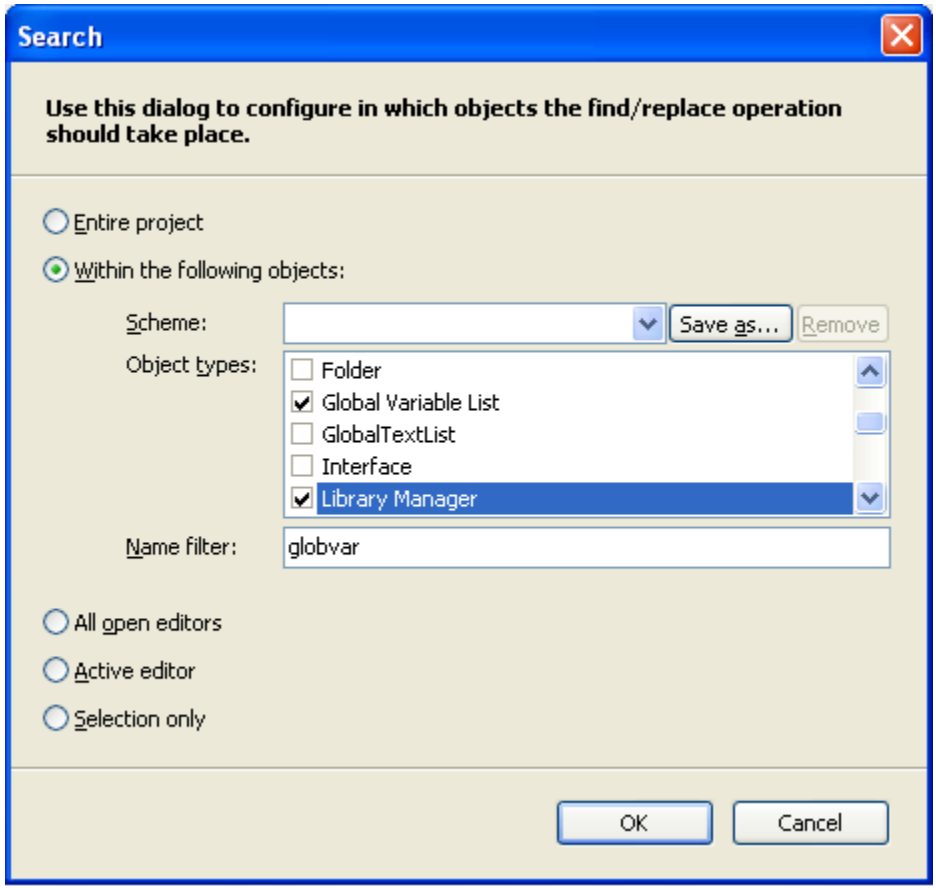


Abb. 121 Search dialog

Activate one of the following options:

- **Entire project**: All editable places within all project objects will be noticed.
- **Within the following objects**: Only the editable places within those objects will be noticed which are defined by the following settings:
  - **Object types**: Put a check to all object types which should be searched for.
  - **Name filter**: Optionally set a filter on certain objects names by using placeholders "*". Example: Enter "*CAN* to explicitly search for the specified search string in all objects including "CAN" in the object name.
  - **Scheme**: Optionally save the currently defined search configuration. Make sure to have set the desired Object types and optionally a Name filter. Then press button **Save as...** and in dialog **Save Scheme** define a name for the current configuration. All saved schemes will be available later in the selection list via button . They can be removed from there via button **Remove**.

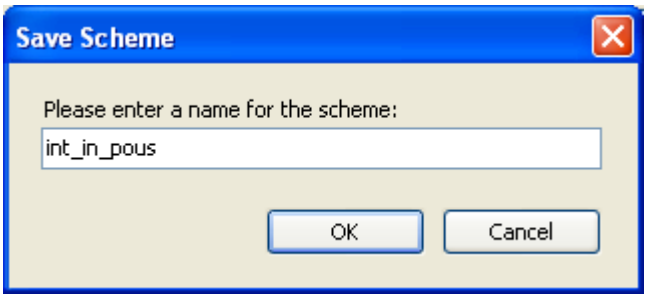


Abb. 122 Dialog Save Scheme

- **All open editors**: All editors currently opened in a window will be searched.
- **Active editor**: Just the editor where currently the cursor is placed, will be searched.
- **Selection only**: Only the currently selected text will be searched for the specified search string.

After having set all find and search options press button...

... Find Next to step through the found locations of the searched string step by step. The respective editor windows will be opened and the found string will be highlighted.

... **Find All** to get a list of the found locations in the Message window. The progress of the search process is displayed within the status line; the search may be interrupted by making use of the button **Cancel** in the status line. The search being completed the following information is displayed for each location:

Description: Expression containing the search string.

Project: Project name.

Object: Object name.

Position: Position (for example Line number) within the object, in brackets "Decl" for Declaration part resp. "Impl" for Implementation part of the editor window.

Below see the number of total found objects, of matching objects and total objects searched.

Example: Search results for string "b":

Messages

Find all 'b' (Entire project, search up)

Description	Project	Object	Position
globvar1:INT;	TS_Datentypeditor1	PLC_PRG	Line 7 (Decl)
fbinst:FB1;	TS_Datentypeditor1	PLC_PRG	Line 3 (Decl)
fbinst:FB1;	TS_Datentypeditor1	PLC_PRG	Line 3 (Decl)
aBool: BOOL;	TS_Datentypeditor1	DERIVED_DUT	Line 5
aBool: BOOL;	TS_Datentypeditor1	DERIVED_DUT	Line 5
in:BOOL;	TS_Datentypeditor1	FB1	Line 3 (Decl)
FUNCTION_BLOCK FB1	TS_Datentypeditor1	FB1	Line 1 (Decl)
FUNCTION_BLOCK FB1	TS_Datentypeditor1	FB1	Line 1 (Decl)
Total found: 13 - Matching objects: 4 - Total objects searched: 8			

If you would like to replace the found string by another one, press button **Replace** to get to the Replace dialog.

12.14.3 Replace

Symbol:

Default Shortcut: <Ctrl> + <H>

This command (category Find/Replace) opens the Replace dialog, which is an extended Find dialog.

Like in the Find-dialog first set the options for searching for the string which should be replaced by another one. Additionally enter the new string in the field at **Replace with**: and then use one of the following replace-buttons:

Replace: Press this button to perform a replacing of the first string which was found. In this case you can step to the next found string by button Find Next.

Replace All: Press this button, if you want to replace all found strings at once.

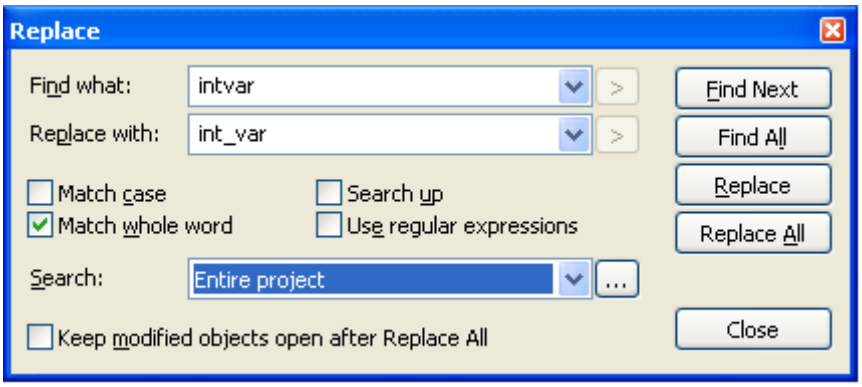


Abb. 123 Replace dialog

12.14.4 Find Next

Symbol:

Default Shortcut: <F3>

This command (category Find/Replace) is available to get to the next found position after the Find or Replace command has been used to search for a certain string. By default it is part of the Find & Replace menu within the Edit menu.

12.14.5 Find Next Selected

Default Shortcut: <Ctrl>+<F3>

This command (category Find/Replace) searches for the next string which matches that one which is currently selected in the editor.

12.14.6 Find Previous

Symbol:

Default Shortcut: <Shift>+<F3>

This command (category Find/Replace) is available to get to the previous found position after the Find or Replace command has been used to search for a certain string. By default it is part of the Find & Replace menu within the Edit menu.

12.14.7 Find Previous Selected

Default Shortcut: <Shift>+<Ctrl>+<F3>

This command (category Find/Replace) searches for the next string which matches that one which is currently selected in the editor.

12.15 File

The command category 'File' provides commands for handling the project file.

12.15.1 File commands

The 'File Commands' plug-in provides commands for command category 'File' which can be used for handling a project file.

It depends on the currently used feature set whether all commands of this category are available.

Per default most of the commands are part of the 'File' menu, some of the 'Project' menu. If necessary, open the Customize dialog, category Menu, to view resp. modify the current menu configuration.

The commands:

- New Project...
- Open Project...
- Close Project
- Save Project
- Save Project As...
- Project Archive: Extract Archive...
- Project Archive: Save/Send Archive...
- Save Project And Save Into Library Repository
- Save Project As Compiled Library
- Merge Project
- Recent Projects
- Exit
- Project Environment
- Project Information
- Project Settings
- Compare...

New Project...

Symbol:

Default Shortcut: <Ctrl>+<N>

This command (category 'File') is used to create a new project with the aid of dialog 'New Project'.

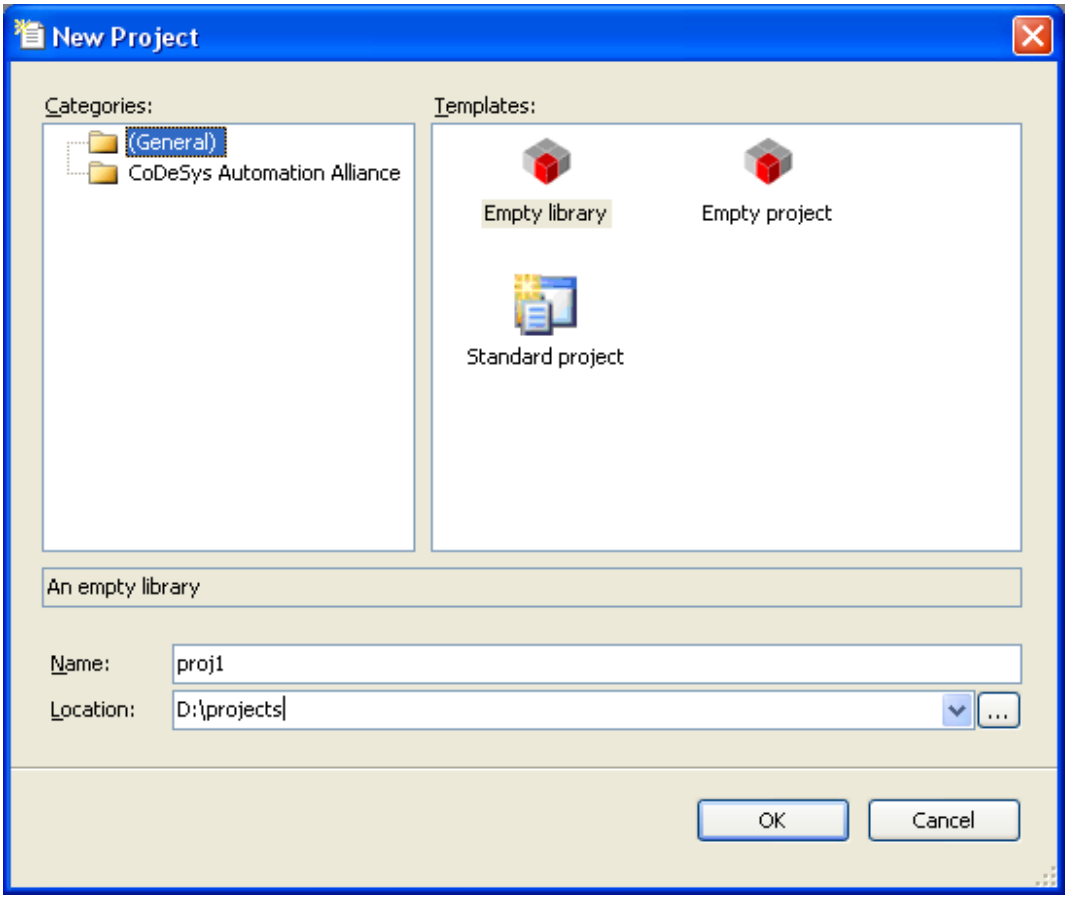


Abb. 124 Dialog - New Project

Categories: The available categories of templates and project wizards are offered in a tree structure. If a category is selected the assigned templates and wizards will be displayed in the Templates list on the right side.

Templates: This is a list of all templates/wizards belonging to the category currently selected in the Categories tree on the left side. A template determines the base configuration of a project file.
By default there is a template "Standard project" supported by a wizard (see below) as well as one for an "Empty Project" *.project resp.for a "Library project" *.library can be used, which do not yet provide any already inserted POU or device objects.

Name: Name of the project to be created. The default name is specified by the currently selected template/wizard and includes a numeric suffix guaranteeing the uniqueness within the file system (for example "Unbekannt1"). You can edit the entry considering the file path conventions of the operating system. Optionally you can add an extension (for example ".project"), by default automatically the extension defined by the currently selected template/wizard will be assigned.

Location: Location of the new project file. The default path is specified by the currently selected template/wizard. You can use the Windows standard browser for modifying the path via button ... or choose one of the recently used location paths via button .

OK: A new project will be created according to the done settings. If any settings are missing, an error icon will be displayed at the respective entry field in the dialog. When the cursor is placed on an error icon, a tooltip will provide a hint what to do. If a project was already opened when setting up the new one, you will now be asked whether this should be saved and closed before creating the new project.

The name of the new project in each case will be displayed in the title bar of the frame window of the programming system. Note: An asterisk behind the project name will indicate that the project has been modified since the last save.

Wizard for Standard project:

If you choose the Standard project template, on **OK** a wizard dialog will appear:

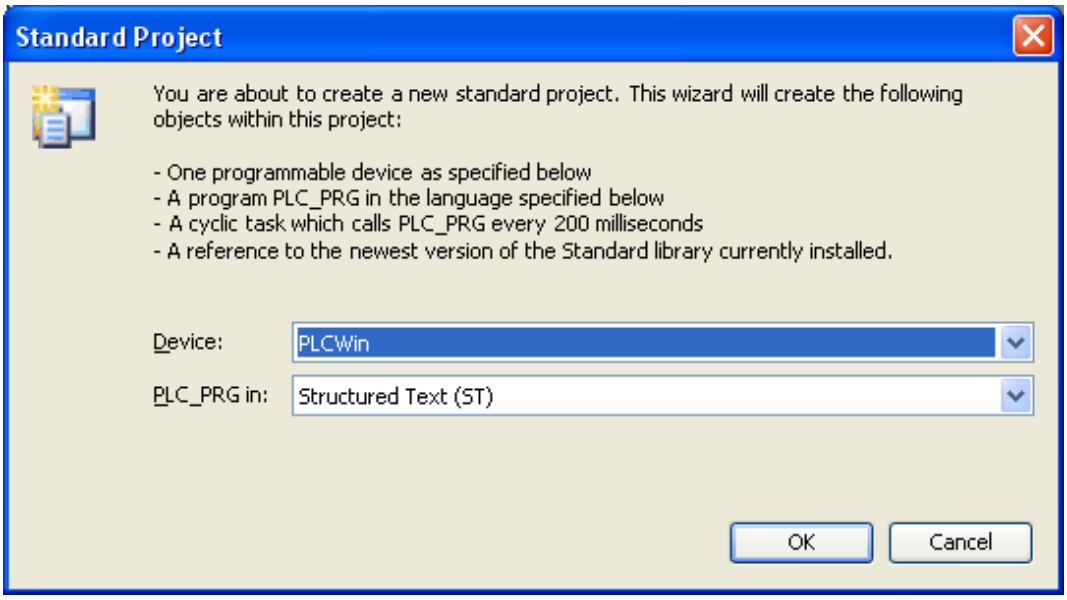


Abb. 125 Dialog - Standard project wizard

Confirmation with **OK** will create the following standard project

The project contains a device as selected from the **Device** selection list, an **Application** "Application", a **Library Manager**, a program POU "**PLC_PRG**" and a **Task Configuration** with a task "**MainTask**" in the device tree. PLC_PRG will be set up in the programming language selected from the selection list at **PLC_PRG in**:. The task will be a cyclic one, calling PLC_PRG every 200 milliseconds. The newest version of the currently installed library **Standard.library** will be included in the Library Manager.

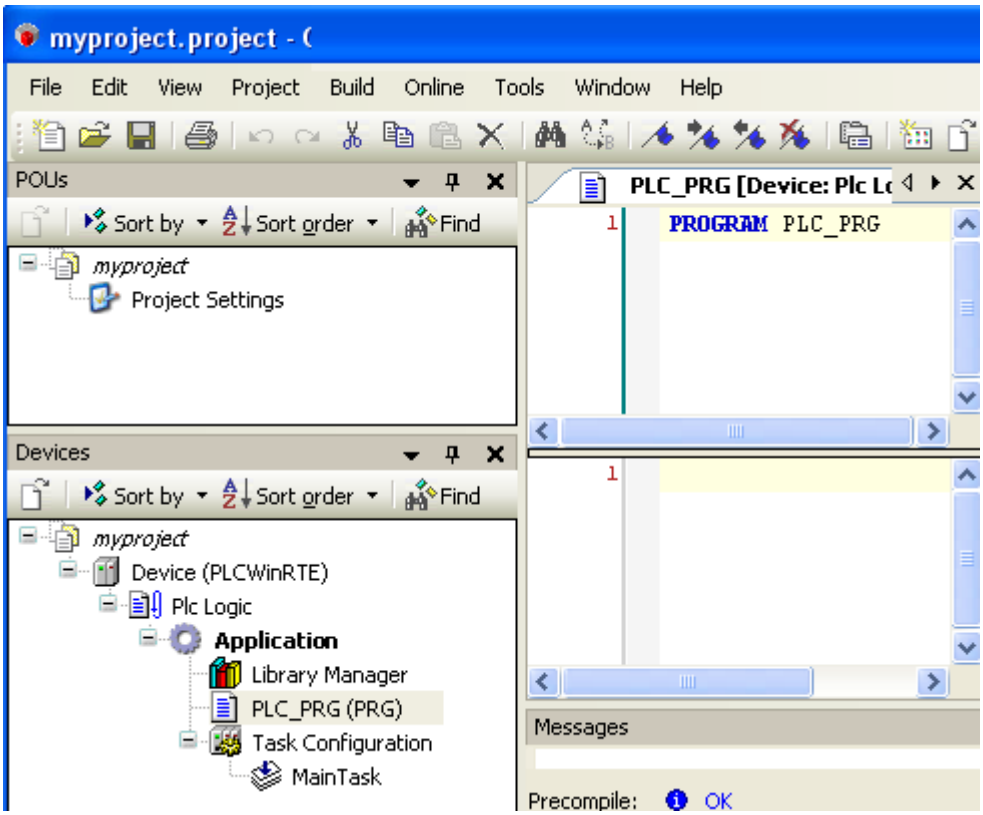


Abb. 126 Example - Standard Project

Open Project...

Symbol:

Default Shortcut <Ctrl> + <O>

This command (category 'File') can be used to open an existing project file with the aid of the standard dialog for opening a file. If an appropriate converter is installed, also projects of different formats can be opened.

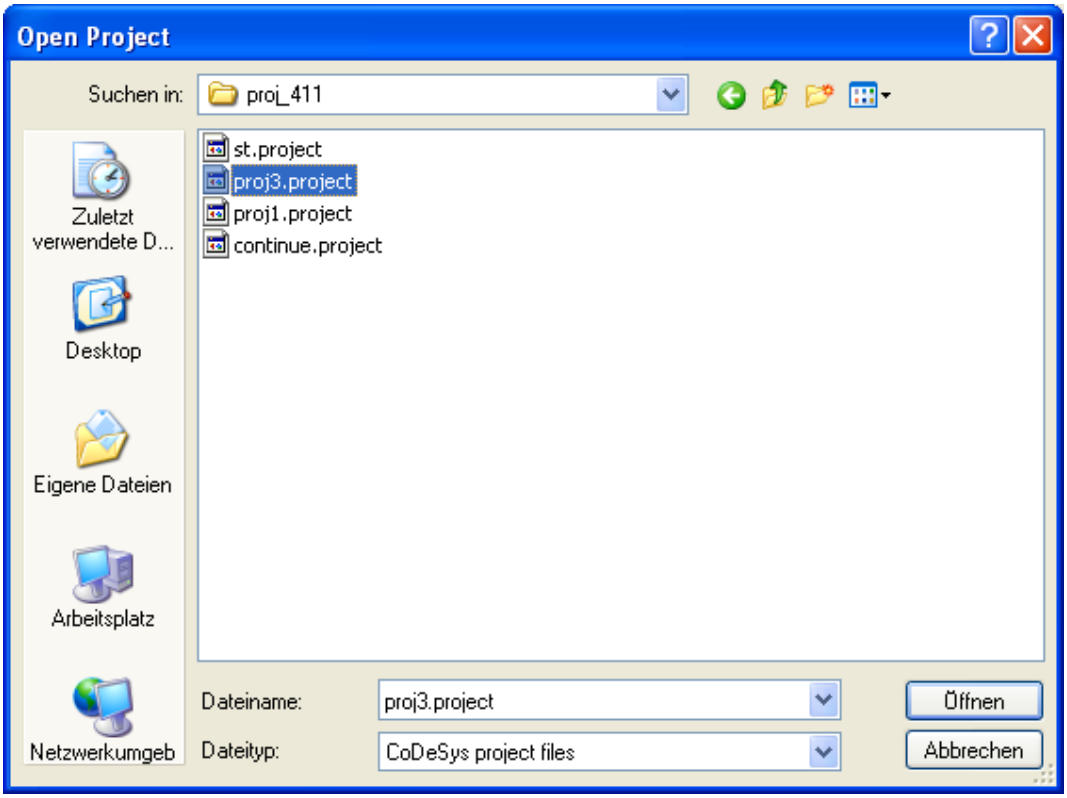


Abb. 127 Dialog 'Open Project'

File type: Select the filter for one of the following default types:

- »PLC Designer« project files: new »PLC Designer« projects, extension ".project"
- »PLC Designer« library files: new »PLC Designer« projects, extension ".library"
- All files: all available files

additionally there might be further file type options, depending on the available project converters:

- »PLC Designer« project files (before V3.x): Files with extension ".pro" will be displayed
- »PLC Designer« library files (before V3.x): Files with extension ".lib" will be displayed

Open: The chosen project will be opened resp. converted and opened resp. refused to get opened. The following cases are possible:

(1) Another project is still open

(2) Project was saved with different version of V3.x

(3) Project was not saved with V3.x

(4) Project was not terminated regularly and AutoSave was activated

(5) Project is read-only

(6) You try to open a library project out of a library repository folder

1) Another project is still opened: You will be asked whether it should be saved and closed.

2) The project was saved with an older version of »PLC Designer« V3:

If the **storage format** is different because the loaded project had been stored with an older version of »PLC Designer«, two cases are possible:

1. If the project cannot be stored with the storage format of the currently used programming system version, you must do an update in order to be able to continue working on this project. Do not get confused by the expression "your changes" used in this message, at this place it means changes caused by internal actions of any components during project loading.

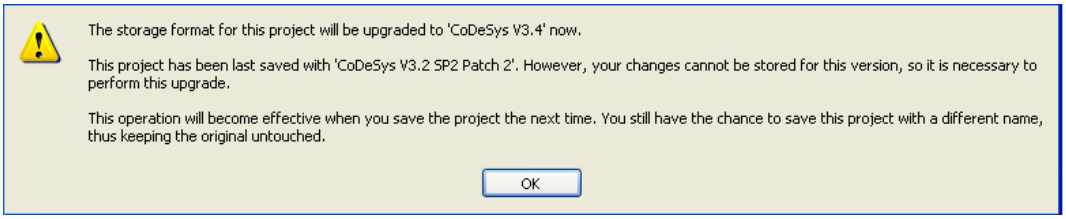


Abb. 128 Message box concerning required storage format update, example

2. If the project allows to get stored in the old format, you can choose whether you want to do this - however accepting possible data losses - or whether the storage format should be updated, whereupon however the project will not be usable any longer with the old version of the programming system.

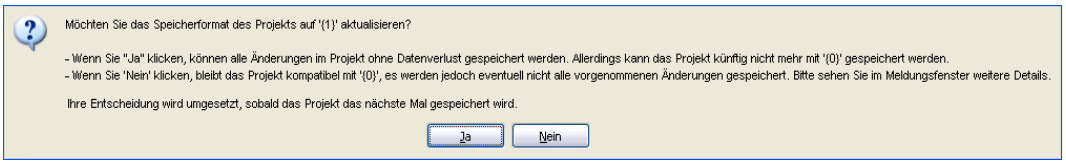


Abb. 129 Message box offering storage format update, example

Project Environment:

Besides the storage format also the **versions of libraries** (explicitly included ones), the **visualization profile** and the **compiler version** of the project to be loaded might be different to those installed with the current programming system. In this case the **Project Environment** dialog will open with the respective subdialog(s), where you can do version upgrades, if there are newer versions installed on your system. Regard that, if deciding not to update any version setting now, you still will have the possibility to do that later. The Project Environment dialogs are always accessible via command 'Project Environment'. Please see Project Environment for further description.

Note: If you load an old project for which the compiler version is "newest" (Project Settings), while here in the new programming system the Project Environment setting for the compiler version is "Do not update", then the version most recently used in the old project will be used (not the "newest" in the new environment).

3) The project was not saved with »PLC Designer« V3.x:

- If you have used the appropriate file type filter when selecting the project file to be opened, automatically the corresponding converter -if available - will be used and the project will be converted to the new format. Notice that the way of conversion depends on the currently used converter. Usually, If necessary, you will be asked to define a library or device replacement as described below: Library Reference Conversion, Device Conversion. See also the notes below this list.
- If you have set the file type 'All files' when selecting the project file to be opened, no converter gets activated and the dialog '**Convert Project File**' will appear, where you explicitly have to activate the conversion of the project by choosing one of the options:

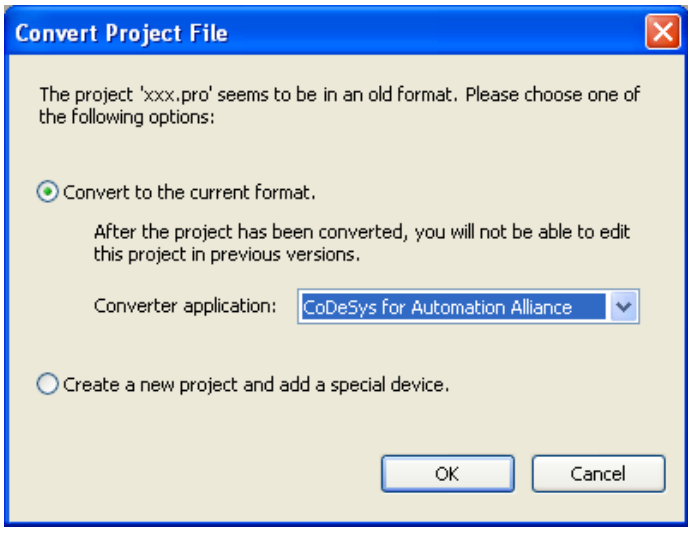


Abb. 130 Dialog 'Convert Project File'

- **Convert to the current format**: Chose from the selection list the required converter. Notice that after conversion the project cannot be used in previous versions.
- **Create a new project and add a special device**: - **not yet implemented**

See also the notes below this list.

- If libraries are included in the old project, for which up to now no "conversion mapping" has been stored in the library options, the dialog '**Library Reference Conversion**' will open, where you can define how to convert these references:

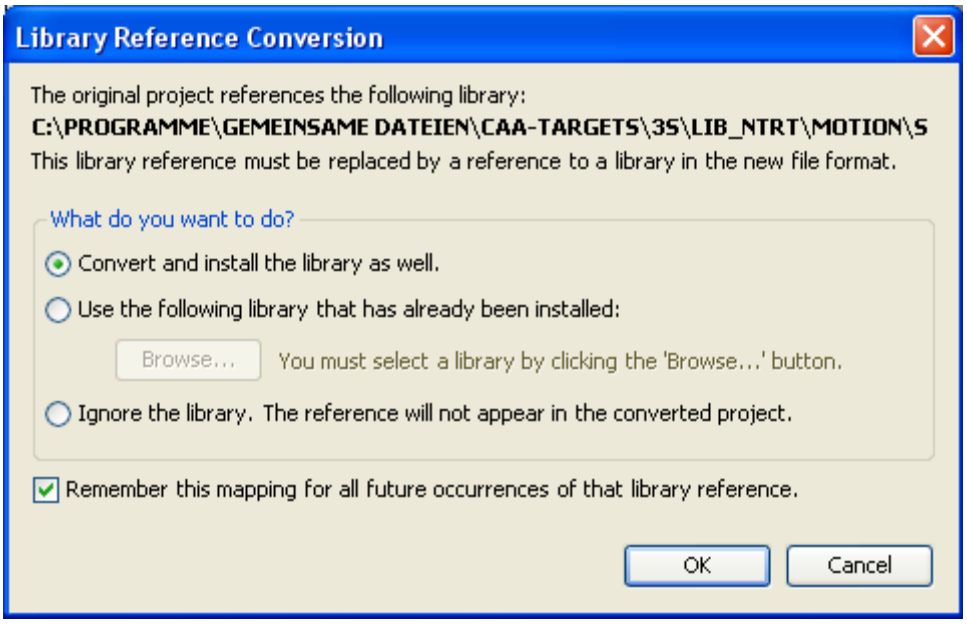


Abb. 131 Dialog 'Library Reference Conversion'

(For general information on the library management please see Library Management)

Convert and install the library as well: The referenced library will be converted to the new format and remain referenced in the project. Thereby it will be automatically installed and used in the project further on. If the library does not provide the required Project Information (Title, Version, Company), you will be asked to enter them in dialog 'Enter Project information'. The library by default will be installed in category 'Miscellaneous'.

Use the following library that has already been installed: Activate this option if you want to replace the currently included library by another one, which is already installed on the local system. The **Browse** button opens the **Select Library** dialog. Here you can select the desired version of one of the currently installed libraries. This corresponds to configure the version handling of the library in the Library properties dialog. "*" means, that the newest library version available on the system will be used in the project. The list of available libraries is structured like described for the Library Repository dialog, you can sort by Company and Category.

Ignore the library. The reference will not appear in the converted project.: If this option is activated, the library reference will be removed and the library will not be included in the converted project.

Remember this mapping for all future occurrences of that library reference.: If this option is activated, the current settings of the dialog will be applied also for future project conversions when the respective library is referenced.

In the converted project the library references will be available in a global Library Manager in the POUs window.

After having finished the library reference conversion, the project conversion will continue with the dialog 'Open Project' as described above.

- If a V2.3 project is opened which includes a device setting, for which not yet a "conversion mapping" is defined in the »PLC Designer« 2.3 Converter options, the dialog '**Device Conversion**' will open, where you can define whether and how to replace the old device definitions by currently valid ones:

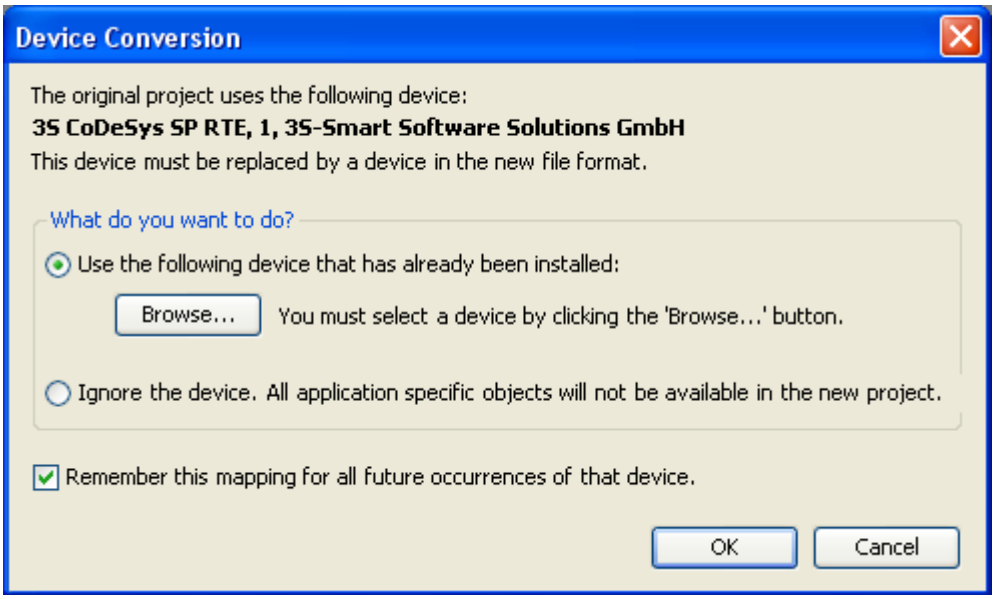


Abb. 132 Dialog 'Device Conversion'

(For general information on devices please see Device, Device Tree)

The originally used device is indicated and you have to decide what should happen with that device reference in the new project. Choose one of the following options:

Use the following device that has already been installed: The **Browse** button opens the dialog '**Select Target Device**', where you can choose one of the devices currently installed on the local system. This device then will be inserted instead of the old one in the devices tree of the new project.

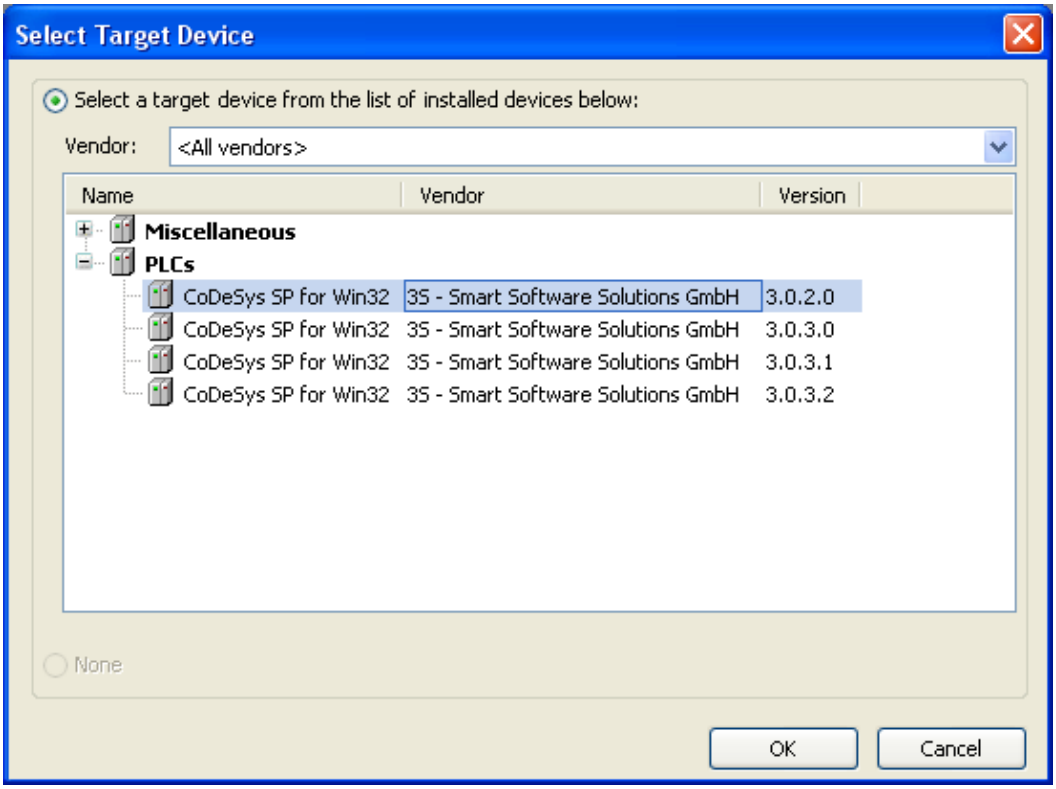


Abb. 133 Dialog 'Select Target Device'

Activate option **Select a target device...** in order to be able to choose one of the listed device entries. The usage of the dialog (sorting by vendor and categories) corresponds to that of the device repository dialog.

Ignore the device. All application specific objects will not be available in the new project.: In this case the device tree in the new project will not get an corresponding device entry at all and application specific objects like for example the Task configuration will not be transferred as well.

Remember this mapping for all future occurrences of that device.: If this option is activated, the settings of the dialog, that is the currently defined conversion mapping for the device, will be stored in the »PLC Designer« 2.3 Converter options and thus be applied at further conversions.

Note: For the »PLC Designer« V2.3 converter: Adapting a V2 project with respect to the V3 syntax on import can only be done if the project can be compiled in the converter without detected errors.

Note: For the »PLC Designer« V2.3 converter: The project directory defined in the V2.3 project options as well as the project information will be applied to the Project Information dialog.

4) If the **Auto Save** function had been activated and »PLC Designer« had been terminated non-regularly before the last project was saved after a modification, you will get the 'Auto Save Backup' dialog when going to re-open the same project. For details see Options for Load and Save.

5) If the project you want to open is **read-only** on disk, you will be asked whether you want to open the project in read-only mode or whether you want to make the project writable.

6) If you try to open a **library project** installed in a Library Repository you will get an error message. The library project cannot be opened from the given path.

After closing the dialog with **OK**, the project name will appear in the title bar of the user interface frame. An asterisk behind the project name in the title bar indicates that the project has been modified since the last save.

Close Project

Batch command: file close (not yet implemented)

This command (category 'File') will close the project while the programming system will stay opened. If the project contains non-saved modifications, you will be asked whether you want to save these changes.

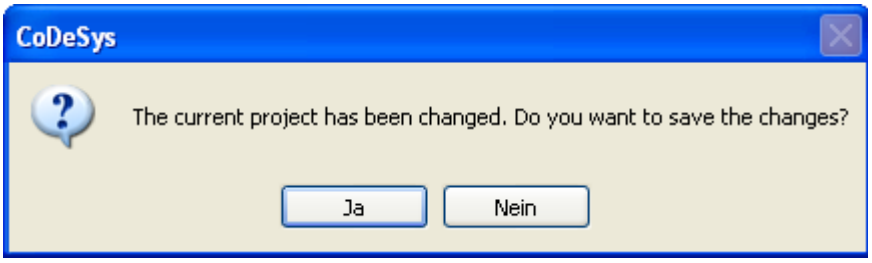


Abb. 134 Dialog asking whether to save a changed project:

For terminating »PLC Designer« use command Exit.

Save Project

Symbol:

Default Shortcut <Ctrl> + <S>

Batch command: file save (not yet implemented)

This command (category 'File') is used to save the project at the currently defined location. It is only available if any modifications have been done in the project since the last saving. This is indicated by an asterisk behind the project name in the title bar.

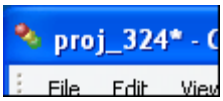


Abb. 135 Title bar, modified project

In case the project is write-protected the following message dialog will be opened:

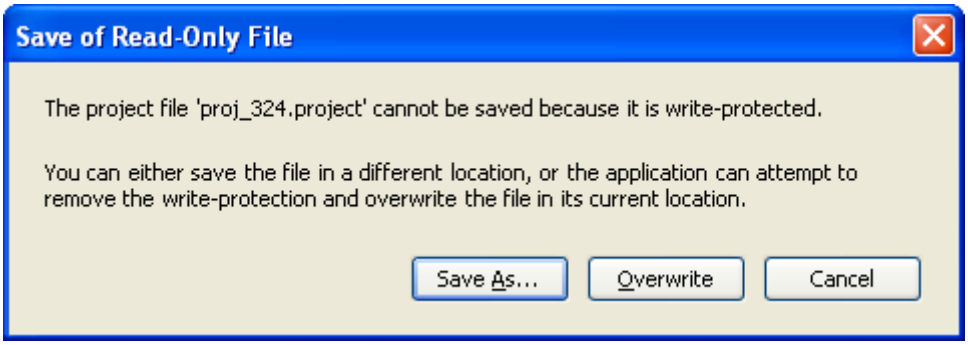


Abb. 136 Message dialog, project write-protected

Save As.. allows to define a new project path.

Overwrite removes the write protection from the project file and saves the project.

Backup file:

Optionally a project backup file can be kept. If option **Create backup files** is activated in the options dialog for 'Load and Save', at each saving the project will be copied to a file *Projectname.backup* additionally to getting saved to *Projectname.project*.

Save Project As...

This command (category 'File') is used to save the project, whereby location and type of the file can be defined. The Windows standard dialog for saving a file will be opened for this purpose.

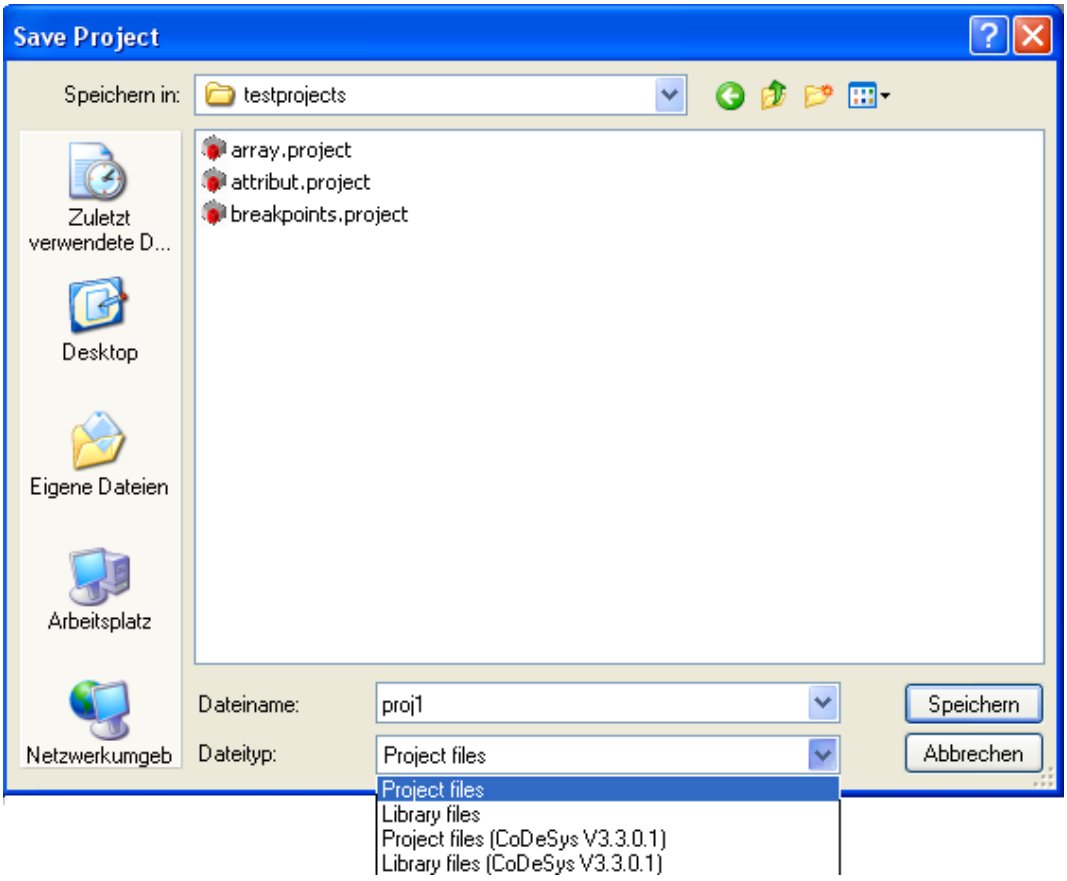


Abb. 137 Save Project dialog

Perform the following steps:

1. In the **Save in** and **File name** fields per default the currently defined file location and file name are already set. If desired browse for another path and/or edit the file name entry.
2. In the **Data type** field choose one of the available file types. The selection list contains for normal projects as well as for library projects all versions of the programming system for which a project can be stored. If the project should be opened later in an older version, it is reasonable to store it exactly for this version because then you will get immediately informed on possible data loss. Regard in this context also the default project settings for the storage format. Basically there are two types:

   Project files r The project will be saved as a »PLC Designer« project file "<filename>.project" for the currently used resp. for the selected version of the programming system.

   Library files resp. **Library files (**»PLC Designer« **V<version>)**: The project will be saved as a »PLC Designer« library file "<filename>.library" for the currently used resp. for the selected version of the programming system.
   Regard the following before saving that project as a library:

- If the library should later be suitable for getting installed in other projects, enter at least a title and a version number, optionally (recommended) also a category and the company name in the Project Information (Summary).
- The saving of a library project does not include an automatic check for errors!
- Regard that **in contrast to »PLC Designer« V2.3** there is no more differentiation between "external" and "internal" libraries. Now you can specify in the properties of each object, whether it should be handled as "external".
- Consider whether you at once want to install the created library to the System Library Repository. In this case use command Save Project And Save Into Library Repository.
- If the library project should be protected against later modifications, activatevoption Released in the Project Information dialog. This will effect, that later at the first try to re-edit the library project, an explicit confirmation will be forced that the modification is really wanted.

If there is already a file with the defined name, a **message dialog** will open to ask you whether that file should be replaced. If you choose No, you will get back to the 'Save Project' dialog described above to select another path. If you are going to save the project in another version of the programming system than currently used, you will get made aware of possible data loss.

Project Archive: Extract Archive...

This command (category 'File commands') is used to extract an archive file (default file extension .projectarchive) that has been created by use of the command Save/Send Archive.

Regard that extracting an archive will **require to close all currently loaded projects** in any of the currently opened instances of the programming system. This is to avoid direct impact on running projects if for example libraries are changed due to the extract operation.

The archive file can be selected via the standard dialog box that arises in response to the command execution. After confirmation of the selection with **Open** there will pop up a dialog box:

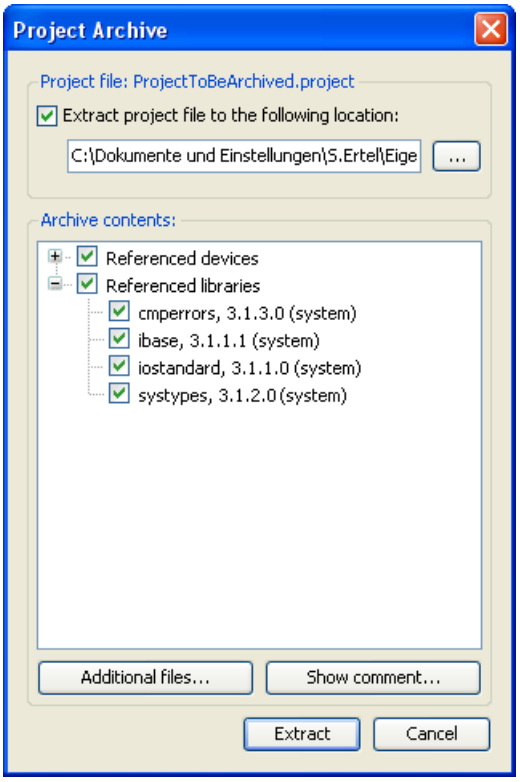


Abb. 138 Project archive dialog

General: Under **Extract archive content relatively to:** the path to which the archive is to be extracted to is specified. You can modify the target directory either by hand (click on the path indication and modify it by typing) or by a click on ..., what gives rise to the standard dialog for browsing.

The lower rectangle of the dialog box displays the file categories packed in the archive. A click on the sign +, which antecedes each category, will expand a list of its associated files. All categories mentioned are marked by ☑, what indicates that all of the corresponding files will be extracted. If you want to prevent certain files or even an entire category from being extracted, you have to deactivate their marking by a click (the mark will then appear as ■ for partial deactivation, respectively ☐ for entire deactivation).

The archive may contain other than the project files that have been added explicitly.

After a click on **Additional files...** there will pop up the following dialog box:

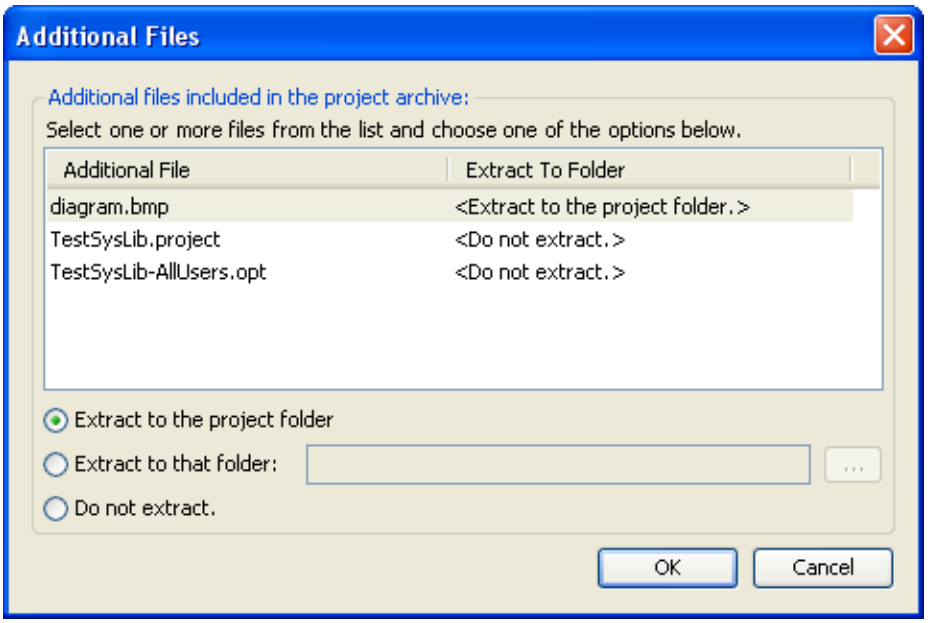


Abb. 139 Additional files dialog

where a complete list of the additional files is displayed. By default, an additional file will not be extracted together with the project files (see the remark **<Do not extract.>** to the right of the file name). If you want an additional file to be extracted, you have to select it by a click on its name. Afterwards you have to choose one of the three options provided below:

- **Extract to the project folder** will extract the selected file to the same directory as the project files (specified by the path). The remark to the right of the file name will be changed into **<Extract to the project folder.>**.
- **Extract to that folder:** will extract the file to the folder that you can specify either by typing the corresponding path into the text field or by making use of the standard dialog for browsing after a click on ... . The remark to the right of the file name will be changed to the specified path.
- **Do not extract.** will reset the selected file to the default mode.

After adjusting the extract properties of the additional files you may exit the Additional file dialog pressing **OK**.

A click on **Show comment...** will display the comment eventually given on the archived project by its author. If no comment is contained in the archive, a corresponding error message will be displayed instead.

Having configurated the set-up in the dialog box you may click on

- **Extract** to unpack the archive to the specified path.

  If a file that oughts to be extracted is named in the same way as a file already existing in the target directory, a warning message will be displayed and you will be asked whether you want to replace the local file or not. You can make your decision become valid for every further name mismatch by activating the option ☑**Apply to all items and files**.

- **Cancel** to abort the operation; the archive will not get extracted.

Project Archive: Save/Send Archive...

This command (category 'File commands') is used to set up and create a project archive file, and all files which are referenced by and used within the currently opened project are packed. The archive file (<filename>.projectarchive) can either be stored or sent as attachment of an email. The latter is useful to give forward the set of all project relevant files as the archive can be unpacked easily by use of the command Extract Archive.

Note: The archive function is not intended for restoring a project environment. It is designed for an easy packing of all files belonging to a project.

After executing the command the dialog box **Project Archive** opens:

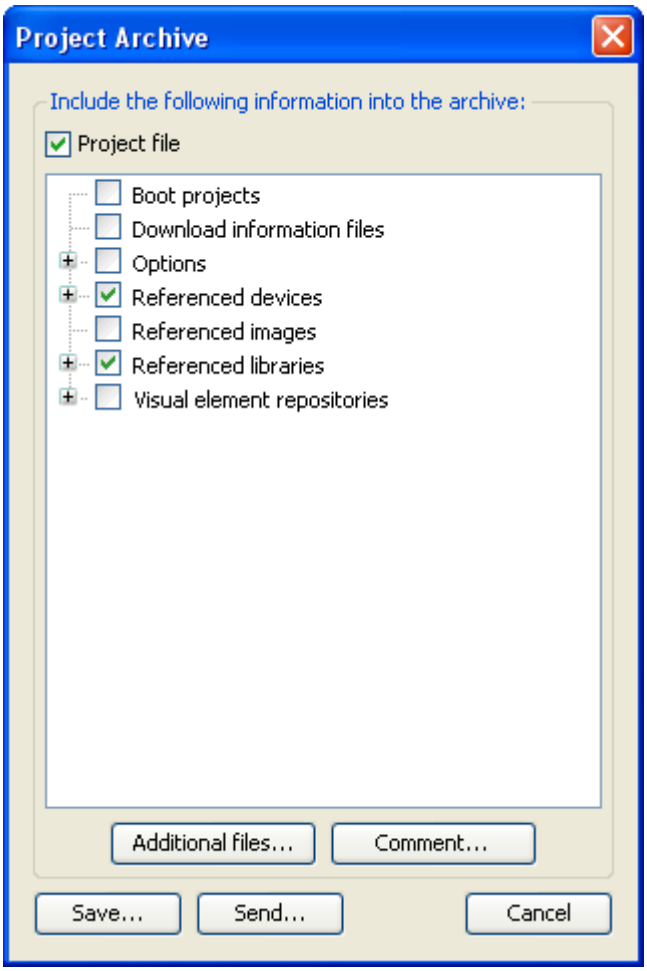


Abb. 140 Project Archive dialog

In this dialog box you can define, which file categories should be added to the archive file: Select or deselect a category by activating/deactivating the corresponding checkbox. Do this by a single mouse click in the checkbox or by a double click on the category name. If a category is marked by ☑, all files belonging to this category will be added to the archive file. A category is marked by ☐, if the project does not include any corresponding file.

To select/deselect a single file of a special category press the aligned button ⊞ to get a list of all associated files. By default, all files of an activated category are selected. To modify a selection activate or deactivate the desired files in the same way as the categories. Now, if not all but only certain files of an category are activated, its mark will appear as ■.

To add any other files to the archive then the ones listed above, press the button **Additional files...,** and the corresponding dialog box opens.

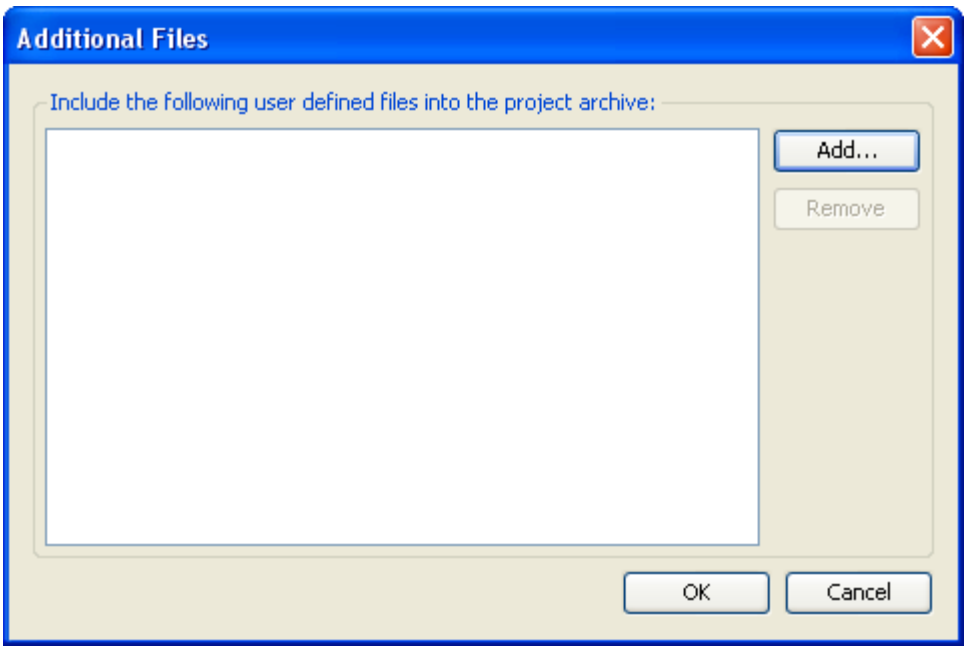


Abb. 141 Additional files dialog

Press the button **Add** to open the standard dialog, where you can browse for files (sorted by file type) and open the ones you selected (by confirmation with **Open**). The selected files will be added to the list of the **Additional files...** dialog. To remove a file from this list, select it by a click and press the button **Remove**. After the list of selected files fits to your demands, close the dialog with **OK**.

To add a note on the project press the button **Comment...**. You may enter your comment in the rising text editor before closing it with **OK**. When your archived project will be imported, your comment can be accessed in the corresponding dialog via **Show comment...**.

After carrying out the desired enlistment choose one of the following items of the main dialog:

- **Save...** to create and store the archive file to a desired path that you select via the opening standard dialog. Therein, you may also change the default name <projectname>.projectarchive of the archive file. Confirm with **Save** to start the building process.
- **Mail...** to create a temporary archive file that will be attached to a simultaneously generated empty email.
  The successful operation of this feature relies on a correct installation of MAPI (Messaging Application Programming Interface). If the operation has been unsuccessfull, an error message will be displayed.
  The temporary archive will be deleted automatically as soon as the message has been sent.
- **Cancel** to cancel the action; no archive file is generated.

Save Project And Save Into Library Repository

Symbol:

This command (category 'File') saves the project as a library in the "System" Library Repository. This is an extension to saving a project as a library via command "save Project As...", because the library additionally gets installed on the local system and thus at once is available for insertion in a project via the Library Manager. **Please see** the description of command Save Project As... concerning what you must regard for creating a library.

Save Project As Compiled Library

This command (category 'File') can be used to save a library project in encoded format. To be exact, an encoded image of the "precompile context" of the library will be created, which means that the implementations of the library modules will not any longer be accessible and visible.

The command opens the standard dialog for saving a file in the file system. The file type is automatically set to "**»PLC Designer« compiled library files**". By default the current project file name is pre-set and can be modified if desired. The file extension is "**.compiled-library**".

The further handling of *.compiled-library files is the same as for *.library files, that is they can be installed and referenced in the same way.

So the source code of a library can be protected and additionally shorter loading times and smaller library files are obtained.

Note: The usual debug functions can not be applied on "compiled libraries", that is stepping into a library module is not possible.

Merge Project

THIS FEATURE IS CURRENTLY NOT AVAILBLE

Use this command (category 'File') to merge two »PLC Designer« projects, i.e. to copy another project into the current project. Per default this command is not in the 'File' menu, but in the 'Project' menu.

'Merge Project' dialog:

The command opens the 'Merge Project' dialog, which is the Windows standard dialog for browsing for a file. Here you define the project to be copied.

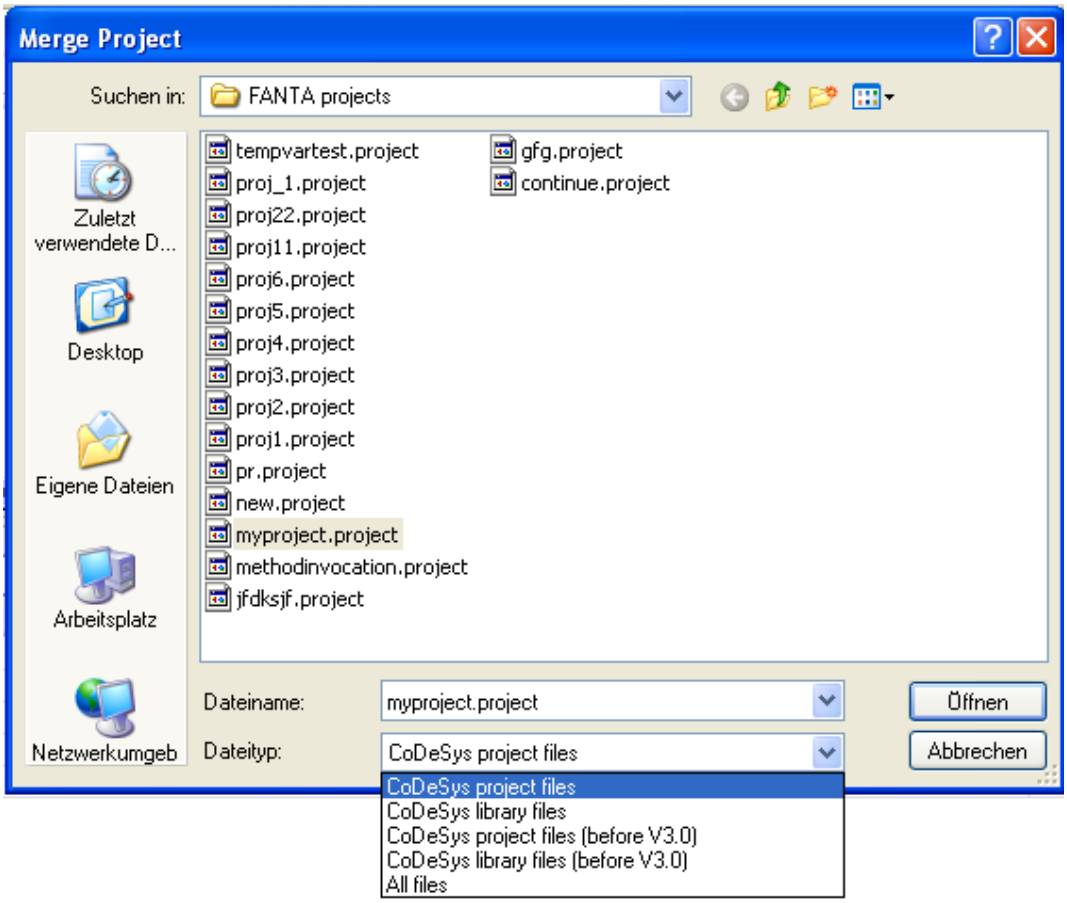


Abb. 142 Merge Project dialog

The filter for the **File type** per default is set to "»PLC Designer« project files" (extension ".project"), which are projects created with »PLC Designer« versions later than V2.3. The same is true for "»PLC Designer« library files". Which further file types are offered, depends on the currently installed converters.

Choose the desired project file and press **Open**.

If the file to be merged is of a different format and conversion is needed, the user will get the appropriate dialogs, like when opening a project of different format. See Open Project for a description.

'Merge Project-<project to be copied>' dialog:

After having chosen the project to be copied and - if necessary after a conversion - the 'Merge Project-<project to be copied>' dialog will be opened. Here you define, which objects of the project should be copied to the currently edited project:

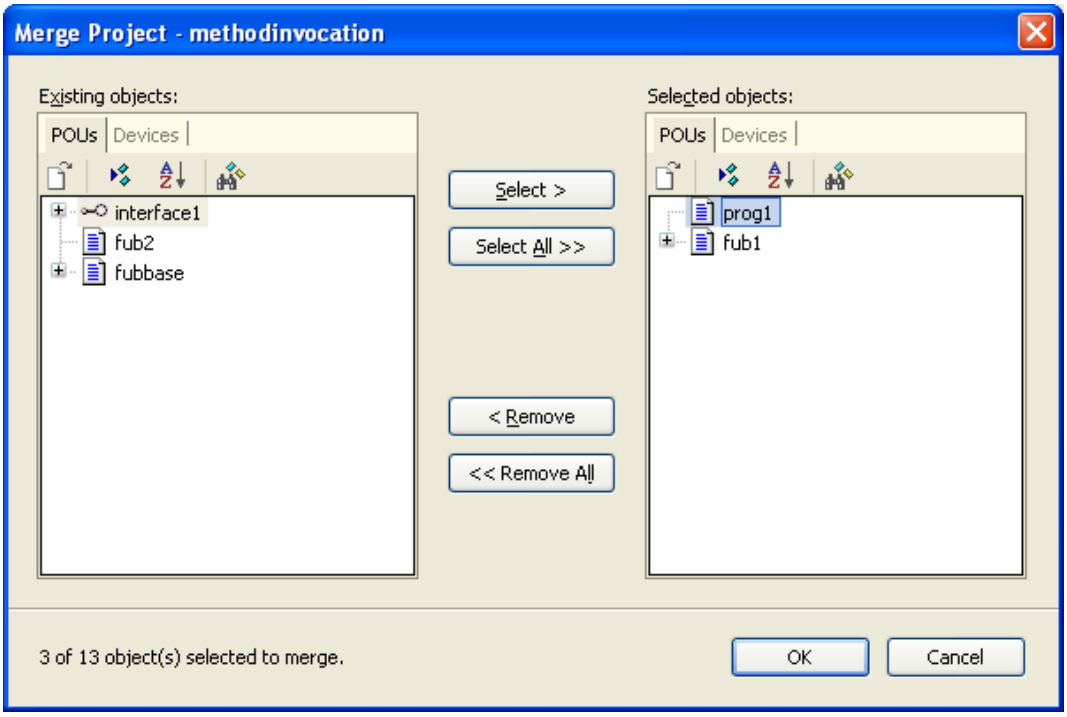


Abb. 143 Dialog Merge Project, Selecting objects

The objects of the chosen project are displayed in tabs representing the projects views (POUs, Devices..). The tree structure and the navigation options (icons above the selection list) correspond to that of the respective view.

Select the desired objects. For multiple selection keep the <Shift>-key pressed, whereby <Ctrl>+ mouse-click can be used to remove a single selection. The **Select all** resp. **Select none** buttons are available for selecting resp. deselecting the complete objects tree.

Press **OK** to start the copy process.

If naming conflicts are detected during the selection configuration, the following dialog will be displayed:

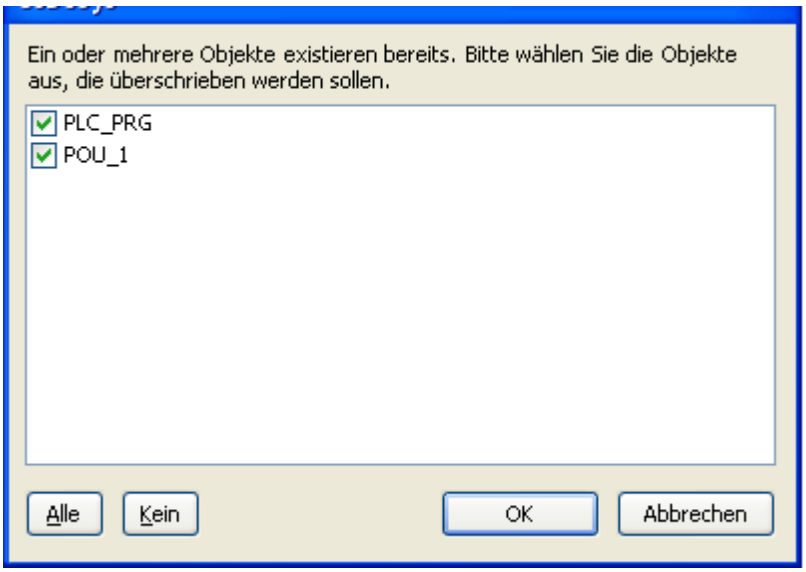


Abb. 144 Dialog for selecting objects to get overwritten

The list shows all objects which have been selected for getting copied but are already existing in the currently edited project.

All objects provided with a check mark will be overwritten during copying. The others will be copied but renamed automatically with a unique name.

All will add a check mark to all entries, **None** will remove all check marks.

OK makes the copy process carry on and the copied objects will be added inserted in the object trees of the currently edited project.

Recent Projects

Use this command (category 'File') to select from a list of the most recently opened projects that one you want to reopen. The number of projects which are kept in this list is defined in the Options for 'Load and Save' (per default in the Tools menu)

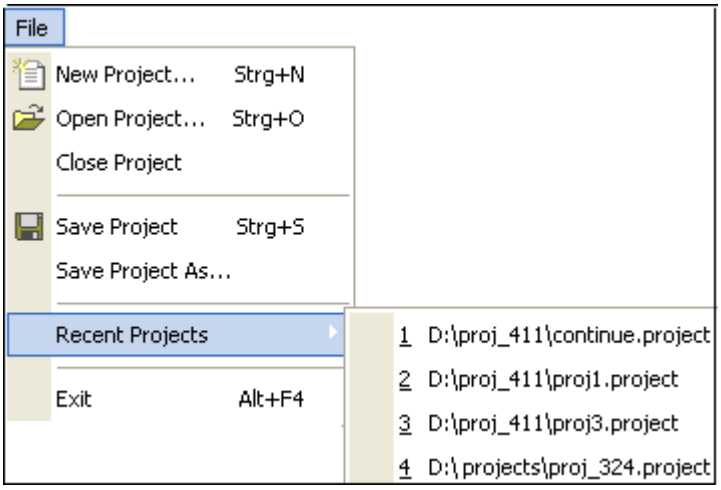


Abb. 145 Recent Projects list

Exit

Shortcut <Alt> + <F4>

This command (category 'File') will terminate the programming system. If currently a project is opened which has been modified since the last saving, a dialog will open asking you whether the project should be saved.

Compare...

Symbol:

By use of the command **Project / Compare** (category 'File commands') you may compare the actual project with another one ("**reference project**"). The reference project and the compare options are specified in the Project Compare dialog, which will be opened, when the command is performed.

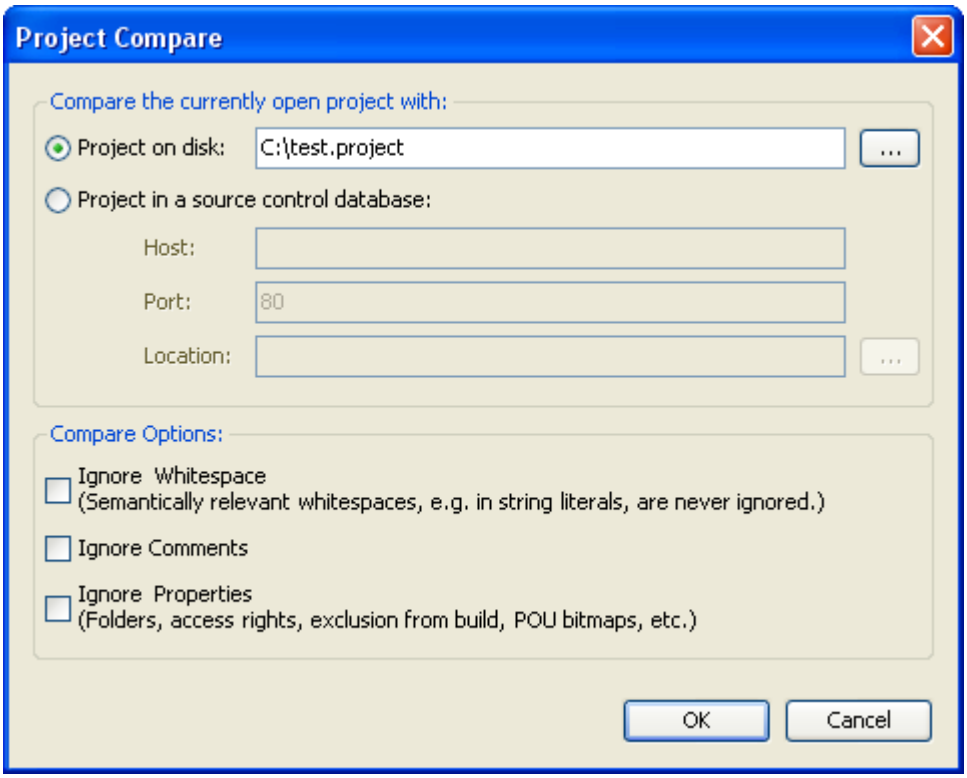


Abb. 146 Project Compare dialog

Compare the currently opened project with:

The reference project can be

- either a **Project on disk**: by default the path of the actual project itself is entered, so that it'll be compared against its version latest saved. You can replace it by modifying the file path after a click on the text field. Alternatively, you may make use of the standard dialog box for browsing after a click on ...
- or a **Project in a source control database**: where you first have to specify the host name as well as the number of the port. Afterwards you may insert the path of the designated project in the text field or select it via the standard dialog box for browsing after a click on ...

Compare options:

Here one or several of the following options concerning the comparison can be activated:

- **Ignore Whitespace**: Discrepancies due to a different number of blanks will not be mentioned.
- **Ignore Comments**: Comments will be excluded from comparison.
- **Ignore Properties**: Object properties will not be compared.

After closing the dialog 'Project Compare' with **OK,** the comparison is executed according to the settings.

Survey of comparison result by means of marked device trees

The result of the comparison is represented in a new window entitled "**Project Compare - Differences**".

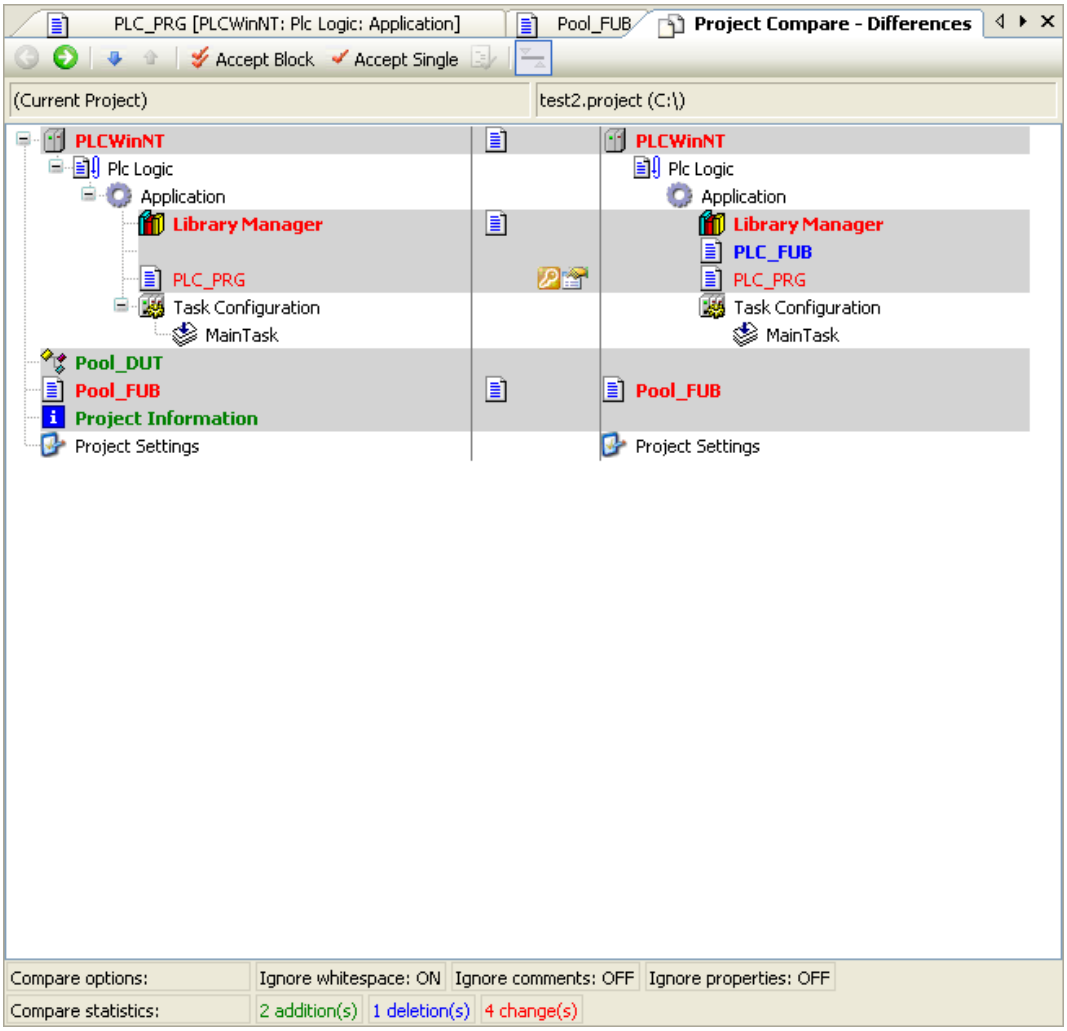


Abb. 147 Example of comparison result

On top of the new window a toolbar (see below, "Working in the compare mode") is placed at your disposal, followed by a title bar and a sub-window. The title bar as well as the sub-window itself is subdivided into a left part representing the current project and the right one representing the reference project. The corresponding file paths are displayed in the title bar whereas the sub-window shows the two device trees associated to the projects.

Therein, you see the name of identical units displayed in black with no further remarks. Otherwise the name of an unit is displayed

- in **bold blue**, if it exists in the reference project only; instead of its name a gap will be inserted at the corresponding place in the device tree of the actual project.
- in **bold green**: if it exists in the actual project only; instead of its name a gap will be inserted at the corresponding place in the device tree of the reference project.
- in **bold red** in both parts of the window, if their are differences in the two versions of the unit concerning the implementation. In addition the unit name is followed by .
- in red, if the differences of the versions are related only to properties (indicated by subsequent to the unit name), access rights (indicated by subsequent to the unit name) or added folders (indicated by subsequent to the unit name). In the last case, the corresponding path of the unit concerned is displayed straight below the sub-window as soon as you place the mouse pointer on its name in the actual project or the reference project respectively.

In addition, each line containing a coloured entry is highlighted in grey.

Below the sub-window you find further lines displaying the options set for the comparing as well as a short statistic showing the number of additions, deletions and changes.

Working in the compare mode

The toolbar on top of the survey provides a set of functions to edit the comparison:

You may achieve detailed information on discrepancies of units marked in bold red either by a double click on their name or by a click on in top of the projects survey. After that the survey will be superposed by a **detailed view** showing the contents of the two versions opposed to each other, and the varying elements are listed in colours according to the scheme presented above.
In case of a POU its appearance in the associated text or graphic editor will be displayed at left for the actual project, respectively at right for the reference project. The smallest entities compared to each other are lines (ST, IL), networks (FBD, LD) or elements (CFC, SFC). In case of a device the variable entities like address, parameters or the mapping are listed in a table. As described above the left part of the table shows the current project and the right part shows the entities of the reference device. Note that you cannot modify by hand any of the two displayed versions within this comparison, rather accept the reference version instead of the actual one!

Press in the top left-hand corner of the window to return to the survey.

By a click on the cursor jumps to the next unit of the device tree, which shows differences.

By a click on the cursor jumps to the precedent unit showing differences.

By use of Accept Block the version of a block contained in the reference project will be accepted for the actual project as well. The direction of the acceptance goes from the reference project to the actual project, not vice versa.
In the detailed view (see above, first item of this list) a block consists of the entity (line, network, element) the cursor is actually placed on as well as of all cohering entities, which have the same sort of difference marking (for example subsequent lines). After accepting a block within the detailed comparison view the block will appear highlighted yellow in the actual version.
In the survey view however, a block consists of the unit the cursor is actually placed on as well as of all subordinated units. Having accepted a changed block within the survey view will highlight the associated units of the actual project in yellow and add a check mark to them. Accepting a block newly inserted in the actual version will delete it (marked by a small red cross in addition), whereas accepting a block deleted in the actual version will reinsert it into the actual version (marked by a small green plus in addition). Single parts of a block can again be inserted respectively deleted by use of the command Accept Single. By entering a unit hereunto you will be asked if you want to commit these changes to the detailed view.
A repeated click on Accept Block cancels the changes made during its last execution.

By use of Accept Single the unit (survey view) or entity (detailed view, see above, first item of this list) the cursor is actually placed on will be accepted in the version contained in the reference project for the actual project also. The direction of the acceptance goes from the reference project to the actual project, not vice versa.
After accepting an entity within the detailed comparison view the entity will appear highlighted yellow in the actual version. When returning to the survey view you are asked if you want to commit the changes made. If the two versions are identical hereafter, they will now be displayed in regular black in the survey.
However, having accepted a single unit within the survey view will highlight in yellow its name in the actual project as well as the name of all other units the specified unit is depending on. In addition a check mark will appear subsequent to their names.
Accepting a unit newly inserted unit in the actual version will delete itself and all subordinated units and mark the corresponding lines by a small red cross in addition. Though, accepting a unit deleted in the actual version will reinsert itself and all other units the specified one is depending on into the actual version and mark their names by a small green plus in addition. By entering a unit (see above, first item of this list) you will be asked if you want to commit these changes to the detailed view.
A repeated click on Accept Single cancels the changes made during its last execution.

With the folder, access rights or object properties for the unit the cursor is actually placed on will be accepted for the actual project such as they are set in the reference version. The direction of the acceptance goes from the reference project to the actual project, not vice versa. Your assortment of folder, access rights or object properties can be done in the arising dialog box. Afterwards the associated unit will be highlighted in yellow and there will check marks be added to the icons representing folder, access rights or object properties.

The last icon of the toolbar can only be executed after entering the detailed view of units marked in bold red by use of . By default, differing entities (lines, networks, elements) are displayed in red and opposed to each other. You may change this notation by a click on : the differing entities will now appear as inserted (bold green) in the actual project, respectively deleted (bold blue) in the reference project. Remark that a change will now been counted as addition and deletion in the statistics. Executing again recovers the default outlook

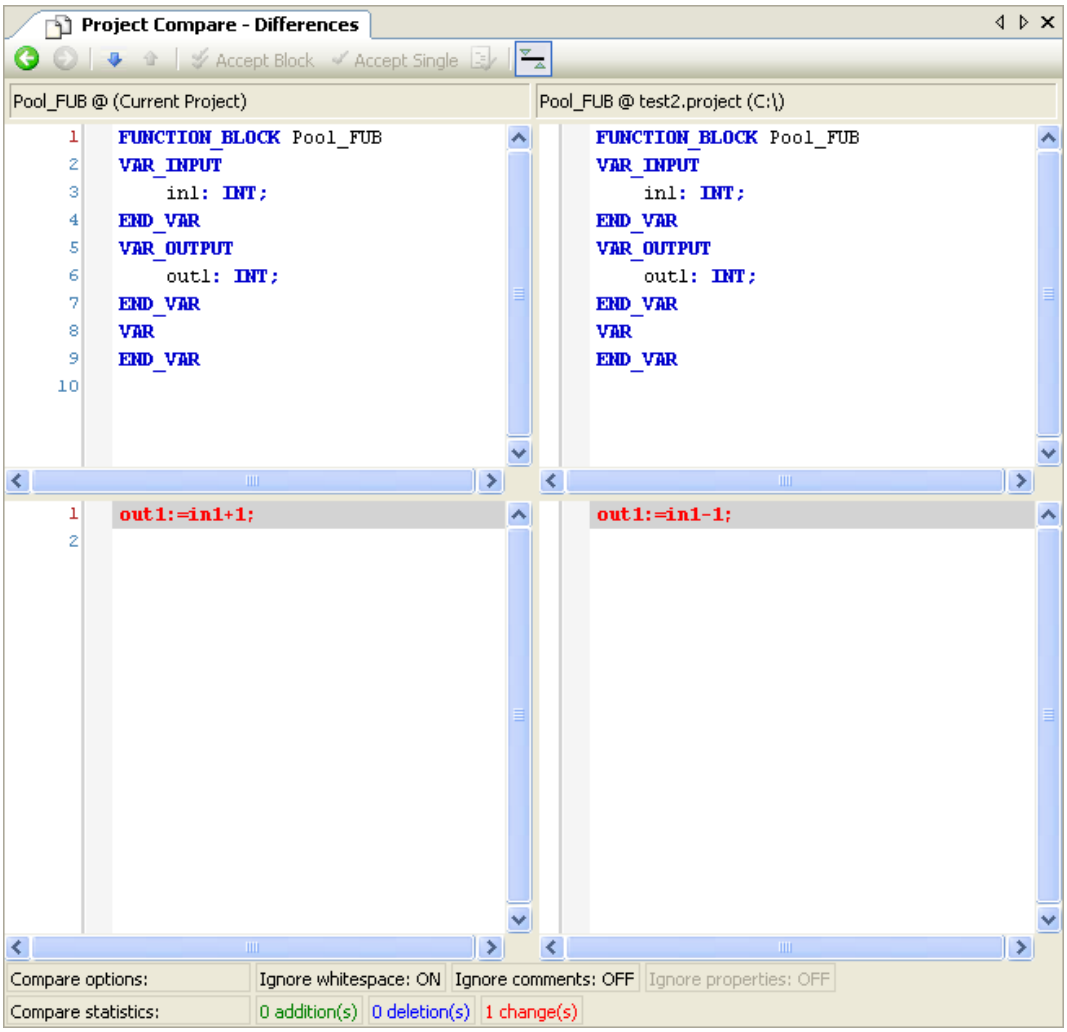


Abb. 148 Example of detailed comparison result of POU (ST) in default view

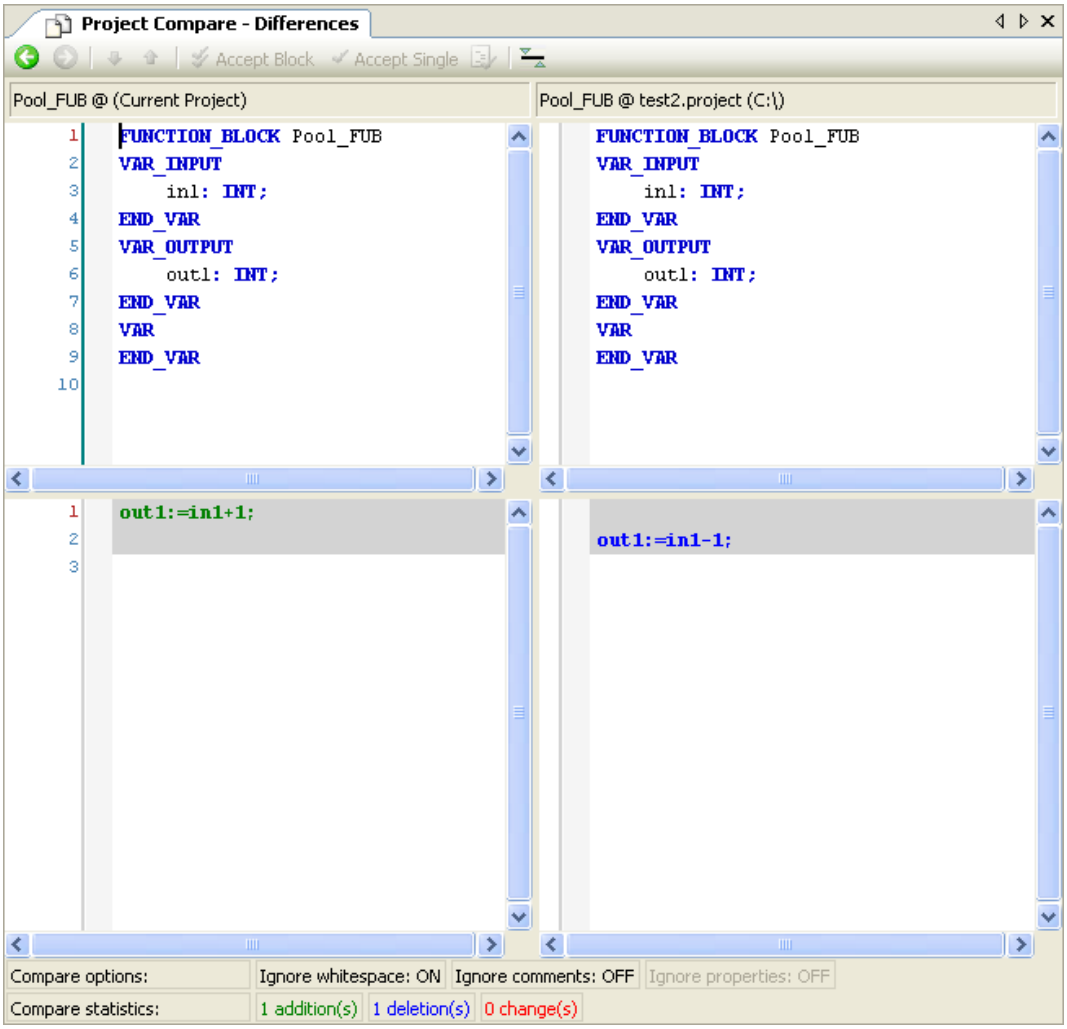


Abb. 149 Example of detailed comparison result of POU (ST) in non-opposed view

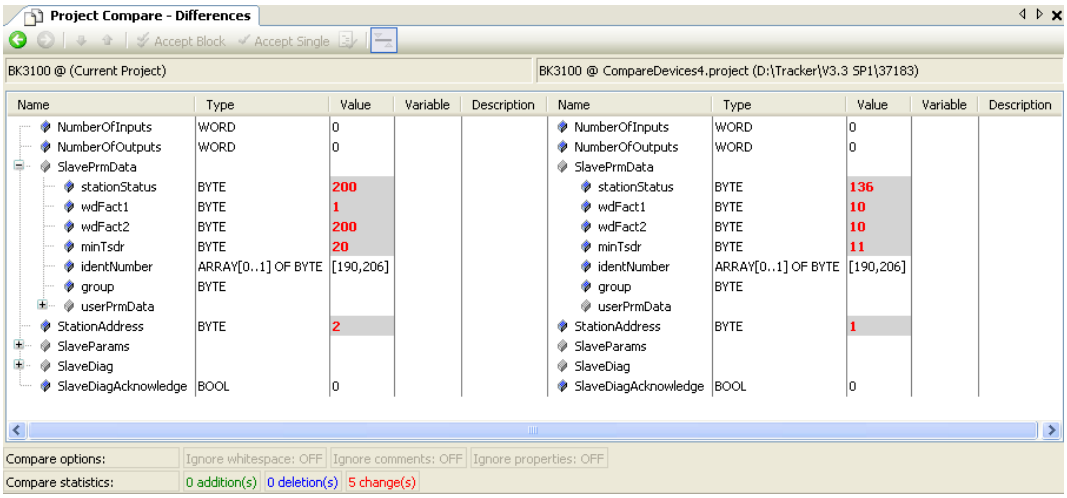


Abb. 150 Example of detailed comparison result of Device

Project Environment

This command (category 'File') opens the **Project Environment** dialog for working on the version compatibility of the project.

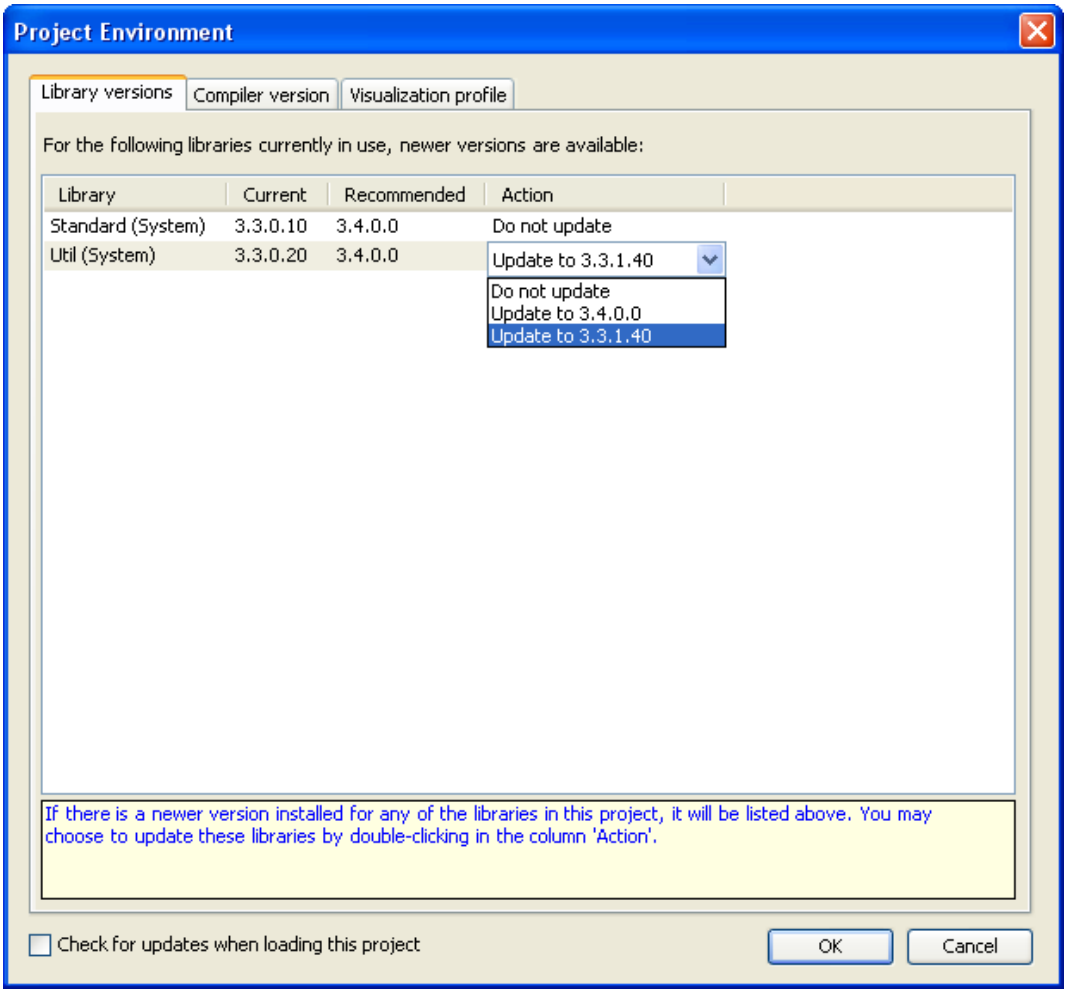


Abb. 151 Project Environment dialog, example

This dialog also automatically appears when you open a project which has been stored with different library or compiler versions resp. visualization profile. If option **Check for updates when loading this project** is activated, this will happen each time this project is opened, if not, the dialog only will appear at the first time.

See the following subdialogs:

Library versions

All libraries which have been included in the project manually (implicitly included ones are indicated by grey entries in the library manager) and which do not have the latest version currently available on the system, will be listed with their **Current** and **Recommended** version number. Set the desired update action (default: do not update) from the selection list, which opens on a doubleclick on the **Action** field.

Compiler version

The **Current** and **Recommended** version are displayed and you can select the desired update action (default:: do not update) from the selection list, which opens on a doubleclick on the **Action** field.

Note: If you load a project created with an older version of the programming system, for which the compiler version is "newest" (Project Settings), while here in the new programming system the Project Environment setting for the compiler version is "Do not update", then the version most recently used in the old project will be used (not the "newest" in the new environment).

Visualization profile

The **Current** and **Recommended** version are displayed and you can select the desired update action (default:: do not update) from the selection list, which opens on a doubleclick on the **Action** field.

OK initiates the currently defined update actions.

Project Information

Symbol:

This command (category 'File') opens the 'Project Information' dialog, where you can view resp. define properties and information on the project file, e.g access attributes, version number, author and company information as well as Statistics concerning the project objects. Notice the possibility of external access on project information data via property keys and automatically generated functions.

Four tabs are available for the information categories:

1. File

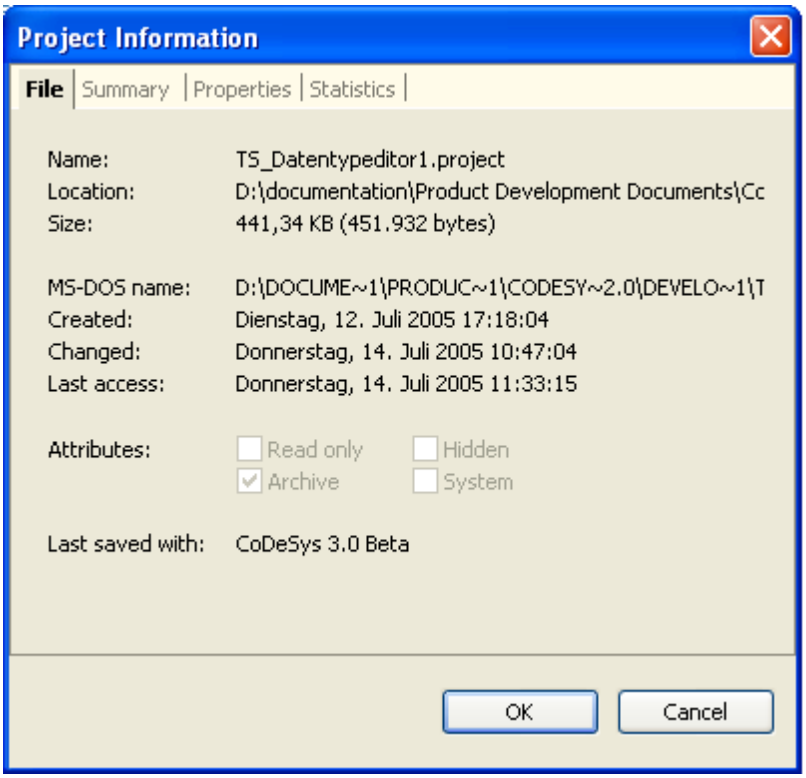


Abb. 152 Project Information dialog, File

Here the following properties of the project file are displayed: Name, Location, Size in Kbytes and bytes, MS-DOS name, Creation date, Date of Last access, Date of Last change, Name of the profile the project was Last saved with.

Further on the currently set file attributes are shown: Read only, Hidden (by default not visible in Explorer), Archive (ready for archiving), System (system file), which by default are not editable in this dialog (see file properties in the Windows Explorer)..

2. Summary

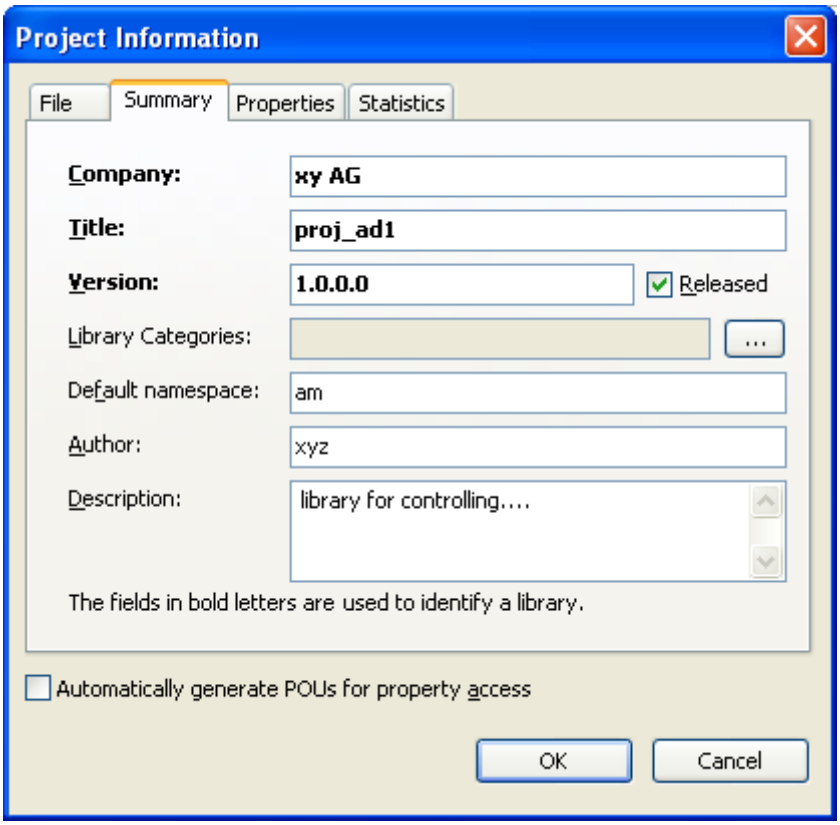


Abb. 153 Project Information dialog, Summary

Here you can optionally add some information on the project file like a **Title**, a **Version**, the **Default Namespace**, the **Author**, the **Company** name and any **Description** text. These information will automatically be displayed as available "keys" in the Properties tab of the Project Information dialog (see below).

Notice for library projects: If a project is intended to be used as a library in other projects, you must enter a **Title**, a **Version** number and the **Company** name here. Any library provided with these project information can be installed on the system and get included in a project. The company name besides the category serves for sorting in the Library Repository dialog.

Optionally also a **Default namespace**, the author's name and a short description can be specified in order to get saved as library project info. If no default namespace is defined here, automatically the name of the library project will be valid as namespace.

Assigning **Library categories**:

The assignment of a library category later serves for **sorting** the available libraries in the Library Repository dialog. If no category is specified explicitly, the library will be added to category "Miscellaneous". All special categories must be defined in a **category description file** in XML format. One or several category description files can exist for this purpose. You can call such a description file in order to choose the desired category for the local library project, or - alternatively - you can call another library project, which itself already has included the information from a category description file.

Via button ... open the dialog 'Library categories', which shows the currently assigned categories.

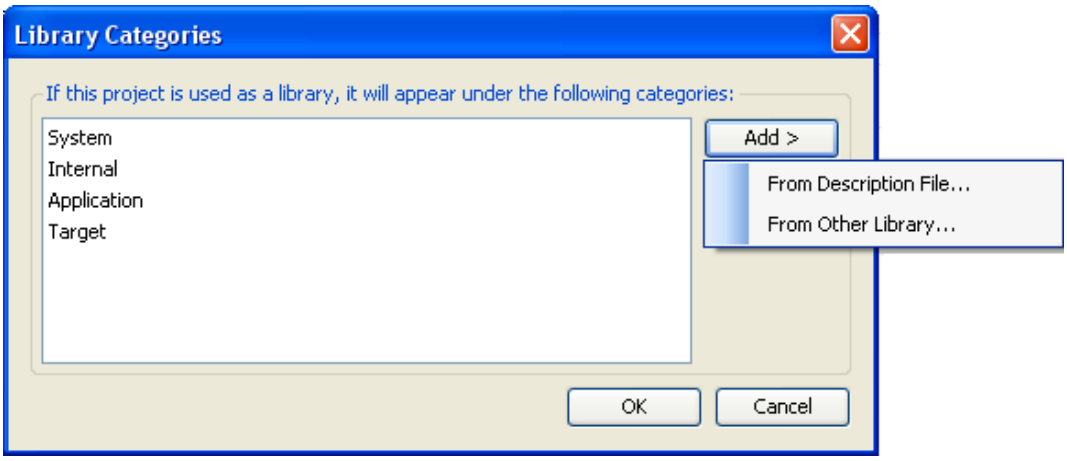


Abb. 154 Library Categories dialog

In order to assign the current library project to one or several (further) categories, use button **Add >** and select one of the options:

From Description File: The standard browse-dialog for selecting a file will open where you can search for the desired description file *.libcat.xml.

From Other Library ...: The standard browse-dialog for selecting a file will open where you can search for a library already containing category information, *.library.

The categories read from the description or library file will be listed now. Remove those you do not need by button **Remove**. Further categories might be added in the same way and finally you confirm with **OK** to get the dialog closed and to get the categories entered in the 'Library Categories' field in the Project Information dialog.

3. Properties

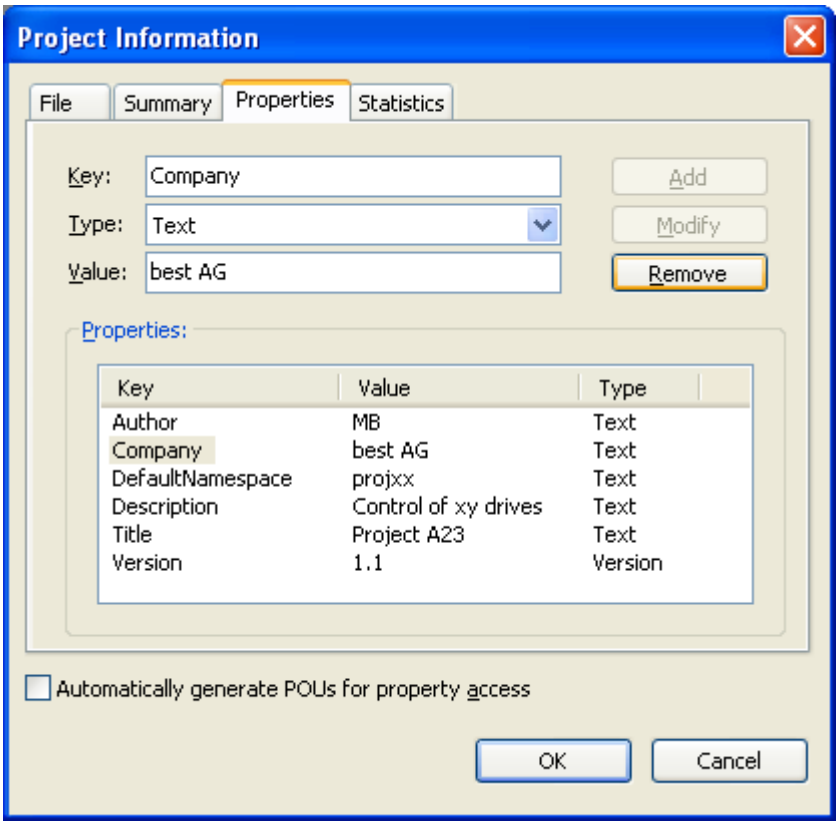


Abb. 155 Project Information dialog, Properties

Here you can define keys for some project properties. These later can be used in customer specific external programs for the purpose of controlling the respective property.

At least the information defined in the Summary tab of the dialog will be visible as "Keys" in the properties table. The properties' names are used as key names, the data type automatically will be "Text" and the "Value"s will be the text strings as defined in the Summary tab. Any further keys can be added as desired, see below.

Add a key:

Enter a key name in the **Key edit field**, choose the desired data type from the selection list at **Type:** (possible types: Text, Date, Number, Boolean, Version) and in the **Value: edit field** enter the desired value, which must fit to the chosen data type. Press button **Add** to add the new key to the **Properties table.**

Modify a key:

Select the entry in column 'Key' of the Properties table, then edit the entries in the edit fields above the table and press button **Modify** to update the entry in the Properties table and for the respective keys **also in the Summary tab**!

Remove a key:

Select the entry in column 'Key' of the Properties table and press button **Remove**.

Automatically generate POUs for property access: If this option is activated, automatically function POUs will be created in the POUs window, which can be used to access the project properties values in the application program. Special functions will be created in this case for the properties 'Company', 'Title' and 'Version' (GetCompany, GetTitle, GetVersion). For accessing additionally defined properties, a respective function for each property type (GetTextProperty, GetBooleanProperty, GetNumberProperty, GetVersionProperty) will be available. In this case call the appropriate function, pass the property key (as defined in the properties tab) as an input, and you will get returned the property value.

Example: The following property is defined in the Properties tab: Key = nProp1, Type = Number, Value = 333. To get the value in the application program, call function GetNumberProperty, for example showprop:=GetNumberProperty("nProp1"); showprop must be declared as type DINT in this case.

4. Statistics:

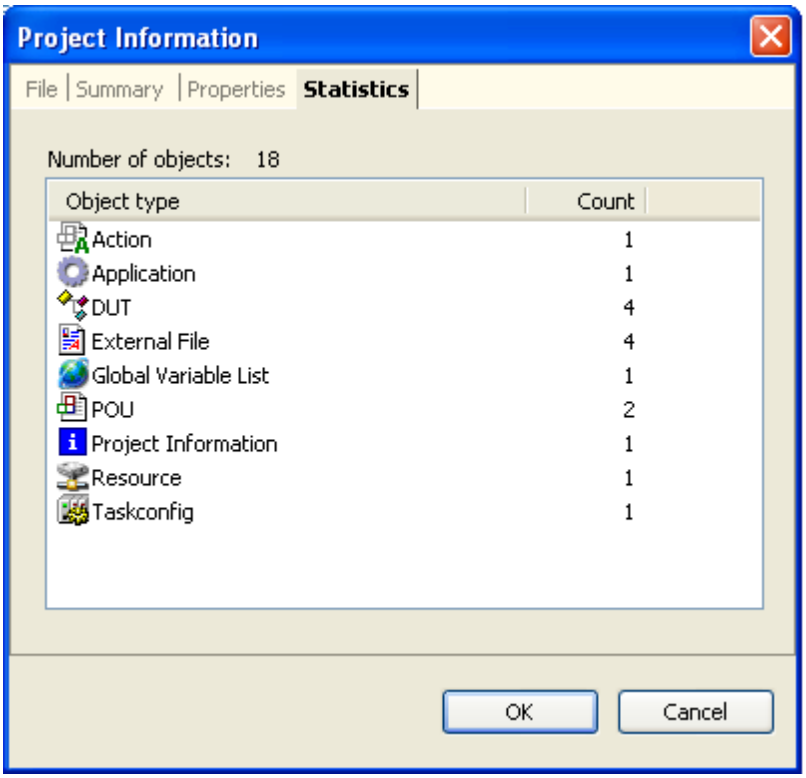


Abb. 156 Project Information dialog, Statistics

The table shows an overview on the objects used in the project: See the total **Number of objects** on top as well as in the table below the number of objects (**Count**) per **Object type**.

Project Settings

Symbol:

This command (category 'File') by default is not available in a menu, but in each new project automatically the object 'Project Settings' is available in the POUs tree. By a double-click on that object, which corresponds to executing the command, the 'Project Settings' dialog can be opened.

The dialog provides subdialogs for various settings like for example project encryption, user and access rights management, version handling, layout definitions for printouts etc.

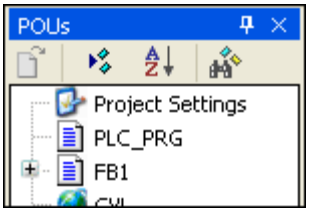


Abb. 157 Project Settings object in POUs window

On the left side of the dialog the categories of possible settings are listed. On the right side the corresponding dialog will appear.

Category Compile options

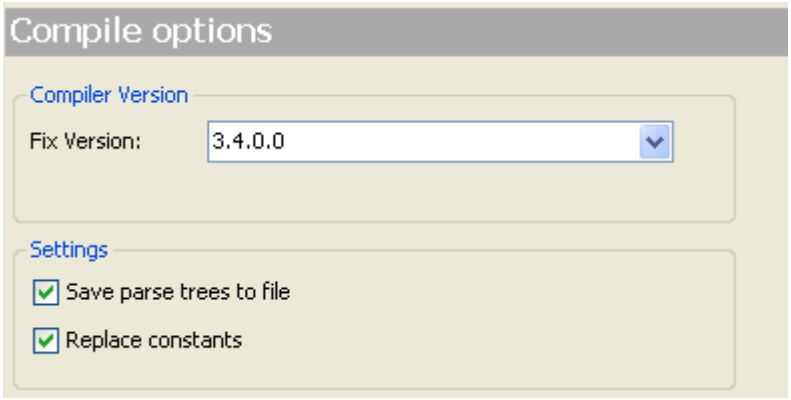


Abb. 158 Project Settings dialog, category Compile options

Compiler Version:

Here you can define which compiler version should be used when the project gets compiled for example by one of the build commands or during download. For this purpose choose the desired version from the **Fix version** selection field.

Note: Whether the data type LREAL will be treated as 64 bit type or converted to REAL is determined by the particular target device. Please see the related documentation for further information.

Settings:

Save parse trees to file: In case of large projects or low-performant systems it might be reasonable to activate this option. It helps to avoid out-of-memory problems by storing the precompile (language model) information of the project in a temporary file and not in the main memory. However this will increase the project loading and compilation time.

Replace constants: Per default this option is activated, which means: For each constant of scalar type (thus not for strings, arrays, structures) directly its value gets loaded. In online mode the constants are indicated by a icon preceding the value in the declaration or watch view. In this case any accessing of constants e.g. via an ADR operator or forcing and writing are not possible. If the option is deactivated, the constant can be accessed, however this will mean a longer processing time.

Category Page Setup

The dialogs for configuring the layout of a printout page are provided by the 'Print Engine' plug-in. For a description see Page Setup. The layout defined here is valid for printouts via the 'Print...' and 'Document...' commands.

Category Source Download

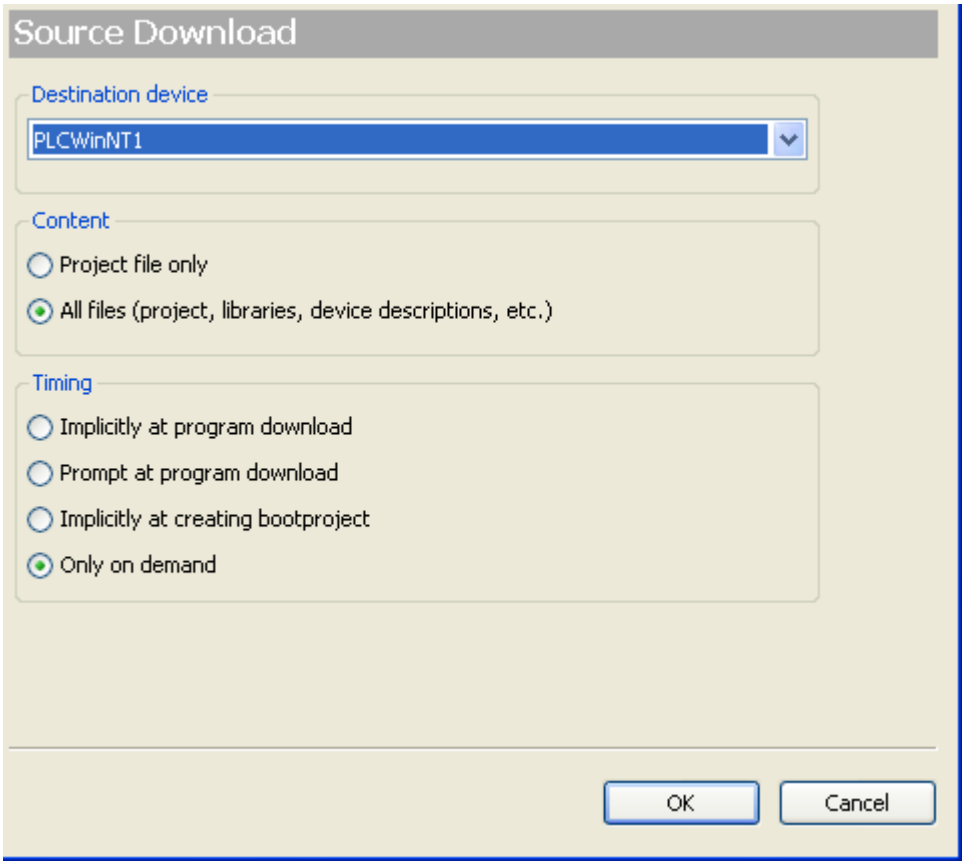


Abb. 159 Project Settings dialog, category Source Download

The following settings are valid for a source download, that is transferring the project file(s) to the PLC.

Destination device:

By default no device is predefined here. However you can select one from the selection list which offers all programmable devices defined in the project. If a destination device is defined here, you cannot perform a source download if you are currently connected to another device.

Content:

Predefines the selection of project-belonging files to be stored in the archive file for source download.

Project file only: only the <projectname>.project file

All files (project, libraries, device descriptions, etc.): Default

Timing:

Predefines when a source download will be done:

Implicitly at program download: The source download is executed at each program download (command Online \ Download '<application>') without any user interaction.

Prompt at program download: At each program download the user will be asked whether also a source download should be done.

Implicitly at creating bootproject: The source download is executed at each boot project creation (command Online \ Create boot application) without any user interaction.

Only on demand: The source download has to be initiated by command Source Download.

Category Security

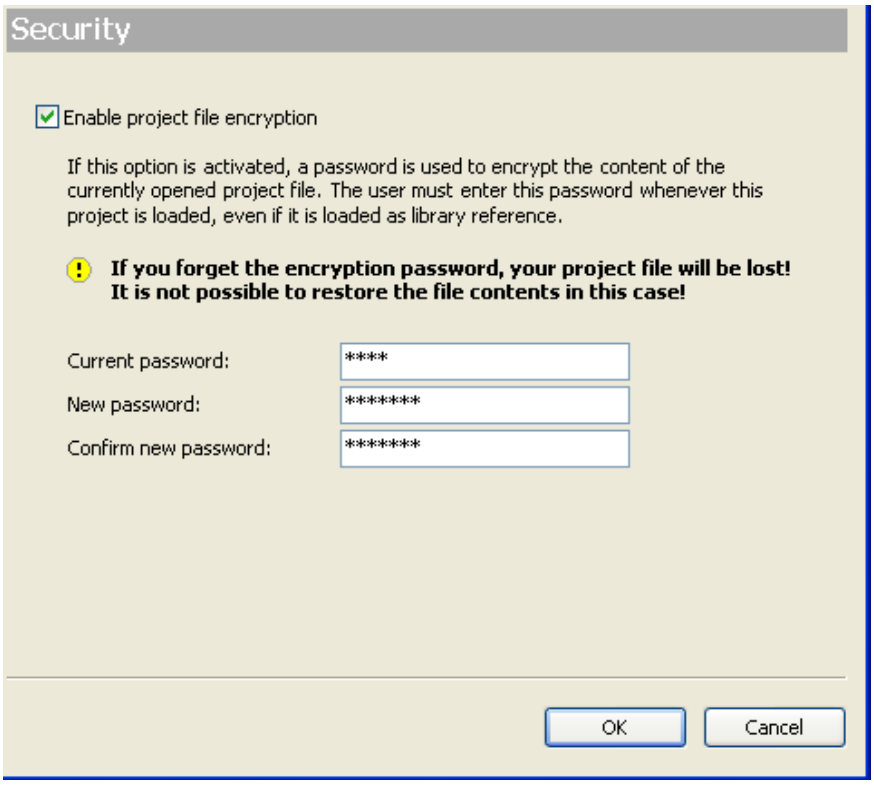


Abb. 160 Project Settings dialog, category Security

This dialog allows the use of a project password. Having activated the option **Enable project file encryption** the password has to be entered in the edit fields **New Password** and **Confirm password** (in order to detect type errors). If the checkbox is still activated while saving the project, this password will be required for a reload of the project, even if it will be loaded as a library reference. The Encryption Password dialog will open in this case.

Abb. 161 Encryption Password dialog

Note: If you do not remember the encryption password, the project will be lost! File contents cannot be restored in this case!

The project settings dialog may be reused for modifying the password. Therefor, in edit field **Current password**, you have to enter the actual password first, before you are allowed to change it by entering your new suggestion twice in the edit fields **New Password** and **Confirm password**.

Category Users and Groups

The dialogs for user and access right management are provided by the 'User Management UI' plug-in.

Category Visualization

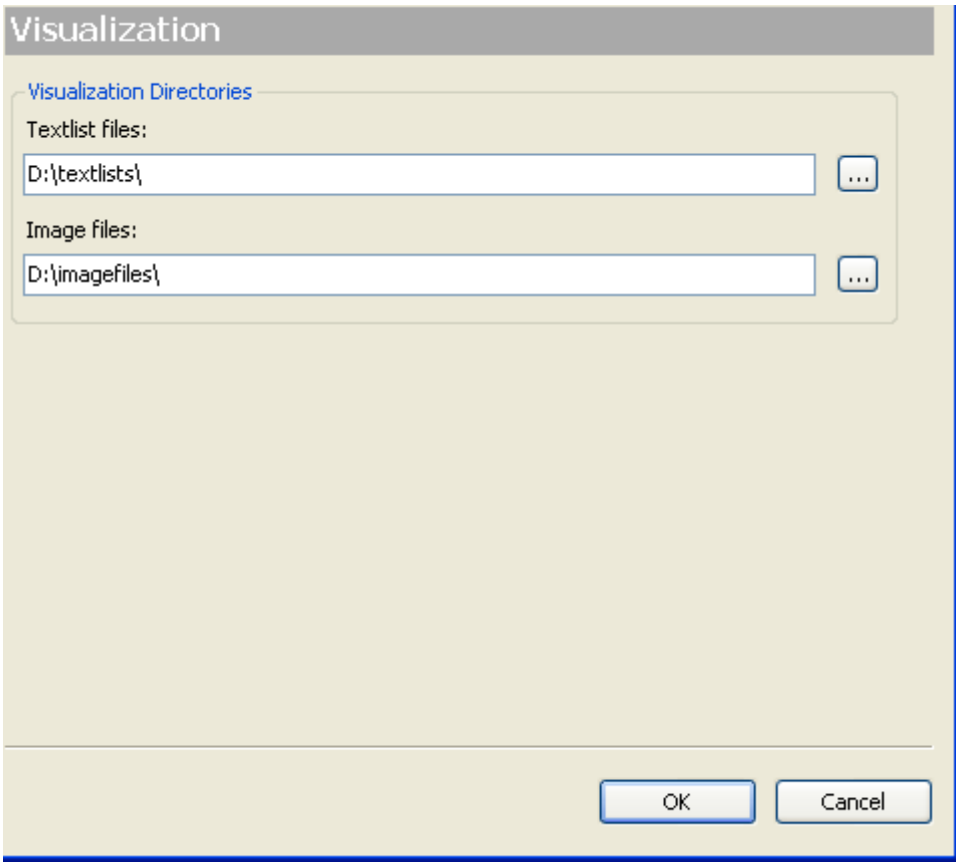


Abb. 162 Project Settings dialog, category Visualization

Visualization Directories: The default settings for the following directories are defined by the Project Options.

Textlist files: Use button ... to open the standard dialog for specifying an existing or new directory for language files (textlists), which should be available for text and language configuration within the visualization. The files found in this directory will be available for selection when a textlist should be imported, e.g. after having been edited by an external translator. Only one directory can be specified here.

Image files: Use button ... to open the standard dialog for specifying one or several existing or new directory for image files (image pools), which should be available for the use in the visualizations. If entering several directories, separate by semicolons.

Category Visualization Profile

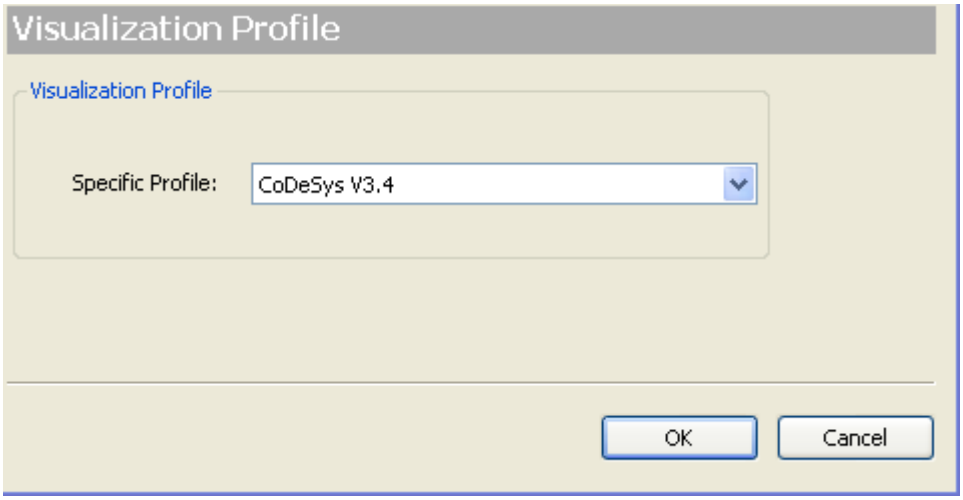


Abb. 163 Project Settings dialog, category Visualization Profile

Define here which of the visualization profiles currently available on your system should be used when the project gets opened:

Specific profile: The profile currently chosen from the selection list will be used.

If you open a project, which does not use the latest available visualization profile, you will be hinted at that and can decide to do an appropriate update action in the Project Environment dialog.

If the current settings should be used for any further projects created with the currently used programming system, activate option **Use this setting as default for new projects.**

12.15.2 Project Settings - Users and Groups

The Project Settings dialog in category 'Users and Groups' provides three subdialogs for the user management for the current project: Users, Groups, Settings.

Users dialog

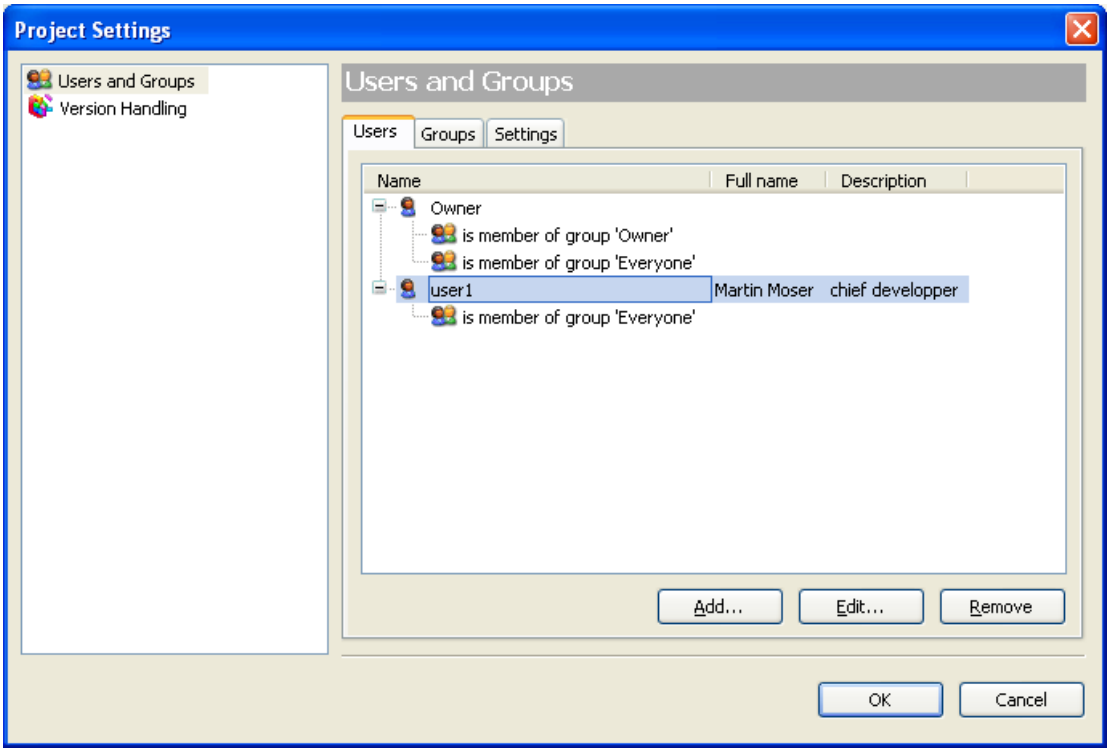


Abb. 164 Project Settings, Users dialog

The currently registered users are listed in a tree structure. If defined via the Add User or Edit User dialog (see below) besides the Name (log-on name) also the full name and a description for the user are displayed. The ownerships of each user can be viewed resp. hidden via the plus resp. minus sign. Each user per default is member of group "Everyone".

Define a new user account:

Use button **Add** to open the **Add User dialog**:

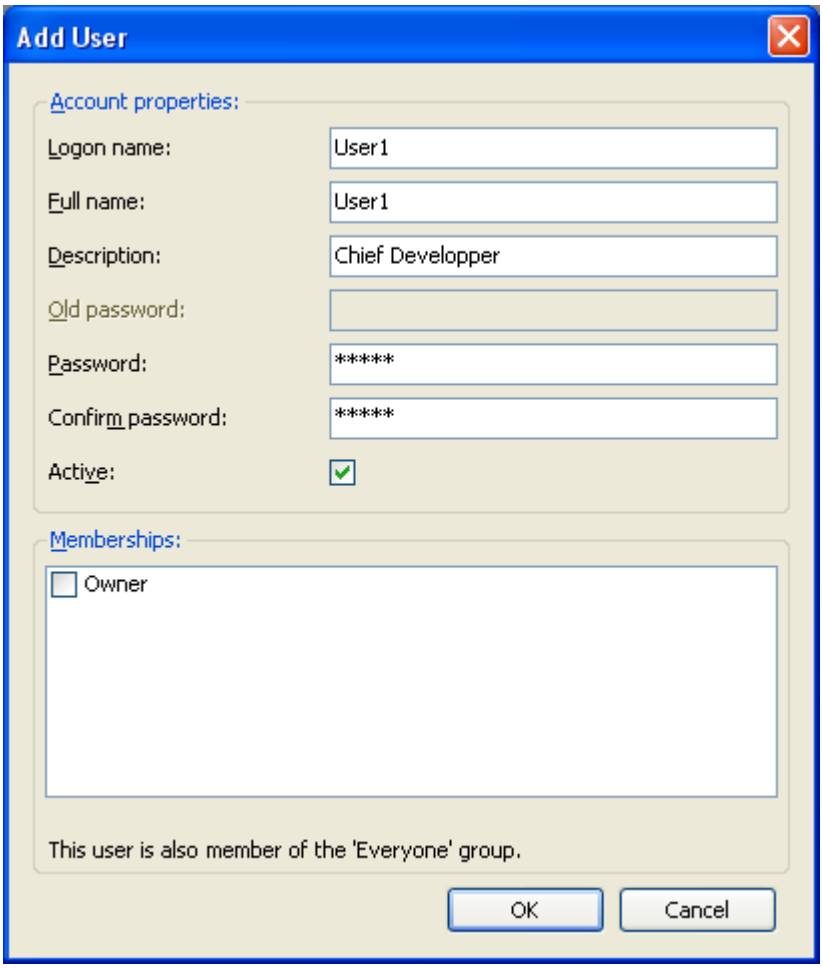


Abb. 165 Add User dialog

Fill in the following fields:

Account properties:

Login name: Log-on name for the new user.

Full name: Complete name of the new user. Just to give additional information.

Description: Description on the new user. Just to give additional information.

Old Password: This field is only editable, when the dialog is used for modifying an existing user account ("Edit User", see below). Before you can modify the password of an existing user, you must enter the currently valid password.

Password: Password for the new user. The entry is masked by * characters.

Confirm password: The entry made in field 'Password' must be repeated here and you will get an error message if the two entries do not match. Also here the entry is masked by * characters.

Active: If this option is activated the user account is valid. If the account is not valid, the user cannot log-on. An account might be deactivated automatically if repeatedly a log-on has been tried with incorrect authentication entries.

Memberships: In this list all currently existing user groups are listed, beside group "Everyone", to which the new user belongs automatically. By selecting the respective entries (☑) you can define to which groups the new user should belong.

To set up the new user close the dialog with **OK**. If there are incorrect entries (no login name, password mismatch, user already existing) you will get an appropriate error message.

Modify an existing user account:

Use button **Edit** to open the **Edit User dialog**. The entry fields are the same as in the 'Add User' dialog (see above). The password fields however - for safety reasons - will show 32 * characters. After having modified the desired entries close the dialog with **OK** to get applied the new settings.

Remove one or several user accounts:

Select the respective users in the Users list and use button **Remove**. Note that you will get no further inquiry! An appropriate error message will appear if you try to delete all users from the group. At least one must remain.

Groups dialog

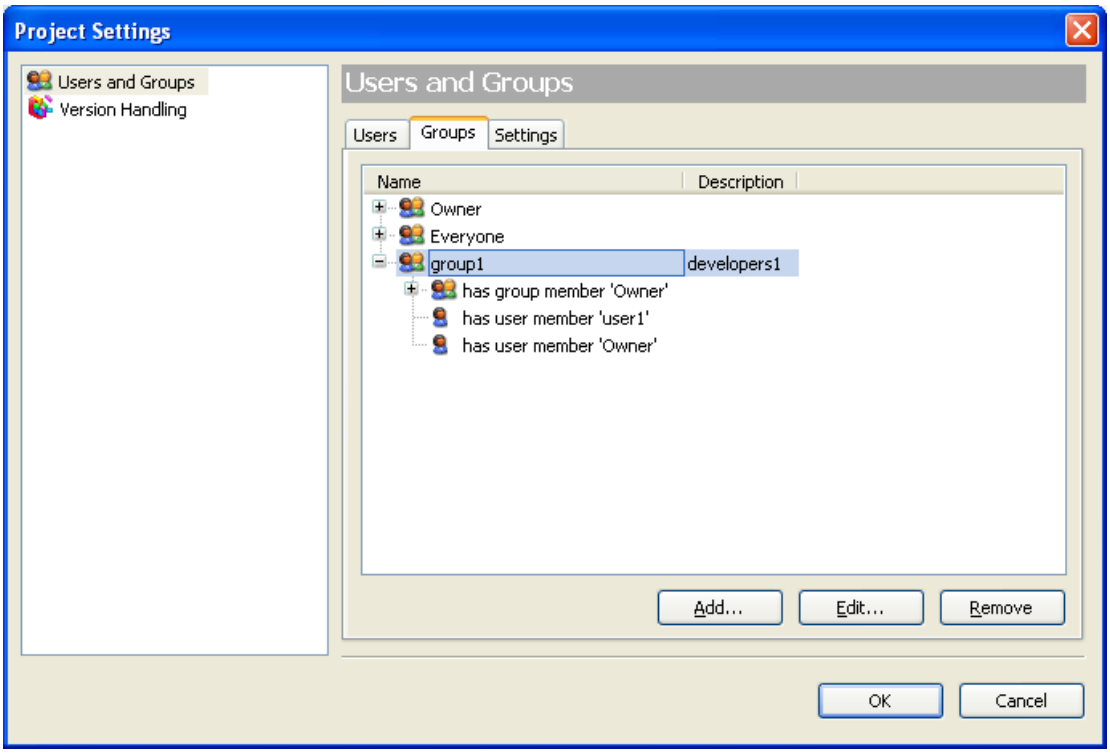


Abb. 166 Project Settings, Groups dialog

The currently available groups are displayed in a tree structure. The members of each group can be viewed resp. hidden via he plus resp. minus sign. A member again also might be a group.

Add a new group:

Use button **Add** to open the **Add Group dialog**:

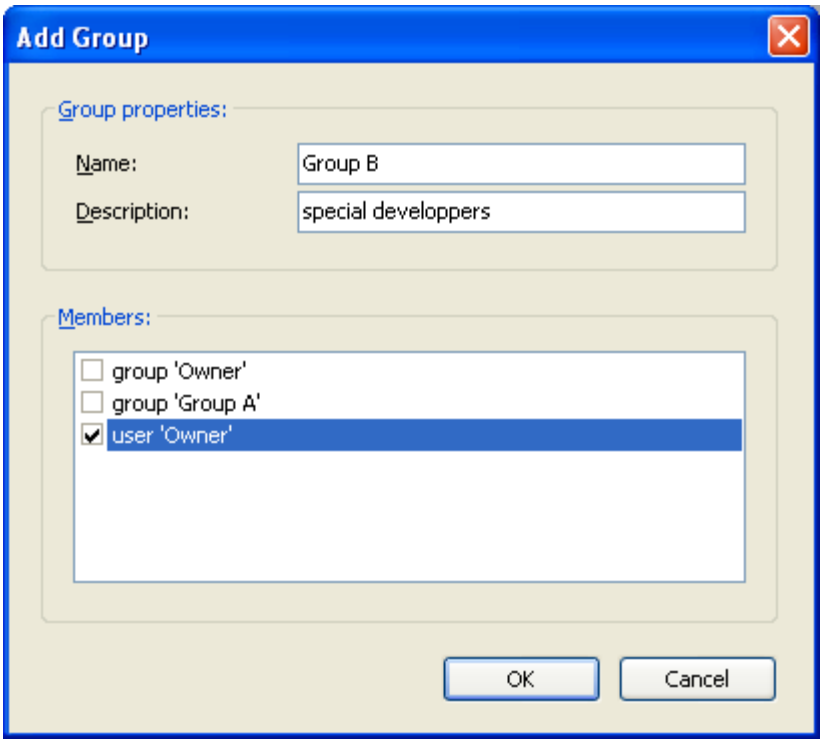


Abb. 167 Add Group dialog

Fill in the following fields:

Group properties:

Name: Name for the new group.

Description: Description on the new group. Just to give additional information.

Members: List of all currently available users and groups. Select those ones (☑) which should be members of the current group.

To set up the new group close the dialog with **OK**. If there are incorrect entries (no name defined, group already existing, in Members having selected a group which would cause a "group cycle", you will get an appropriate error message.

Modify an existing group:

Use button **Edit** to open the **Edit Group dialog**. The entry fields are the same as in the 'Add Group' dialog (see above). After having modified the desired entries close the dialog with **OK** to get applied the new settings.

Remove one or several groups:

Select the respective entries in the groups tree and use button **Remove**. Note that you will get no further inquiry! The members of the deleted groups will remain unmodified. An appropriate error message will appear if you try to delete the groups "Everyone" and/or "Owner".

Settings dialog

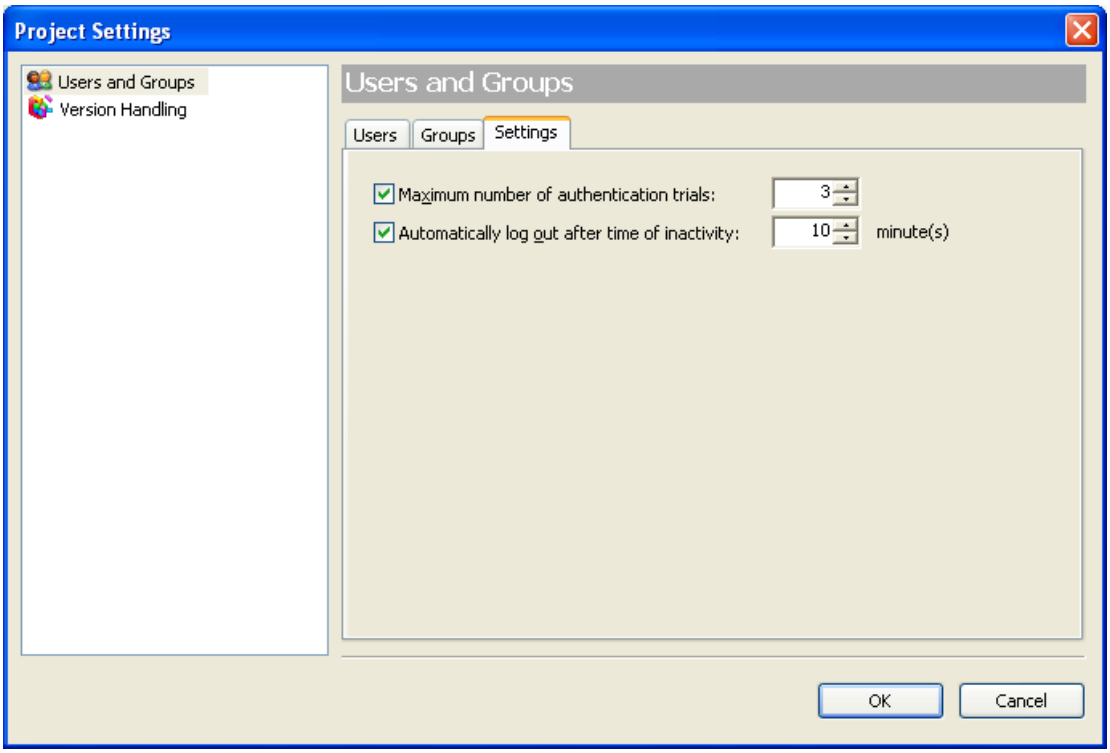


Abb. 168 Project Settings, Users and Groups Settings dialog

The following basic options and settings concerning the user accounts can be made:

Maximum number of authentication trials: If activated, the user account will be set invalid after the specified number of trials to log in with a wrong password. If not activated, the number of erroneous trials is unlimited. Default: option activated, number of trials:3; permissible values: 1-10.

Automatically log out after time of inactivity: If activated, the user account will be logged out automatically after the specified number of minutes of inactivity (no user actions via mouse or keyboard registered in the programming system). Default: option activated, time: 10 minutes; permissible time values: 1-180 minutes).

12.16 Help

The command category 'Help' provides functions for getting information, links and online help on »PLC Designer«.

12.16.1 Help commands - Online Help

The help commands provided by the 'Online Help' plug-in belong to command category 'Help'.

By default they are part of the 'Help' menu. If necessary, open the Customize dialog, category Menu, to view or modify the current menu configuration.

The commands:

- Contents...
- Index...
- Search...
- Sync with Table of Contents
- About...
- Further commands might be available, indicating links on an Internet page.

12.16.2 Contents...

Default Shortcut: <Shift>+<Ctrl>+<F1>

This command belonging to command category Help opens the Contents tree of the online help in a separate window.

The contents is structured via books which can be opened or closed via a single mouse-click on the plus/minus-sign. The particular pages can be displayed in a separate window by a single mouse-click on the page entry.

12.16.3 Index...

Default Shortcut: <Shift>+<Ctrl>+<F2>

This command belonging to command category Help opens a dialog showing a list of all index keywords provided by the online help in a separate window.

When you enter a string in the **Look for:** field the first keyword matching this string will be selected in the list. By a double-click on list entry either the respective help page will be opened, or - in case of multiple pages found for this keyword - a list of help pages will be displayed in the Index Results window.

12.16.4 Search...

This command belonging to command category Help opens a search dialog in a separate window. This dialog supports a full-text search over all pages of the online help. You can enter the string to be searched in the **Look for:** field.

The following search options can be set:

Search in titles only: The string will only be searched in the titles of the help pages.

Display partial matches: If the option is activated, the string will also be found if it is part of another string. Example: "log" will be also be detected in "dialog" whereas if the option is deactivated, "log" will only be detected if used in the help text as a separate string "log".

Limit to ... matches: The number of matches which should be detected and reported in the Search result window can be limited.

The list of help pages containing the searched string will be displayed in the Search Results window which will open automatically.

12.16.5 Synchronize Contents

This command belonging to command category Help will update the Contents view in a way that the currently displayed help page will be selected in the contents tree. If needed the respective book will be opened for this purpose.

12.16.6 About...

This command belonging to command category Help opens a box showing Version information on the currently used »PLC Designer« Programming System.

12.17 ImagePool

The command category 'Image Pool' provides commands for creating and configuring image pools.

12.17.1 Image Pool commands

The commands for working with image pools are provided with the 'Image pool' plug-in.

For customizing the menu structure use the Customize dialog.

The commands:

- Create GlobalImagePool
- Create Image File List

12.17.2 Create GlobalImagePool

Use this command (category Image Pool) to manually add the "GlobalImagePool" which is an image pool for static images used in a visualization. Notice that this pool automatically will be created as soon as the first static image is defined in a project.

12.17.3 Create Image File List

not yet implemented

12.18 Installation

This command category provides the commands for installing/unistalling managed libraries resp. for visualization elements.

12.18.1 Library Repository

Installation commands - Library Repository

The 'Library Manager Object' plug-in provides for command category 'Installation' the command 'Library Repository...' for handling library locations and installing and uninstalling of libraries.

By default this command is part of the 'Tools' menu. If necessary, open the Customize dialog to view resp. to modify the menu configuration (notice category 'Objects') .

For general information on the Library Management in »PLC Designer« please see:
Library Management**.**

Library Repository...

Symbol:

Note: It depends on the currently used Installation) opens the Library Repository dialog, which is also accessible via the Library Manager editor. A library repository is a database for libraries which have been installed on the local system in order to be available for getting included in »PLC Designer« projects.

Note: A library project *.library, which is stored in a library repository, cannot be opened out of there for editing or viewing in the programming system.

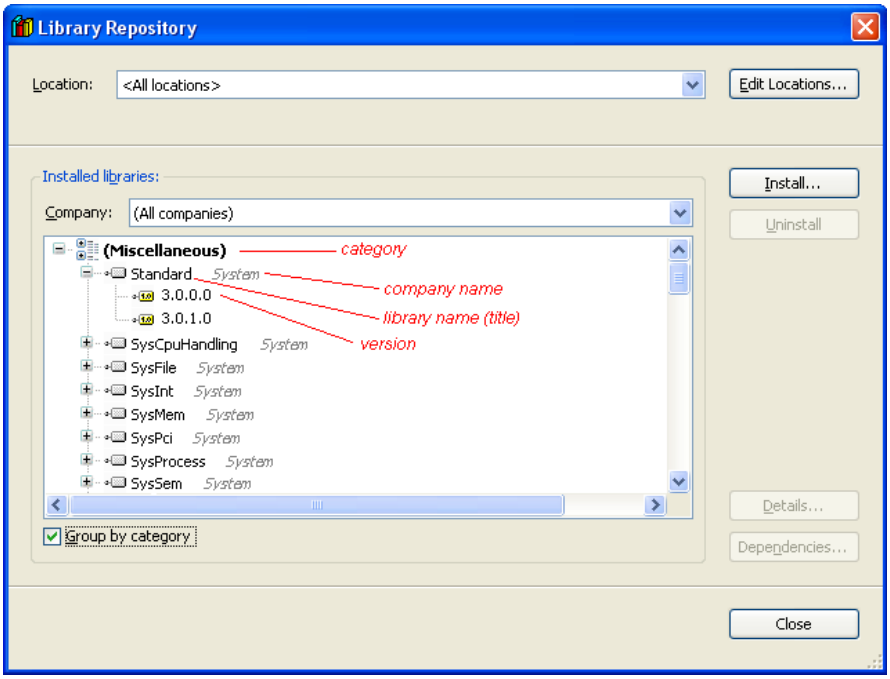


Abb. 169 Library Repository dialog

For general information on the Library Management in »PLC Designer« please see Library Management.

The dialog shows the currently defined library Locations (repositories) and the currently installed libraries. Repositories might be added, modified or removed in this dialog and libraries can be installed or uninstalled.

For the currently selected **Location**, which is a directory on the local system where library files are stored, and for the selected **Company** all currently **Installed libraries** will be displayed in a list. The list shows the libraries names (Title), Version number and Company name as provided by the Project Information of the library .

If option **Group by category** is activated, the listing will be sorted according to the library categories, whereby the category names are displayed as nodes which can be folded resp. unfolded to show/hide the libraries. If this option is deactivated, the libraries will be listed alphabetically. The library categories are defined by external description files *.libcat.xml.

According to the available buttons see in the following descriptions on...:

- Library Repositories (Edit Locations...)
- Library Installation and Uninstallation (Install..., Uninstall...)
- Further information on the library (Details..., Dependencies...)

Library locations (Repositories)

Several repositories can be used to manage libraries. All currently defined repositories are shown in the selection list behind Location:. By default in each case the "System" location is available as defined by the »PLC Designer« installation.

To edit the repositories path or name use button **Edit Locations...** to open the **Edit Repository Locations** dialog:

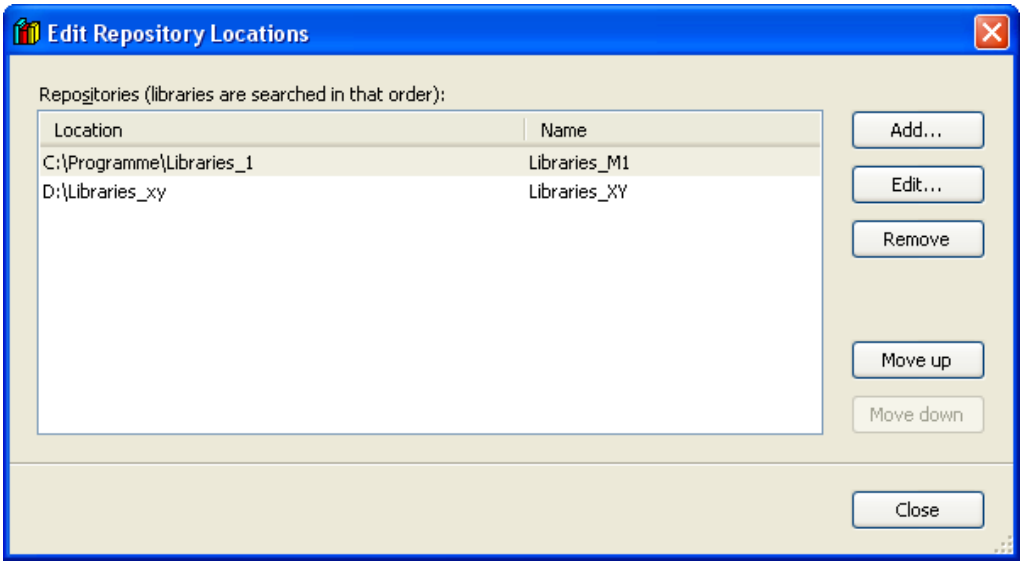


Abb. 170 Edit Repository Locations dialog

The currently defined locations are listed in the Repositories window. They will be searched later for a library in the given order from up to down. To modify this order use buttons **Move up** or **Move down** for moving the currently selected entry.

Notice that location "<all locations>" will display the libraries of all currently defined locations. In this view no installation is possible.

Defining a new repository resp. modifying path and name of an existing repository:

To add a new repository, use the **Add** button. The **Repository Locations** dialog will open where in the **Location** field you have to enter the path of the new repository. For this purpose you might use button **...** and browse for resp. define a new suitable folder. Notice that the chosen folder must be empty! In the **Name** field enter a symbolic name for the location for example "Libraries for System1".

To modify an existing repository, select the respective entry in the Edit Repository dialog and use button **Edit**. Also the Repository Locations dialog will open where you can edit the path and name for the repository.

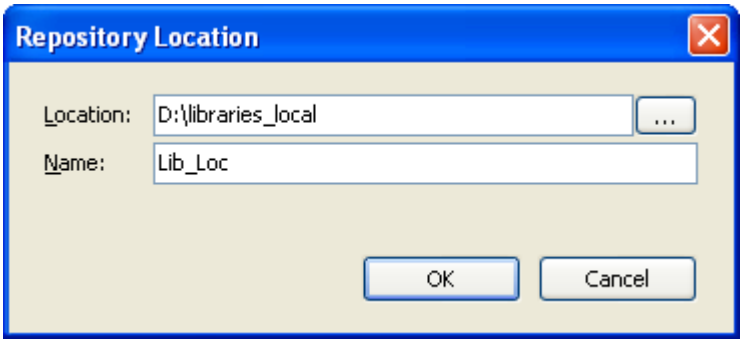


Abb. 171 Edit Repository Locations dialog, Add or Edit a repository

Note: Only empty folders can be defined to become a repository. The "System" repository is not editable. This is indicated by the entry being displayed in italic font.

Removing an existing repository:

When you select an entry in the repositories list and use button **Remove**, you will get asked, whether just the entry in the list should be deleted, or if also the folder containing the library files should be removed from the file system.

Library Installation and Uninstallation

Only a library installed on the local system (Library Repository) can be included in a project. As a precondition for the installation it must have got assigned a title, a version info and optionally a company name in its Project Information.

To install a library, in the Library Repository dialog select the repository to which the library should be added, and then press button **Install....**

The 'Select Library' dialog will be opened, which is a standard dialog for browsing for a file. By default the filter is set to "»PLC Designer« library files", which are identified by their Project Information properties and by default have extension "*.library". You might change the filter to "»PLC Designer« library files (before 3.x) to view library files having extension *.lib, or to 'All files'.

Select the desired library and close the dialog. The library will be added to the list of currently installed libraries in the Library Repository dialog.

If you selected a library which cannot be installed because it does not provide the obligate Project Information (Title, Version), an appropriate error message will appear.

To uninstall a library select it in the list of currently installed libraries in the Library Repository dialog and use button **Uninstall**.

Further Information on particular libraries

Details...: For the currently selected library - you must select the line containing the version! - detailed information will be provided in a dialog: Title, Version, Company, Size, Created (date), Change (date), Last access (date), Attributes. Title, Version and Company name come from the libraries Project Information Properties.

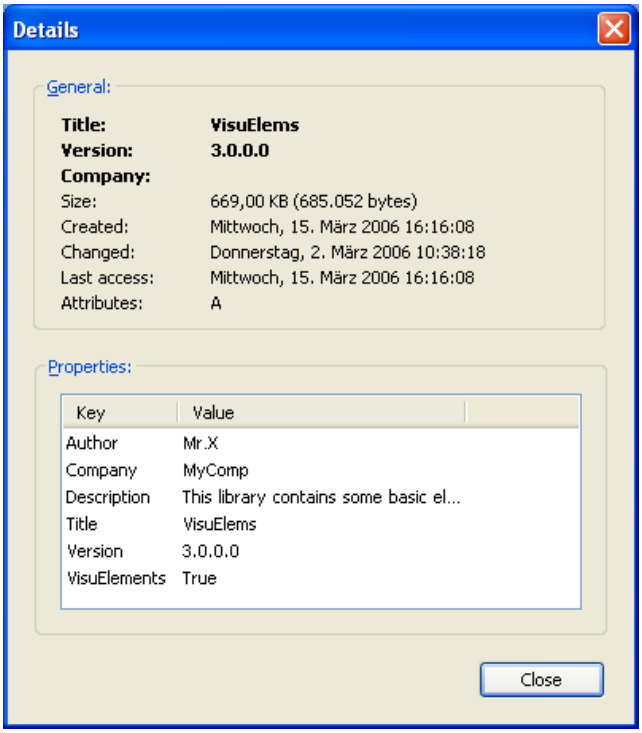


Abb. 172 Details dialog

Dependencies... For the currently selected library the dependencies concerning other libraries will be displayed, that is the libraries which are included in the library are displayed with Title, Version and Company information.

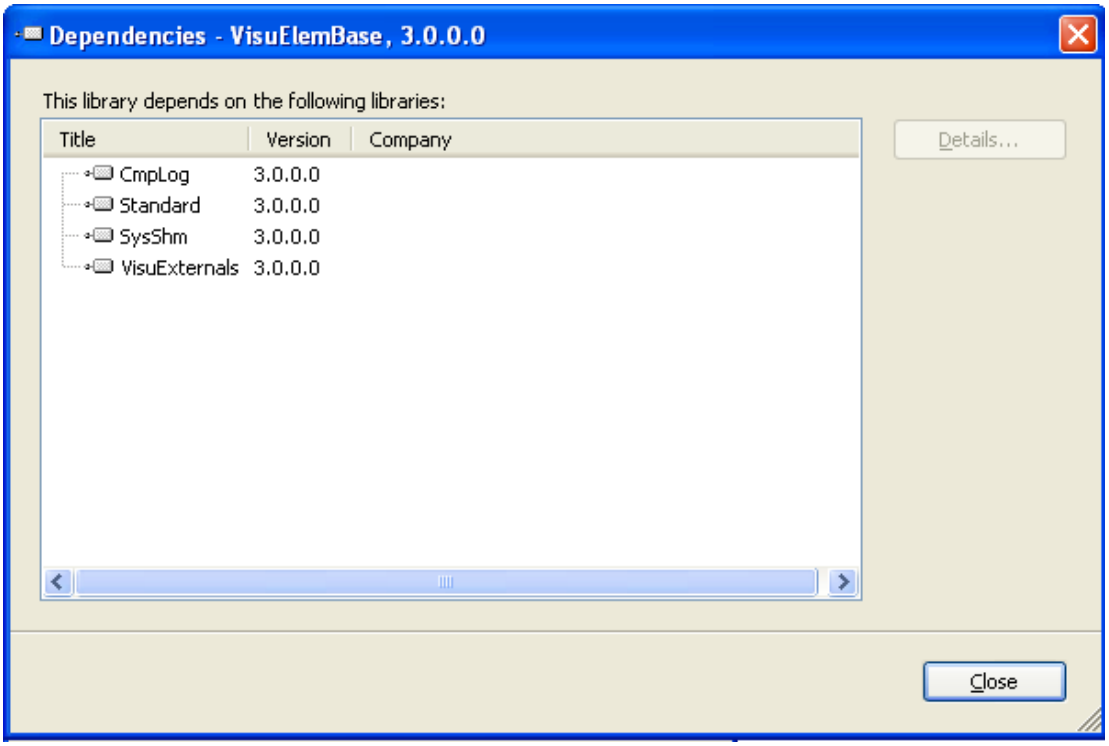


Abb. 173 Dependencies dialog

12.18.2 Visual Element Repository

Installation commands - Visual Element Repository

The 'VisualElement Repository' plug-in provides the command Visual Element Repository (category 'Installation') which is used to open the Element Repository for a visualization.

The command is of category 'Installation' and by default is part of the 'Tools' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (category "Installation").

Regard that the command is **only available when no project is opened**. This is because a defined set of the Visual Element Repository settings is managed as a "visualization profile" and visualization profiles primarily are available independently from a project.

Visual Element Repository

This command (category 'Installation') opens the **Visual Element Repository dialog**, where you can define element repositories and visualization profiles.

Regard that as soon as you leave the dialog via button **Close**, the current settings will be stored with the profile currently specified in the Profile Selection field.

A visualization profile defines which visualization elements (coming from which element library versions) will be available in the ToolBox when working in the Visualization Editor in a project basing on this profile. Each visualization project needs to use a visualization profile and it is defined in the Project Settings which one is to be used by default.

A profile can only be configured outside of a project, so the "Visual Element Repository" Editor is **only available when no project is opened**.

The Visual Element Repository dialog in its upper part allows to edit the repository locations. See below for a description of the Edit Repository Locations dialog. In the lower part it allows to edit the visualization profiles. See below for a description of how to edit visualization profiles.

Edit Repository Locations

This part of the Visual Element Repository dialog provides a selection list of the currently configured locations (directories on the local system) of element libraries. To edit or to add a location use button **Edit Locations...**. The **Edit Repository Locations** dialog will open:

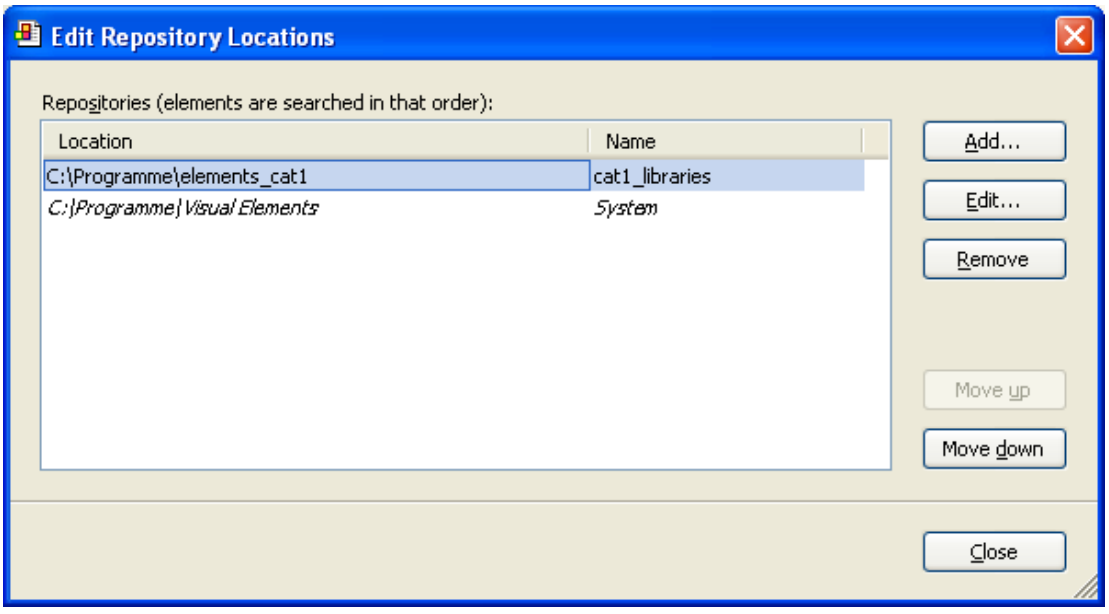


Abb. 174 Edit Repository dialog

One or multiple repositories can be used to manage the visualization elements. All currently defined repositories are listed here. Per default in each case the "System" location is available as defined by the »PLC Designer« installation.

The currently defined locations are listed in the Repositories window. They will be searched later for a library in the given order from up to down. To modify this order use buttons **Move** up or **Move down** for moving the currently selected entry.

Editing an existing or adding a repository:

To edit the repositories path or name, select the respective location in the list and use button **Edit** to open the **Repository Location** dialog.

The same dialog is used for adding a new repository. For this purpose use the **Add...** button.

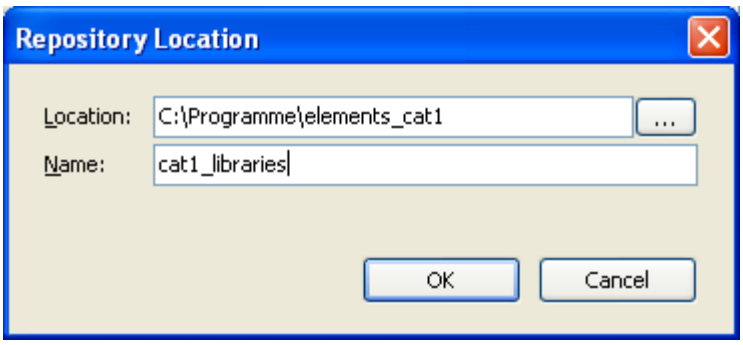


Abb. 175 Edit Repository Locations dialog, Add or Edit a repository

The following entries have to be made resp. modified:

In the **Location** field you have to enter the path of the new repository. For this purpose you might use button ... and browse for resp. define a new suitable folder. Regard that the chosen folder must be empty! In the **Name** field enter a symbolic name for the location e.g. "Elements Category 1".

Note: Only empty folders can be defined to become a repository. The "System" repository is not editable. This is indicated by the entry being displayed in italic font.

Removing an existing repository:

Use the **Remove** button to delete the currently selected repository.

Edit Visualization Profiles

This part of the Visual Element Repository dialog allows to configure visualization profiles. A visualization profile stores the current repository settings, that is a certain set of elements ("Installed Elements") coming from one or multiple libraries. This set of elements will be provided in the toolbox of the Visualization Editor when the respective profile is used in the project. For configuring a profile choose the desired elements from the list of generally "Available Elements" which is composed basing on the currently installed element libraries. You can edit an existing profile or create a new one.

Profile Selection

The **Profile** field contains a selection list providing the names of the currently available visualization profiles.

If you want to edit an existing profile select its name from the list and use the 'Available Elements' dialog to fill the 'Installed Elements' list as described below. The new configuration will be stored with the specified profile name as soon as you leave the dialog via button Close.

If you want to create a new profile, use **Add** or **Copy**. In each case the **Enter Visualization Profile Name** dialog for defining a new name will appear. Per default the previously used profile name will be entered automatically, extended by "_0" to get unique. If desired, enter another unique name:

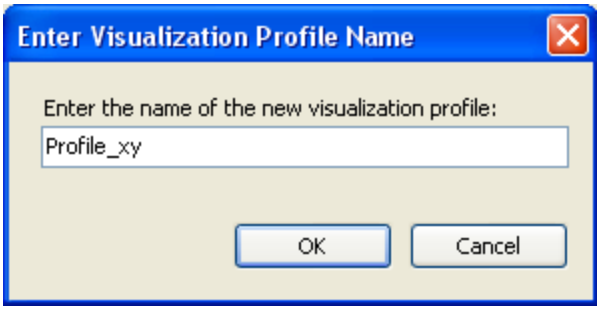


Abb. 176 Dialog 'Enter Visualization Profile Name'

If you add a new profile, the list of installed elements initially will be empty, if you copy an existing profile it will be filled with the respective elements.

Installed Elements

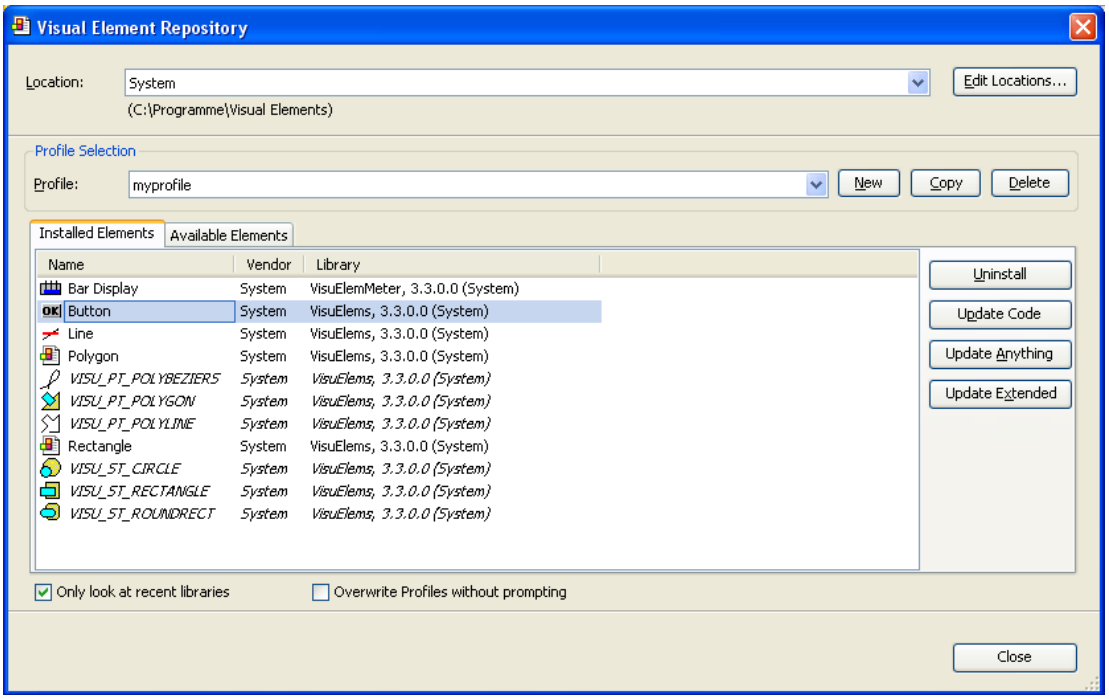


Abb. 177 Visual Element Repository dialog, Installed Elements

This list shows the elements currently assigned to the profile specified in the Profile Selection field. Each the name (**Name**), provider (**Vendor**) and the element **Library** are displayed.

If you want to browse only the latest version of each library for installed elements, activate the checkbox **Only look at recent libraries**. This will save operation time.

You can uninstall elements (that is remove them from the 'Available Elements' list) by selecting one or several entries and use button **Uninstall**.

The list can be updated with one of the following options, e.g. if any libraries have been changed:

Update Code: Use this button to update the list concering possible changes in the implementation of the libraries' modules.

Update Anything: Use this button to update the list not only concerning implementation changes but also concerning any changes in their interfaces (declaration parts).

Update Extended: Use this button to update the list with specific element parameters. For this purpose the dialog 'Extended Update' appears, where you can change e.g. the permissible number of points in polygons. The new setting will be valid for the current profile.

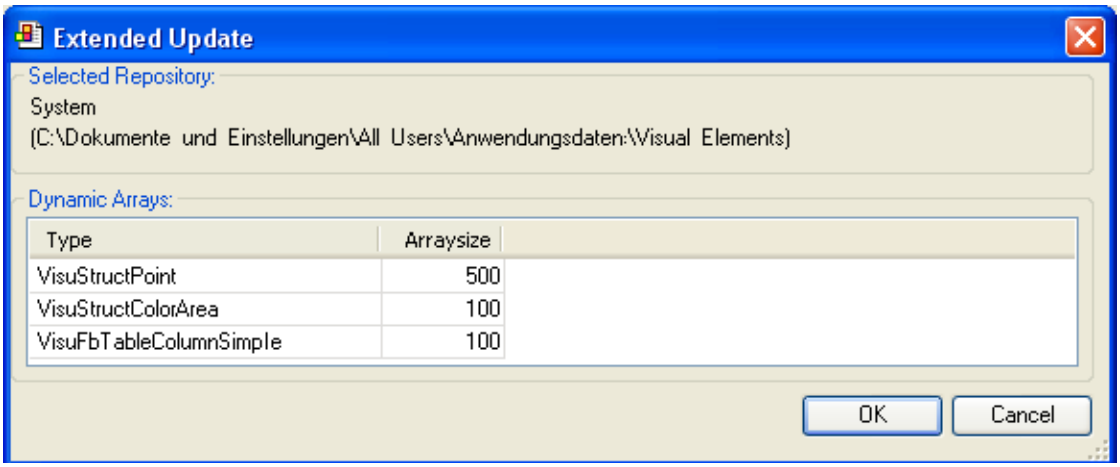


Abb. 178 Extended Update dialog, example

If you want to install further elements, open tab 'Available Elements':

Available Elements

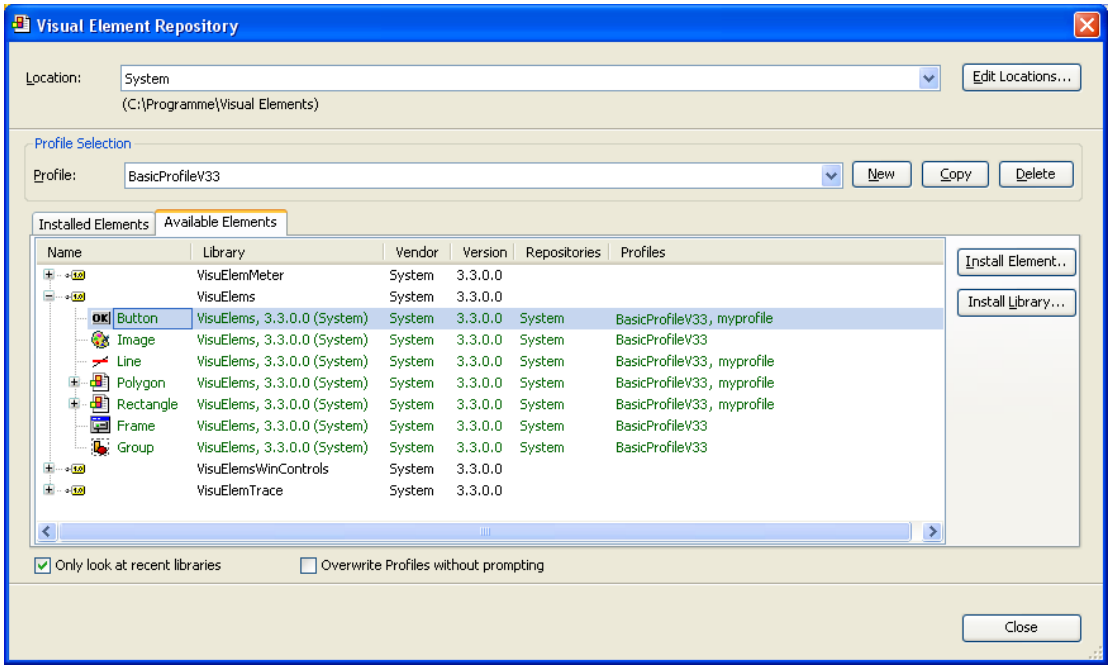


Abb. 179 Visual Element Repository dialog, Available Elements

Which visualization elements are currently available on the system depends on the currently installed libraries containing visualization elements.

The list displays in a tree structure these libraries and their elements. Green-colored entries indicate elements which are already installed in the profile specified in the Profile Selection field.

To install a further library in order to make further elements available, use button **Install Library...**. to open the Library Repository dialog, where the installation can be done like for any other libraries. The new library and its elements will be added to the current list of available elements.

To install one or several elements for getting usable with the specified profile, select it/them in the list and use button **Install Element...** The elements will be added to the 'Installed Elements' list (see above).

Note: Currently only one version of an element can be installed. If you try to install two versions of the same element, an error message will be dumped.

12.19 Library Manager

This command category provides commands the Library Manager Editor, that is for handling the libraries to be used in the project.liotheksverwalter, also für den Umgang mit den im Projekt verwendeten Libraries.

12.19.1 Library Manager commands

The 'Library Manager Editor' plug-in provides in command category 'Library Manager' the commands listed below for the Library Manager. By default, they are items of the 'Libraries' menu. If necessary open the Customize dialog to view resp. modify the menu configuration.

- Add Library...
- Properties...
- Try to Reload Library

12.19.2 Add Library...

This command (category 'Library Manager') by default is part of the 'Libraries' menu and the Library Manager editor window.

Activate the command if you want to include libraries via the currently opened Library Manager into the project. Only libraries which are already installed on your system can be added. Multiple versions of a library can be included in the project at the same time, within one or multiple Library Managers.

The command opens the 'Add Library' dialog with its tabs '**Library**' and '**Placeholder**' .

Subdialog 'Library'

Only display latest versions:

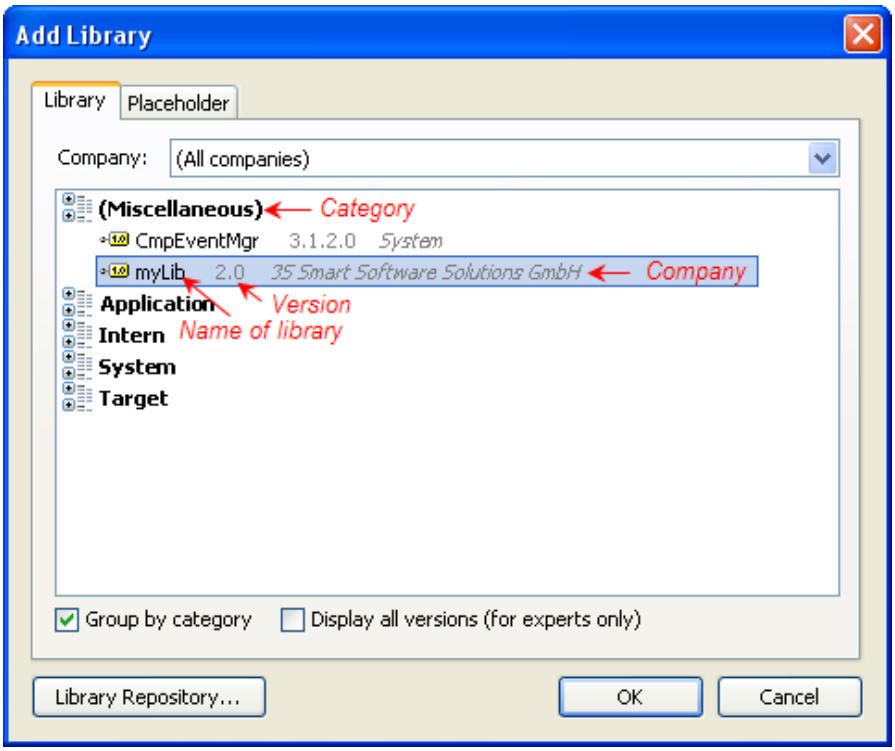


Abb. 180 Add Library dialog, 'Library', Display only latest version

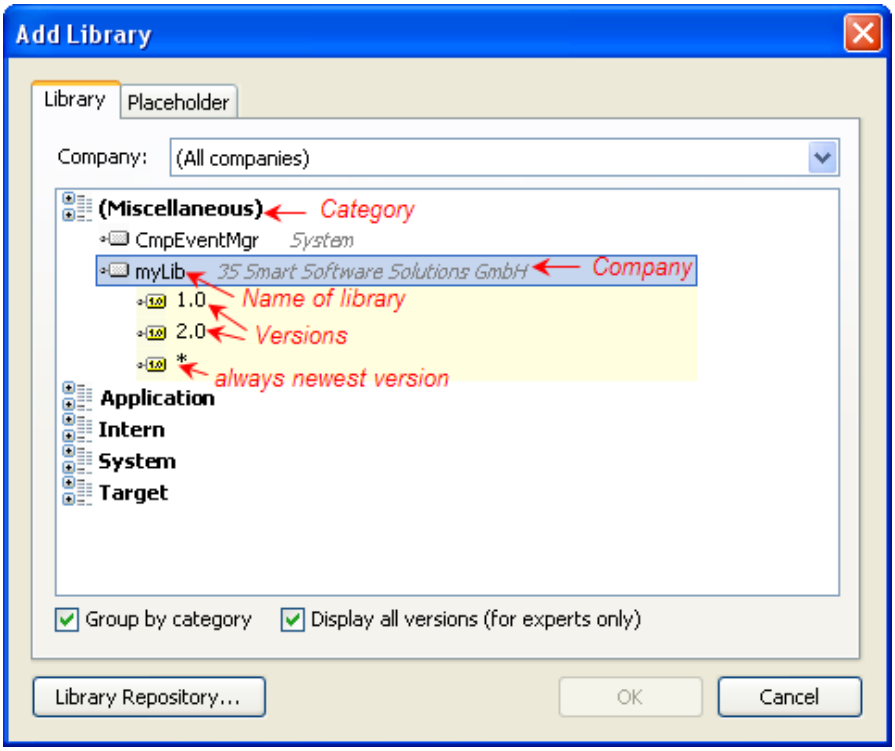


Abb. 181 Add Library dialog, 'Library', Display all versions

Here all libraries currently installed on your system will be listed. You can filter the display by setting a certain providing **Company** from the selection list. "(All companies)" will list all available libraries.

If option **Group by category** is activated, the libraries of the currently set company will be listed according to the available categories, otherwise alphabetically. If the grouping by categories is activated, the categories appear as nodes and for the currently selected category the libraries (or further categories) will be displayed indented below.

Choose the desired library. If option **Display all versions (for experts only)** is activated, all installed **version**(s) of the libraries appear(s) indented below the currently selected library entry. Additionally to the explicit version identifiers each a "*" is available, which means "latest version". In this case you can choose among the versions. By default however this option is deactivated and only the latest version will be displayed. In this case a **multiselection** of libraries is possible: Keep the <Shift> key pressed while selecting the desired libraries.

After having confirmed with OK, the selected libraries will be added to the list in the Library Manager window.

If you want to include a library which is not yet installed on the local system, you might use the **Library Repository** button to get to the Library Repository dialog for doing the required installation.

Subdialog 'Placeholder'

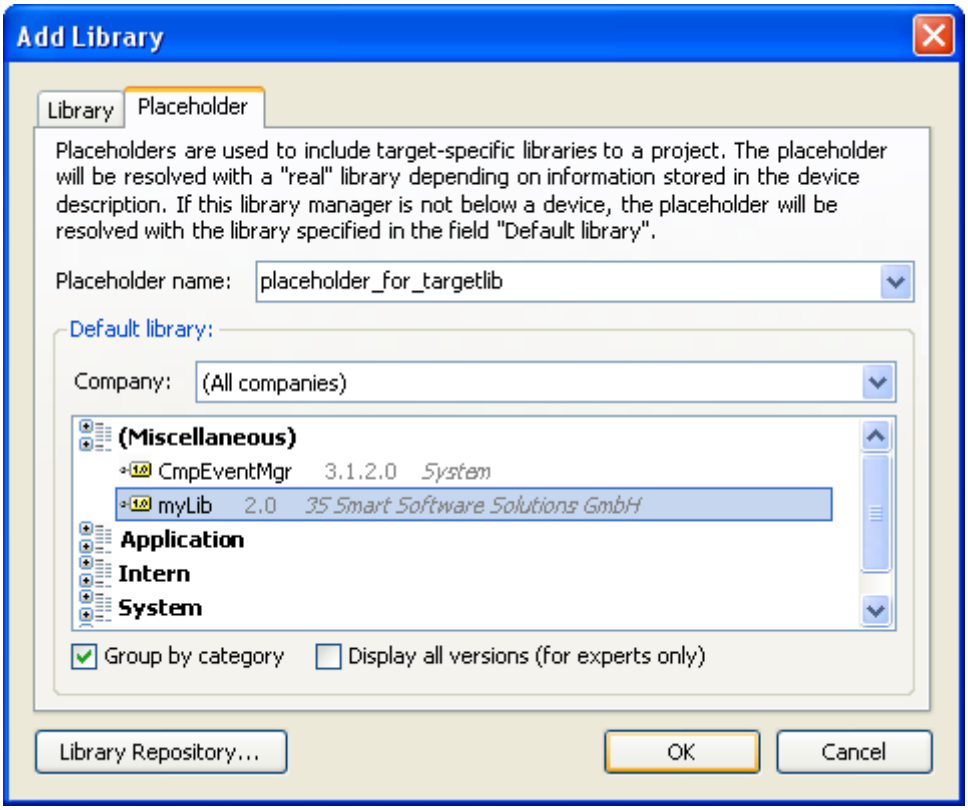


Abb. 182 Add Library dialog, 'Placeholder'

The tab 'Placeholder' is dedicated to the following two use cases:

- Creation of a target independent project
- Creation of a library project <library_xy>, which references another library that is meant to be **target-specific**, that is device-specific.

Placeholder within project

If a project shall be designed as compatible for multiple interchangeable target devices, the target specific libraries have to be included within the library manager of the project via placeholders.

As soon as the target device gets specified, the placeholders will be cast according to the related device description. Even if no device description is actually available, the placeholders allow the project to pass a syntactical check.

To include a library into the library manager via a placeholder, the library has to be selected within the bottom part **Default library** of subdialog 'Placeholder'. Thereby it is possible to constrict the catalogue of proposed libraries according to the providing **company**.

In addition the **placeholder name** has to be inserted in the associated edit field. To ensure correct insertion of the name you might use the selection list offering all placeholder names currently defined in device descriptions.

Placeholder within library project

If the library project is based on further libraries that are **target-specific**, that is device-specific, these libraries have to be included in the library project via placeholders.

This means that instead of specifying one particular library for being included, a placeholder will be inserted, which will be replaced later, when <library_xy> is used in another project for a certain device, by the name of a device-specifically defined library. This name must be specified in the respective **device description file**, <library_xy> which assigns the placeholder name to a "real" library name.

If the library manager for any reason currently is not assigned to a device, the placeholder will be replaced by the **default library** specified here in the dialog. (This for example allows compilation of the currently edited library project without detected errors even if no suitable device description is available at that moment.)

In the **Placeholder Name** field enter any string as a name for the placeholder. Further on choose a **Default library** from the currently installed libraries. This is to be done like described above for adding a library in the 'Library' subdialog and like there option **Display all versions (for experts only)** might be activated to get displayed all currently installed versions of a library.

After closing the dialog with OK, the placeholder library will be entered in the library manager tree. When you open the Properties dialog for the a library placeholder, you get information on the currently set default library.

In the following example the placeholder is not yet replaced ("resolved"); if the library manager later is positioned below a device with an appropriate description file, in column 'Name' the name of the device-specific library will appear.

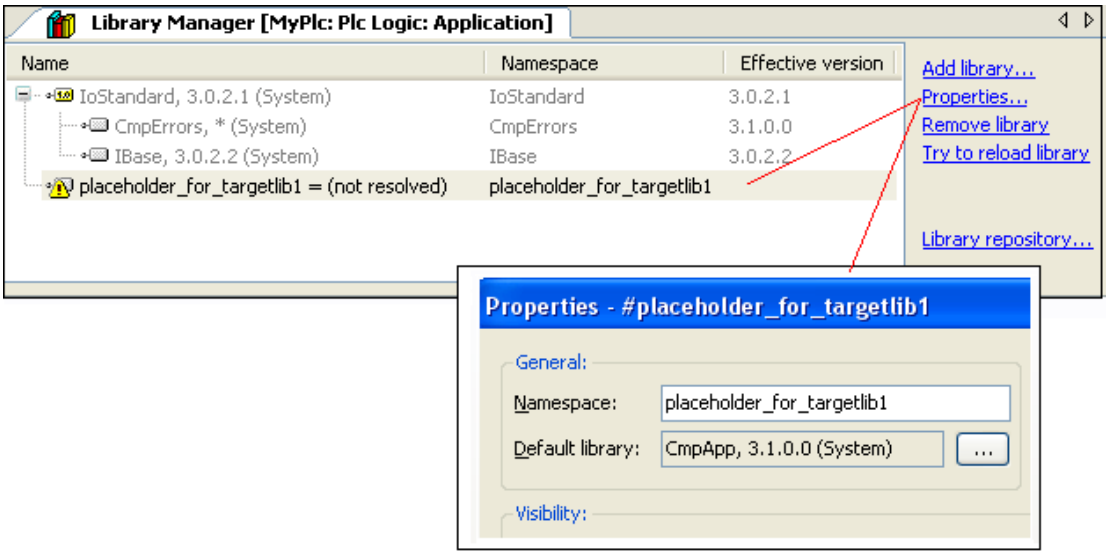


Abb. 183 Example for library placeholder inserted in library manager

12.19.3 Properties...

This command (category 'Library Manager') by default is available in the Library Manager editor window.

It opens the **Properties** dialog for the library which is currently selected in the Library Manager window and allows the following settings concerning namespace, version handling, availability and visibility of library references:

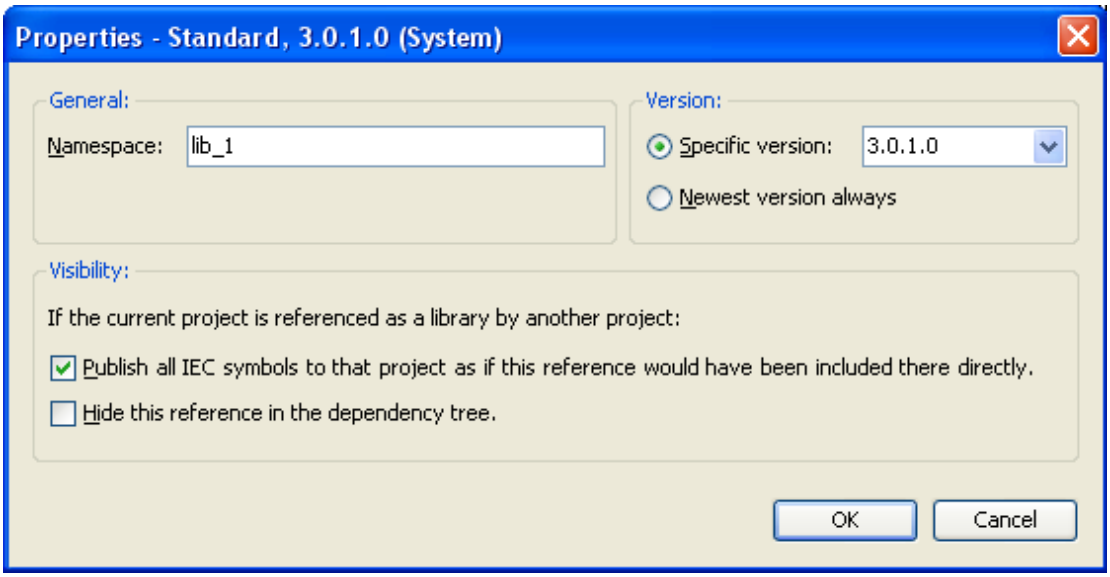


Abb. 184 Properties dialog for library

General:

Namespace: The current namespace of the library is displayed. By default the namespace of a library primarily is identic with the library name, except another string has been defined explicitly in the Project Information when creating the library project. You can edit the namespace anytime here in the properties dialog.

Default library: If a library placeholder is currently selected in the Library Manager, this field will contain the name of the library which should replace the placeholder, if no device-specific library is available.

Version:

Here you can configure, which version of the library should be used in the project:

Specific version: Exactly the version entered here (you can select from the list) will be used.

Newest version always: Always the newest version found in the library repository will be used, that is the modules actually used might change because a newer version of the library is available.

Visibility: These settings are of interest as soon as the library gets included that is referenced by another library: Per default they are deactivated:

Publish all IEC symbols to that project as if this reference would have been included there directly: As long as this option is deactivated, the components of the current library - if referenced by another one - can be accessed uniquely by using the appropriate namespace path (composed of the namespaces of the "father" library and its own namespace in addition to the modules resp. variable identifier.

Note: Recommendation: The option should only be activated, if you want to use "container libraries", not containing own modules, but just including other libraries for the purpose of "packaging" them. This packaging for example allows to include multiple libraries in a project at once just by including the "container library". In this case however it might be desired to get the particular libraries on top-level of the projects' Library Manager, because then the modules can be accessed directly, that is the namespace of the container library can be left out. And exactly that you can reach by prospectively activating the option "Publish...".

Hide this reference in the dependency tree: If this option gets activated, the current library will not be displayed later, when its "father" library is included in a project. This allows to include "hidden" libraries, however needs careful use, because in case of library error messages you might have problems to find the causing library.

12.20 Message View

Notice the commands of category 'Message View' if you want to navigate between messages displayed in the message window or between messages and the implementation code.

12.20.1 Message View commands

The 'MessageView' plug-in provides for command category 'Message View' commands (see below) which serve to navigate between messages in the Messages Window (Messages view) resp. between messages and the concerned position in the project.

By default these commands are part of the 'Edit' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'Message View').

- Go To Source Position
- Next Message
- Previous Message

12.20.2 Go To Source Position

If the cursor is placed in a line of the message window which indicates a detected error or alert (warning), use this command to get to that position in the project (source position), which has caused the message. The cursor will be placed there. If the concerned object was not yet opened in a window, it will be opened.

To get to the next resp. previous message use Next Message resp. Previous Message.

12.20.3 Next Message

Shortcut: <F4>

Use this command in the message window to set the focus to the next message.

12.20.4 Previous Message

Shortcut: <Shift>+<F4>

Use this command in the message window to set the focus to the previous message.

12.21 Objects

The command category 'Objects' provides commands for handling the project objects and folders.

12.21.1 Object Commands

The 'ObjectCommands' plug-in provides for command category 'Objects' commands for handling project objects (see below). By default these are part of the 'Project' menu. If necessary, open the Customize dialog to view resp. to modify the menu configuration (notice category 'Objects').

The commands:

- Add Object
- Edit Object
- Edit Object with....
- Edit Object - Offline
- Implement interfaces...
- Export ...
- Import...
- Set Active Application

Add Object

Symbol:

This command (category Objects) opens a submenu providing the objects available for getting inserted at the currently selected position in the POUs or Devices tree. For example an Application object can only be added if an existing Device object is selected in the Devices Window, an Action object only can be added if already a Program object is available and is marked.

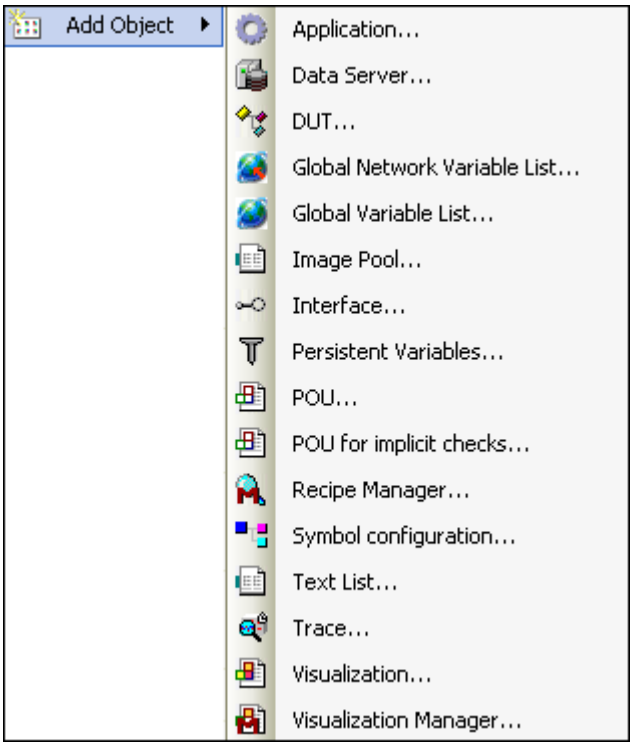


Abb. 185 'Add Object' submenu, example

Select the desired object type and in the appearing dialog define a **Name** which should be used for the object in the Devices resp. POUs tree. Please notice the given recommendations on the naming in order to get the name as unique as possible. Depending on the object type also further configuration settings might be available.

For further information on this please see the respective help page of the particular object type resp. the corresponding editor:

Action
Application
Data Server
Data Source
DUT
External File
Global Variable List
Global Network Variables List
Image Pool
Interface
Library Manager
Method
Persistent Variables
POU
POUs for implicit checks
Property
Recipe Manager
Symbol configuration
Recipe Definition
Task Configuration
Task
Text List
Trace
Transition
Visualization
Visualization Manager

To **rename** an object in the POUs tree, click on the entry to open an edit frame or use the Properties dialog (context menu).

Notice that a **Device object** is to be inserted via a separate command: Add Device.

Edit Object

Symbol:

Use this command (category Objects) if you want to edit or view an object which is available in the POUs or Devices view. The object must be selected, then the command is available in the context menu and by default in the project menu.

The command will open the object in the appropriate editor. If used in online mode you will get a dialog asking you in which view the object should be opened (Select Online State dialog).

Edit Object with....

Currently this command works like Edit Object.

Edit Object - Offline

This command (category Objects) is available in online mode. After a login to the application has been performed, a double-click on an entry in the device tree will open the associated editor in online mode (provided that an instance of the selected object belongs to that application). If you prefer to open the object in offline mode, select it by an single click on the entry in the device tree and use the command "Edit Object (Offline)" that will generally be retrieved under the menu item Project.

Implement interfaces...

This command (category Objects) can be used to update a function block concerning the implemented interfaces. This means: If the interface definition has been modified, for example another method has been added, this modification can be done accordingly also in the function block by selecting the function block object in the POUs- resp. Devices tree and using command 'Implement interfaces...'.

Example:

1. Interface ITF1 contains method GetName; When creating the function blocks FB1 and FB2, each implementing ITF1, GetName will be inserted automatically also below FB1 and FB2 in the object tree:

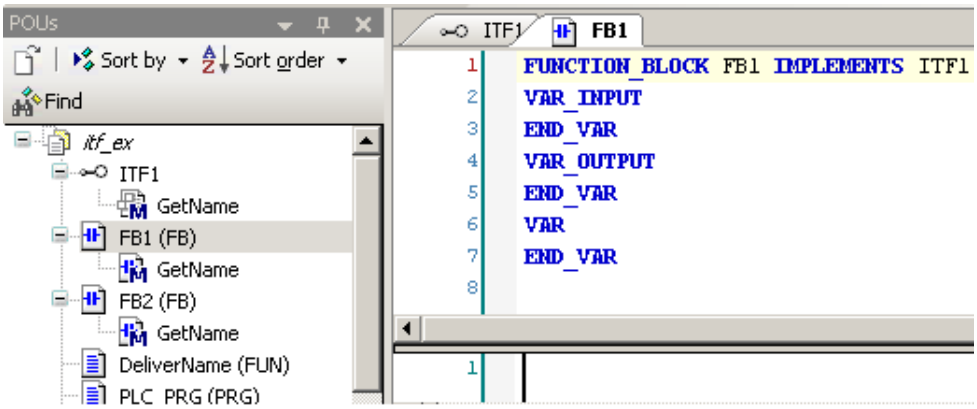


Abb. 186 FB1 and FB2 implement ITF1

2. The definition of InterfaceITF1 now gets extended by method GetID:

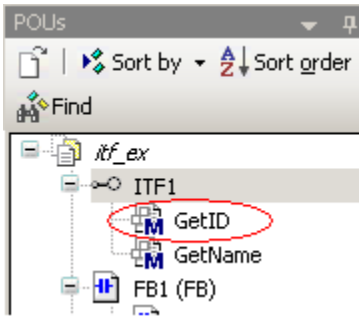


Abb. 187 Interface modification

3. Update of FB1 concerning interface ITF1 via command 'Implement interfaces...' (context menu):

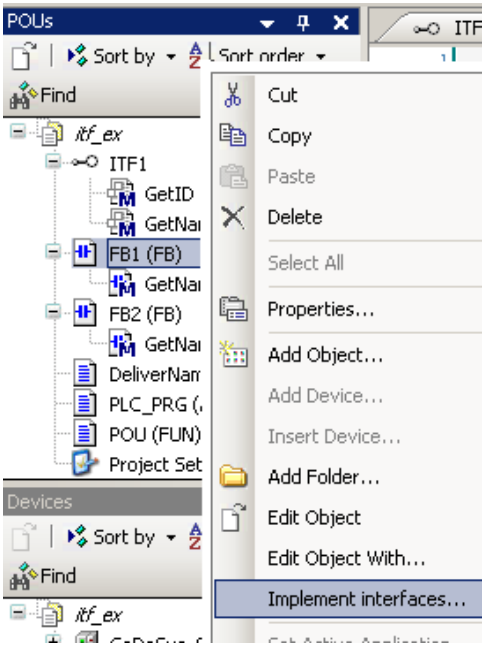


Abb. 188 Update of the interface in FB1

4. Updated interface definition in the function block:

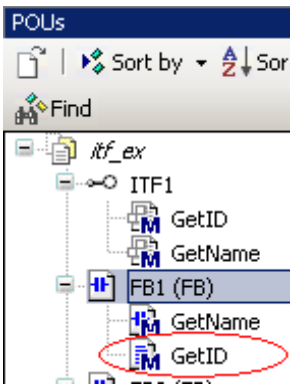


Abb. 189 Interface update in FB1

Export...

Use this command (category Objects) to export particular objects or a total tree to one or several external files. Export files are created in XML format and by default have the extension "**:export**".

The following cases are possible:

- If the command is used from the **'Project' menu** (default position of the command), first the **Export** dialog will be opened, where you have to select the desired objects from the POUs- resp. Devices tree. "Child objects" of a selected object (for example actions below a program object) per default are automatically selected too. The selection however can be modified by setting or removing the corresponding checks.
  If option **One file per subtree** is activated, all objects belonging to the same subtree will be exported into the same export file, otherwise a separate file will be created for each particular object. After having selected the desired objects a dialog for defining the target folder for the export file(s) and - if only one file will be created - for defining the file name. After having closed this dialog, the export will be done.

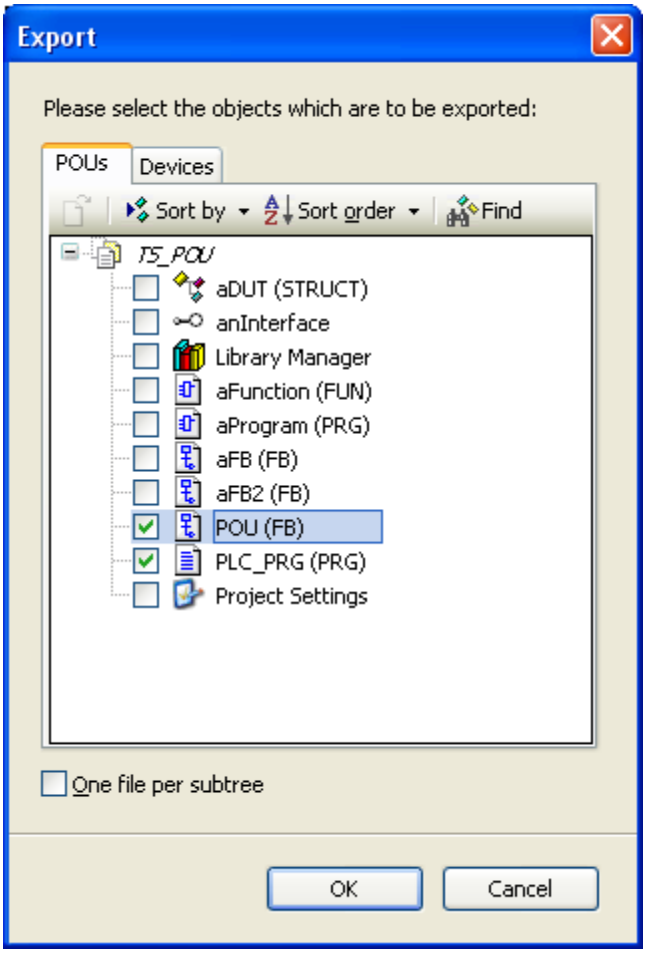


Abb. 190 Dialog 'Export'

- If the command is used from the **context menu**, a dialog for defining the target folder for the export file(s) will open. In the specified folder then for each object currently selected in the POUs or Devices tree an export file will be created. In this case the export file(s) automatically get named like the object resp. the path of the object within the POUs or Devices tree. Examples: "Device.export" for a top-level object "Device"; "Device.App1.POU1.export" for an object "POU1" which is inserted below the application object "App1" which again is inserted below object "Device". Note that when selecting an object the possibly available "child objects" (for example actions below a program object) will not automatically be exported too, like it is the default when using the 'Export' dialog (see above). They must be selected explicitly.

Export files can be re-imported into a project via command Import....

Import...

Use this command (category Objects) to import one or several objects which are available in the form of an export file in xml-format. A dialog for browsing for such files will open, by default the filter being set to files with extension "**.export**". "export"-files can be created by exporting objects from a project via command 'Export'). The imported object(s) will be added to the current project. If already objects with the same name are available, you will get the appropriate message dialogs and if applicable you can decide whether objects should be overwritten or not.

Set Active Application

Use this command (category Objects) to define which of the available applications defined in the **Devices** view should be the active one. If an application is selected in the objects tree the command will be available in the context menu and by default in the Project menu. After performing the command, the chosen application will appear as an bolded entry in the objects tree.

Keep in mind that online activities each only concern the active application!

12.21.2 NavigatorControl

Objects commands - NavigatorControl

The 'Navigator Control' plug-in provides commands for category 'Objects' commands for handling folders in the project view windows. By default these commands are part of the 'Project' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'Objects').

The commands:

- Add Folder...

Add Folder...

Symbol:

This 'Objects' command (category Objects) will be available in the context menu if you are working in the POUs or Devices view.

If you want to add a folder on the uppermost level in the objects tree of a view window, make sure that the window is active, but that no existing object or folder is selected. If you want to add a folder on a lower level, select that entry in the tree below which the folder should be inserted.

Then perform the command to open the **Add Folder** dialog and there define the name of the new folder. The name may contain spaces, digits and special characters.

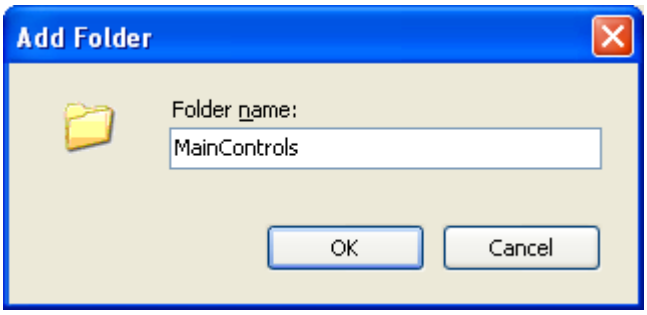


Abb. 191 Add Folder dialog

After confirming with OK the folder will be inserted in the tree preceded by the folder icon .

If the folder has been created below an existing tree entry, click on the plus-sign which will now appear before that entry to make visible the new folder.

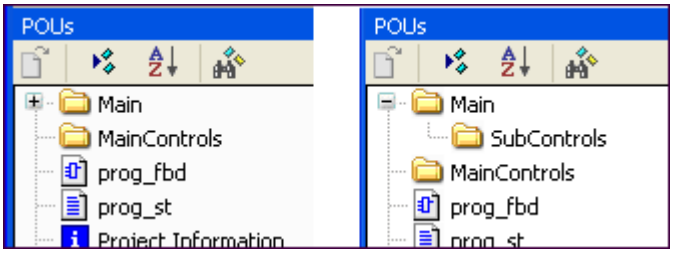


Abb. 192 Example of folders in POUs view

12.21.3 PLCopenXML

Objects commands - PLCopenXML

The 'PLCopenXML' plug-in provides commands for command category 'Objects' for the export and import of project objects to resp. from an XML.-file according to the PLCopen format.

By default these commands are part of the 'Project' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'Objects').

The commands:

- Export PLCopenXML...
- Import PLCopenXML...

Export PLCopenXML...

This command of category Objects serves to export all or a selection of objects of the current project to a **xml-file** according to the PLCopen format.

The command opens the Export PLCopenXML dialog where you can select the desired objects:

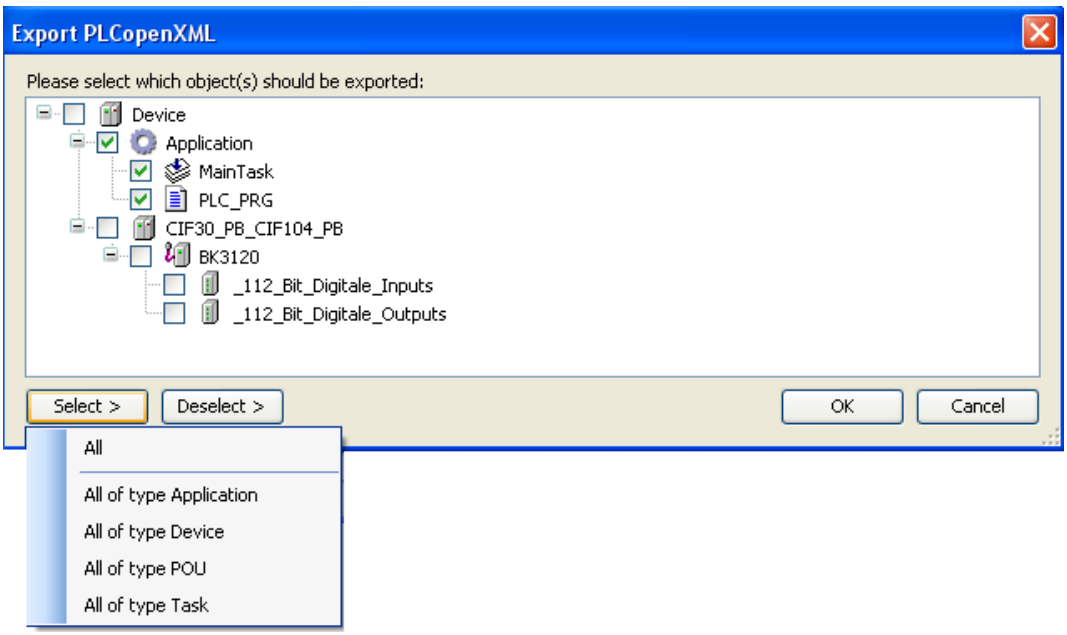


Abb. 193 Export PLCopenXML dialog

Due to the restrictions given by the PLCopen format, there are also restrictions concerning the selection of the objects to be exported. You will realize this when checking the desired objects in the **Export PLCopenXML**dialog by a mouse-click in the preceding checkbox. Selecting certain objects will automatically set the selection checkmark also for other objects. For example the application object will be checked too as soon as the assigned task configuration object gets selected.

To ease the selection you might use the **Select** resp. **Deselect** button and choose one of the options. There is the possibility to check resp. deselect at once all objects of a certain object type (Application, Device, POU, Task) or all objects at all.

After having defined the desired selection, confirm with OK. The standard dialog for saving a file will appear, where you can define the name for the export file. The file filter automatically is set to **PLCopen xml files (*.xml)**.

The export file might be re-imported in a project by command Import PLCopenXML....

Import PLCopenXML...

This command of category Objects serves to import the content of a xml-file describing objects in PLCopen format. This file can have been created from a project by the Export PLCopenXML command.

The command opens the standard dialog for browsing for a file. The filter is automatically set to to **PLCopen xml files (*.xml)**. Select the desired file and confirm with OK. The **Import PLCopenXML** dialog will open:

Contents tab:

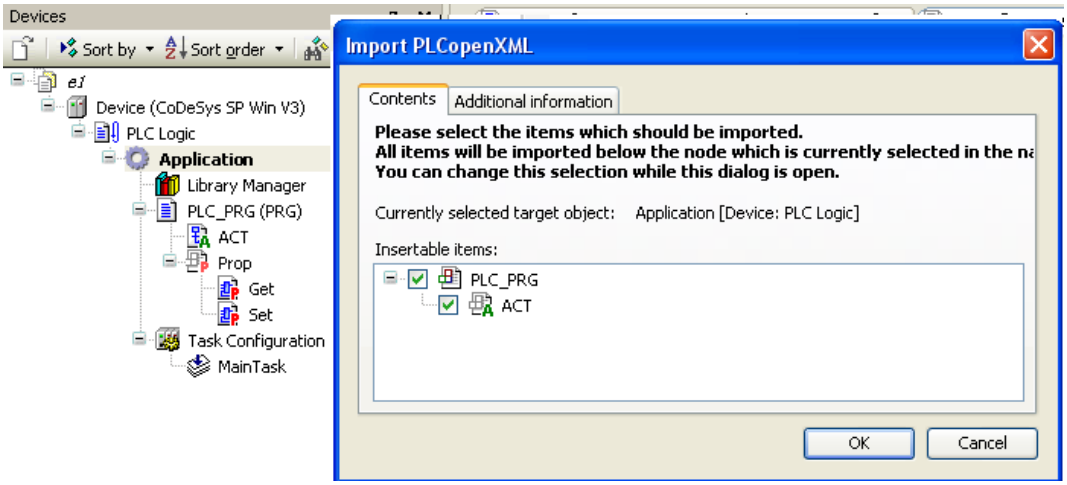


Abb. 194 Import PLCopenXML dialog, example

Depending on the object currently selected in the projects' POUs or Devices tree (**Currently selected target object**) this dialog will only list those objects resp. entries from the xml-file, which can be inserted at this position (**Insertable items**). Which of them actually should be imported can be configured (as far as this is possible due to the prescribed dependencies between certain object types) by setting or removing a checkmark in the box preceding the particular entry.

After having selected the desired objects, confirm with OK to get the objects inserted in the project

If there is already an object with the same name, you will be asked in a dialog, what to do:

- **Rename the new object.** The object will be inserted below the existing one and will get a new name "<name>_<n>", where n is a running number, which is "1" at the first import of an eponymous object; example: "PLC_PRG" -> "PLC_PRG_1"
- **Rename the new object. Do the same for all subsequent conflicts.** The renaming will be done as described above for "Rename the new object" will be done for the current object and for all subsequent objects where the same conflict arises.
- **Replace the existing object.** The existing object will be removed and replaced by the imported one.
- **Replace the existing object. Do the same for all subsequent conflicts.** The replacement as described above for "Replace the existing object" will be done for the current object and for all subsequent objects where the same conflict arises.
- **Skip the new object.** The object will not be imported.
- **Skip the new object. Do the same for all subsequent conflicts.** The current object and all subsequent objects where the same conflict arises will not be imported.

Additional information:

This tab shows the following information read from the File header and Content header of the PLCopen XML-file:

File header: Company name, Company URL, Product name, Product version, Product release, Creation date/time, Content description

Content header: Name (project name), Version, Modification date/time, Organization, Author, Language, Comment

12.22 Online

The command category 'Online' provides commands applicable in online mode.

12.22.1 Online commands - Online Commands

The 'Online Commands' plug-in provides commands of category 'Online' for controlling the application program on a real or on the simulation target system (device, PLC) after having logged in.

Most of these commands are by default available in the 'Online' and 'Debug' menus. If necessary open the Customization dialog to view or modify the menu settings.

When calling the commands, which are marked by a an asterisk * in the list below, the user will be prompted to confirm. Such a confirmation prompt will additionally appear for the commands, which are marked by two asterisks ** in the list below, if option "Secure online mode" is activated in the communication settings of the respective device.

WARNING!
Extraordinary changes of variable values in an application currently running on a controller can cause unwanted behaviour of the controlled system.
Evaluate possible dangers before Writing or Forcing of variables and make appropriate security precautions.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

WARNING!
Online Change **modifies the running application program and does not effect a restart process.**
Make sure that the new application code nevertheless will effect the desired behaviour of the system.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

The commands (* and ** are explained above):

Online menu:

- Login
- Logout
- Create boot application
- Download *
- Online Change *
- Source download to connected device
- Multiple Download **
- Reset warm *
- Reset cold *
- Reset origin *
- Simulation

Build menu:

- Start **
- Stop **
- Single Cycle **
- Write Values **
- Force Values **
- Unforce Values
- Add All Forces to Watchlist
- Display Mode

File menu:

- Source download...
- Source upload...

Not per default available in any menu (see the Customization dialog):

- Connect to '<device>'
- Disconnect from '<device>'
- Reset origin device
- Force values [all applications]
- Write values [all applications]
- Unforce all values [all applications]
- Release Forcelist **

12.22.2 Login

Default Shortcut: <Alt> + <F8>

This command (category Online) connects the application to the target device (PLC or simulation target) and thus changes into the **online mode**.

For a login with the current active application, te code generation must have been completed without errors (see below: Build process before Login) and the Communication Settings of the device must be configured correctly.

If the **communication settings** are not yet set properly, then a dialog box will appear asking you whether you want to set the "active path" by one of the following options (or to cancel the login operation):

- Option1 (Yes): The communication settings dialog should be opened and a network scan performed automatically. If the device recently defined in any project as "active path" (this information is stored on your local system) is found, then this device will automatically be set to be the active one.
- Option 2 (No): The communication dialog should be opened without any further automatically executed configuration action. You have to configure the active path manually.

If the Login command is called from the Online menu, the currently **active application** will be concerned.

If the command is called from the context menu when an application object is selected in the device tree, then the **selected application will be affected, no matter whether it is set active.**

Note: Via the object properties dialog you can allocate memory for the application on the SPS .

The following situations are possible when going to login with the currently active application (error-free, communication settings configured properly):

- **The application is not yet available on the controller**: You will be asked you to confirm the download. For this purpose a dialog box with the following text will open: "Application <application name> does not exist on device. Do you want to create it and proceed with download?" The **Details** button in this dialog provides information on already applications on the PLC (corresponding to the information available in the Applications dialog of the Device Editor).
- **The application project is already available** on the controller and has **not been modified** since the last download, the login will be done without further interaction with the user.

- **Another version of the application is already available** on the controller in **not running** mode. You will be asked whether that should be replaced via a dialog box with the following text: "Unknown version of Application '<Application>' on target: Do you want to perform a download and replace the application?". The **Details** button in this dialog provides some information (Project name, Last modification, IDE version, Author, Description) on the application in the IDE (integrated development environment = programming system) confronted to that on the controller.
- **A version of the application is already available** on the controller in **RUN** mode. You will be asked whether you really want to log in regardless and overwrite the currently running application. For this purpose a dialog box with the following text will open: "Warning: An unknown version of the application '<application name>' is currently in RUN mode on the PLC. However, do you want to download the latest code and replace the existing application?" The **Details** button in this dialog provides information on the application version in the programming system confronted to that on the controller.
- **The application is already available** on the controller but has been **modified** since the last download. You will be asked whether an online change should be done, or a login with loading the complete application code, or a login with out any change of the running application. For this purpose a dialog box with the following text will open: "The code has been changed since the last download. What do you want to do? - Login with online change. - Login with download. - Login without any change." The **Details** button in the dialog provides information on the modified application in the programming system confronted to the previous version on the controller.

Additionally: If there are already one or several applications on the PLC from one or several other projects, a message box will appear where you can decide to remove it / them from the PLC before downloading the current application. (This corresponds to the functionality of the Applications dialog of the Device Editor).

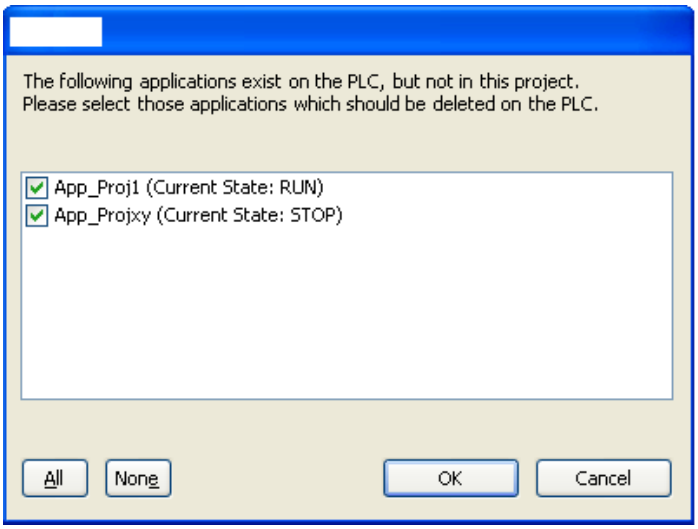


Abb. 195 Message box, Applications on PLC

Note: Child applications (e.g. for the Symbol Configuration, if supported by the device, or for the Trace), which have been downloaded to the PLC once and deleted within the device tree during a subsequent logout from the device, will provoke no online change at a repeated login to the device. At least there will be a request if you want to delete them also from the device.

Build process before Login

Before Login and if the **currently concerned application project has not been compiled since** having got opened or since the last modification, it will get compiled. This means the project will be built corresponding to a "Build" run in offline mode and additionally compilation code for the PLC will be generated.

If errors are detected during compilation, you will get a message box informing you about that with the following text: "There are compile errors. Do you want to login without download?" You might choose to correct the detected errors first, or to login nevertheless, in this case to that version of the application which is possibly already available on the controller.

The errors are listed in the message window in category "Build". See : Build.

Information on the download process

When the project gets loaded to the PLC completely at Login or partially at Online Change, then the message window will show information on the generated code size, the size of global data, the needed memory space on the controller and in case of online change also on the concerned POUs.

12.22.3 Logout

Default Shortcut: <Ctrl> + <F8>

This command (category Online) effects an log out of the application. It disconnects the programming system from the target device (PLC or simulation target) and thus changes into the **offline mode**.

If the command is called from the Online menu, the currently **active application** will be concerned. If the command is called from the context menu when an application object is selected in the device tree, then the **selected application** will be affected, no matter whether it is set active.

12.22.4 Create boot application

This command (category Online) is available in online or offline mode for creating a boot project, also named "boot application". The boot application serves the purpose to provide an application on the PLC, which will be **loaded automatically** when the PLC gets started.

Use in online mode:

The boot application automatically will be stored as **<application name>.app** on the PLC.

Use in offline mode:

The standard dialog for **saving a file** will be opened. Select a folder, where you want to store the current application as boot application file. Automatically the file filter is "Boot application files" and extension ".app" will be applied to the defined file name. So you might store the boot project in any folder in order to transfer it to a PLC later. After confirming with **Save** a further dialog will appear, where you will be asked, whether a possibly already existing compile information file in the project folder should be overwritten:

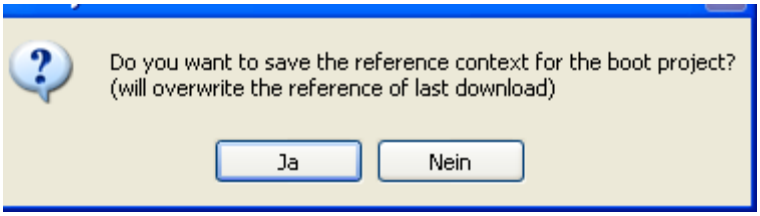


Abb. 196 Dialog for saving new boot project reference data

Choose **Yes**, if you intend to transfer the new boot application file to the PLC by an external tool, but want to log in to this application later without being forced to perform a new download. This namely would be the case, if due to a previous application download already a compile info file would exist (containing code and reference data of this previous application) which of course would not match with that of the new boot application.

Note for »PLC Designer« SP Win V3 devices:

In order to get a boot application loaded to the controller, it not only must be copied there but also the following entries must be made in the configuration file (*. cfg) of the device: name of the application, initialization of the retain area (not for symbol applications).

[CmpApp]
...
Application.1=MyApplication
Application.2=MyApplication.__Symbols
MyApplication.FirstLoadInitRetains=1

12.22.5 Download

This command (category Online) is available in online mode. It includes a build and code generation run of the currently active application program. So besides a syntactical check (build process) also application code will be generated and loaded to the PLC. In the project folder the compile information file **<projectname>.<devicename>.<application ID>.compileinfo** will be created.

Note: All variables except for the persistent variables will be re-initialized.

Note: Via the object properties dialog you can allocate memory for the application on the SPS .

Please see the description of the Login command for the possible situations around login and download.

If you try to download an application while the same version of this application is already available on the controller, you will get an dialog telling you "Program has not changed. No download will be performed." The application will not be downloaded to the PLC.

During download the message window in category "Build" will show a protocol of the running actions (code generation, initialization etc.) and information on the memory areas, code size, global data size and size of allocated memory. In contrast to download at an Online Change as from V3.4.0.0 the modified objects will not be listed any more, for reasons of clarity.

12.22.6 Online Change

WARNING!
Online Change modifies the running application program and does not effect a restart process.
Make sure that the new application code nevertheless will effect the desired behaviour of the system.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

Notes:
1. When an online change is done, the application-specific initializations (homing etc.) will not be executed because the machine keeps its state. For this reason the new program code might not be able to work as desired.
2. Pointer variables keep their values form the last cycle. If there is a pointer on a variable, which has changed its size due to an online change, the value will not be correct any longer. Make sure that pointer variables get re-assigned in each cycle.

This command (category Online) is used to perform an "Online Change" on the currently active application.

Online change means that only the modified parts of an already running project will be re-loaded to the PLC. This is not possible after a Clean all or Clean application operation! The clean process removes the compile information which is automatically stored at each code generation and which is the basis on an online change.

During download the message view in category "Build" lists - among the usual information on the download process - also the changed interfaces, the concerned variables and all objects for which new code has been generated. If data locations change, a message box will hint at possible problems referring to the use of pointers.

Notice that an online change automatically will be offered when you are going to log in on a PLC with an application program which is already running there, but has been modified since the last download.

For information on code generation and online change also see: Code_Generation_and_Online_Change.

12.22.7 Source download...

This command (category Online) by default is available in the File menu for creating and transferring an archive file of the actual project to any device.

The command opens the **Select Device** dialog, where you have to choose the network path to the PLC like in the Communication Settings dialog. Select the appropriate entry in the tree of available devices and press OK. This will set up a connection to the device as long as the source code gets downloaded in the form of an archive file (see also the description of the source download on the currently connected device).

The source code can be re-load to the programming system in offline mode by using command Source upload.

The default settings concerning destination device, content and timing for the source download are defined in the Project Settings, category 'Source Download'.

12.22.8 Source upload...

This command (category Online) by default is available in the File menu for opening a project from a PLC. For this purpose a project archive file must be available there, possibly generated by the Source download function.

The command opens the **Select Device** dialog, where you have to choose the network path to the PLC like in the Communication Settings dialog. Select the appropriate entry in the tree of available devices and press OK.

The **Project Archive** dialog opens, where you can configure which contents of the archive should be extracted for upload and to which path they should be copied. The usage of this dialog corresponds to that of the Project Archive/Extract archive function. After confirmation with OK the files will be copied. If a file is already available in the specified path, you will be asked whether it should be overwritten.

Then a dialog box will appear, asking whether the extracted project should be **opened** in the programming system.

12.22.9 Source download to connected device

This command (category Online) by default is available in the Online menu for creating and transferring an archive file of the actual project to the currently connected PLC. Regard also the possibility to download the project sources on any device via command Source download...

After performing the command the progress of creating and downloading the archive file will be indicated in the status bar. A **Cancel** button is available in the lower right corner.

The archive file by default is named **archive.prj**. It can be re-load to the programming system in offline mode by using command Source upload.

The default settings concerning destination device, content and timing for the source download are defined in the Project Settings, category 'Source Download'.

12.22.10 Start

Symbol:

This command (category Online) starts the application program on the device (PLC).

If the command is called from the Online menu, the currently **active application** will be concerned. If the command is called from the context menu when an application object is selected in the device tree, then the **selected application** will be affected, no matter whether it is set active.

12.22.11 Stop

Symbol:

This command (category Online) stops the application program on the device (PLC).

If the command is called from the Online menu, the currently **active application** will be concerned. If the command is called from the context menu when an application object is selected in the device tree, then the **selected application** will be affected, no matter whether it is set active.

12.22.12 Single Cycle

This command (category Online) will process one cycle of the currently active application.

12.22.13 Multiple Download...

This command (category Online) is available in online as well as in offline mode. It includes a build and code generation run of all applications contained in the project that have been selected by the user. So besides a syntactical check (build process) of these applications also the related application codes will be generated and loaded to the related PLC. In the project folder for each of the applications selected for download a corresponding compile information file **<projectname>.<devicename>.<application ID>.compileinfo** will be created.

After performing the "Multiple Download..." command a list of all applications contained in the project will be provided within the following dialog:

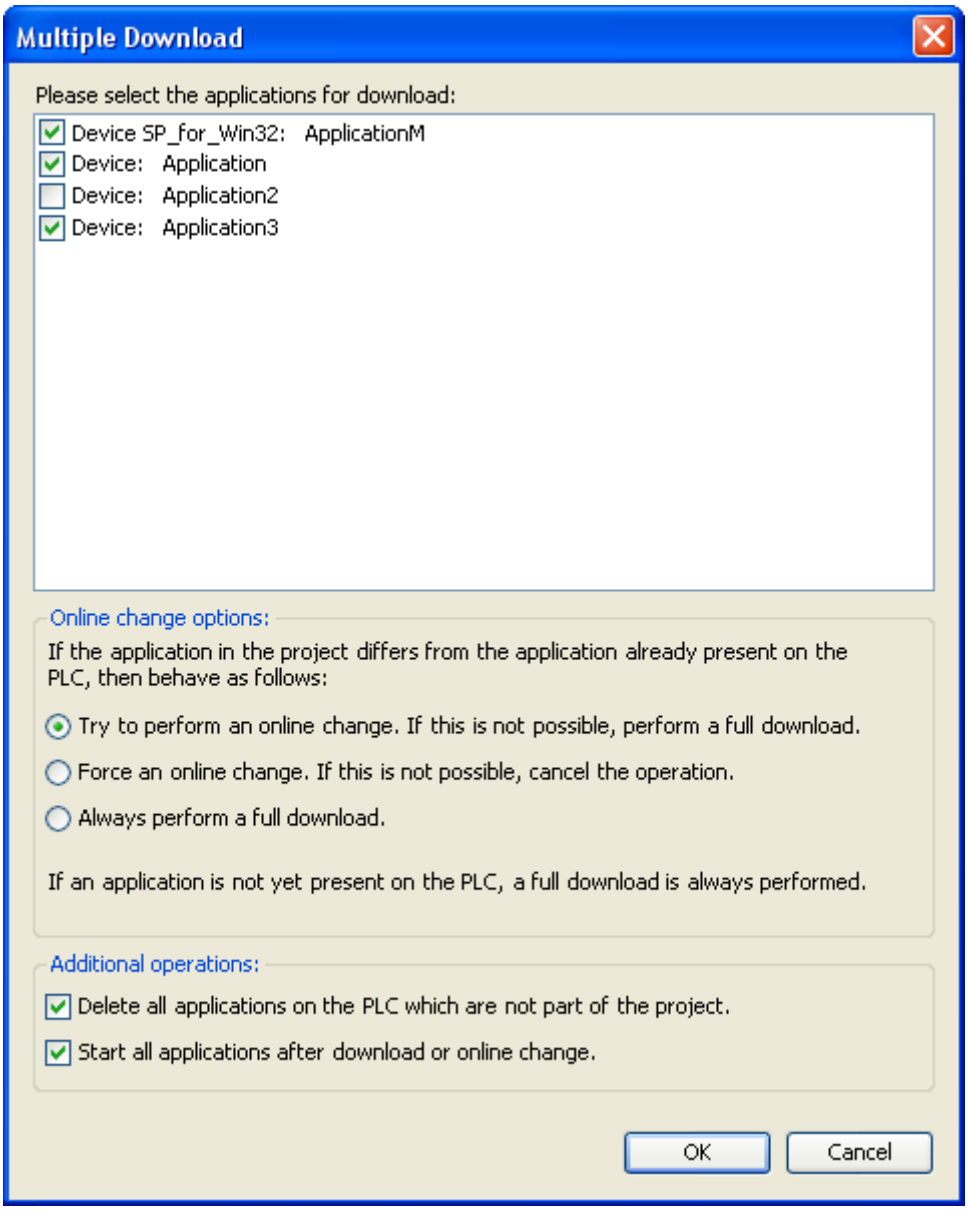


Abb. 197 Dialog "Multiple Download"

An application can be selected for download by activating its preceding checkbox. Multiple applications may be selected, even if they won't be downloaded to the same device. By default either all applications or the applications last time selected are activated.

In case that an application selected for download is differing from an previous version that has already been downloaded to the PLC before, you may choose among the following options:

- **Try to perform an online change. If this is not possible, perform a full download.** By default this option is selected. So the modified parts of the selected applications will be changed on the PLC and only the newly created parts of the selected applications will be downloaded to the PLC.
- **Force an online change. If this is not possible, cancel the operation.** If an online change for (at least one of) the selected applications is not possible (for example if the command 'Clean all' or 'Clean Application' had been performed before), the download will not be executed.
- **Perform a full download.** Regardless the versions already existing on the PLC all parts of the selected applications will be reloaded to the PLC.

For applications that are not yet present on the PLC a full download will be carried out automatically.

In addition you may decide (by activating the corresponding checkbox)

- whether old applications that are no longer part of the project should be deleted on the PLC
- whether the selected applications should get started after the download has been performed

Be aware that variables of type PERSISTENT will generally not get initialized. Instead, case that the data layout has been changed, they will get initialized automatically.

After confirming your settings in the dialog with **OK**, at first a syntactical check of all selected applications will be performed. Thereon for each application communication with the related device has to be verified before the download will be executed.

The download being complete a list of the selected applications will arise with detailed information on the operations performed for each application.

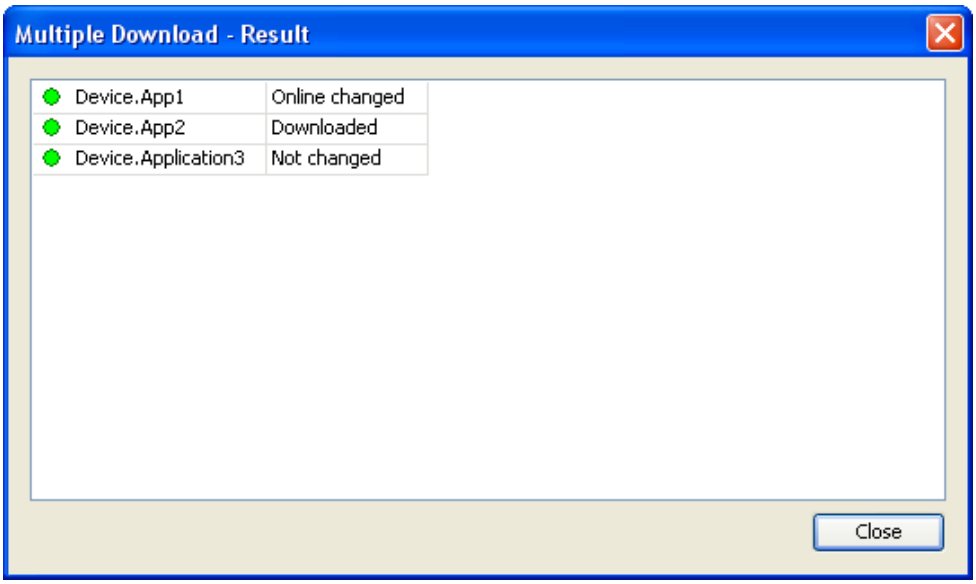


Abb. 198 Dialog "Multiple Download - Result"

12.22.14 Reset warm

This command (category Online) is available in Online Mode. It resets – with exception of the remanent (retain, persistent) variables - all variables of the currently active application to their initialization values.

If you have initialized variables with specific values, they will be reset exactly to that value. All other variables are set at a standard initialization value (for example, integers at 0).

As a precautionary measure,»PLC Designer« asks you to confirm your decision before all of the variables are overwritten. The situation is that which occurs in the event of a power outage or by turning the controller off, then on (warm restart) while the program is running.

A reset disables the breakpoints currently set in the application. If the command **Reset warm** is called during the program run is hold on a breakpoint, the user will be asked whether the cycle should be finished before performing the reset or if the reset shall terminate the task and perform the reset immediately. Be aware, that not all runtime systems are able to perform a reset without finishing the cycle before.

After a reset use the 'Start' command to restart the application.

See also 'Reset origin', 'Reset cold'.

12.22.15 Reset cold

This command (category Online) is available in Online Mode. It corresponds to 'Reset warm', but besides of "normal" and persistent variables also sets back retain variables (!) of the currently active application to their initialization values. The situation is that which occurs at the start of a program which has been downloaded just before to the PLC ("cold start").

A reset disables the breakpoints currently set in the application. If the command **Reset cold** is called during the program run is hold on a breakpoint, the user will be asked whether the cycle should be finished before performing the reset or if the reset shall terminate the task and perform the reset immediately. Be aware, that not all runtime systems are able to perform a reset without finishing the cycle before.

See also 'Reset warm', 'Reset origin'.

12.22.16 Reset origin

This command (category Online) is available in Online Mode. It resets all variables of the **currently active application**, including the remanent ones to their initialization values and **erases the application on the PLC**.

A reset disables the breakpoints currently set in the application. If the command **Reset origin** is called during the program run is hold on a breakpoint, the user will be asked whether the cycle should be finished before performing the reset or if the reset shall terminate the task and perform the reset immediately. Be aware, that not all runtime systems are able to perform a reset without finishing the cycle before.

See also 'Reset warm', 'Reset cold'.

12.22.17 Write Values

Default Shortcut: <Ctrl> + <F7>

WARNING!
Extraordinary changes of variable values in an application currently running on a controller can cause unwanted behaviour of the controlled system.
Evaluate possible dangers before Writing or Forcing of variables and make appropriate security precautions.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

To "Write" a value by this command (category Online) means to set a variable on the PLC to a defined value at the beginning of the next execution cycle. The command effects all variables of the **currently active application**, which are prepared for writing.

To prepare variables for writing, the desired value must be defined in online mode in one of the following places which are used for monitoring :

- In a watch view defined in the project, containing a list of variables to be monitored (watch list).
- In the online view of the object within the declaration part of the respective editor.

Note: Command "Write values [all applications]" which effects all applications of a project, is not per default inserted in any menu. (if needed, please see the Customization dialog)

Note: See in this context also the 'Force values' command for permanently setting a defined value.

Example:

Open an object in online mode, for example a program written in ST. In the declaration part of the editor window you will find the displayable expressions listed in a table. Click on the concerned field in column **Prepared value**, enter the desired value. Perform command **Write values** which is by default part of the **Online** menu. The value will immediately be written to the respective field in column **Value**, which means that it is written to the controller. The Prepared value field will be empty again.

The same can be processed in a watch view containing the desired expression.

Notice in this context the **'Prepare Value' dialog** which is available for currently forced variables and where a new value to be written can be defined.

12.22.18 Force Values, Prepare value dialog

Default Shortcut: <F7>

WARNING!
Extraordinary changes of variable values in an application currently running on a controller can cause unwanted behaviour of the controlled system.
Evaluate possible dangers before Writing or Forcing of variables and make appropriate security precautions.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

This command (category Online) is available in Online Mode.

It effects that one or more variables of the **currently active application** are permanently set to user-defined values in the PLC. The setting will be done both at the beginning and at the end of a cycle.

Sequence of processing in a cycle: 1.Read inputs, 2. Force values 3. Process code, 4. Force values, 5. Write outputs.

Note: Command "Force values [all applications]" which effects all applications of a project, is not per default inserted in any menu. (if needed, please see the Customization dialog)

Note: See in this context also the 'Write values' command for setting a defined value only once at the beginning of a cycle.

The forcing will remain active until it is explicitly suspended by the user for particular resp. for all variables, or until the application gets logged-out.

To prepare variables for forcing, the desired value must be defined in online mode in one of the following places which are used for monitoring :

- In a watch view defined in the project, containing a list of variables to be monitored (watch list).
- In the online view of the object within the declaration part of the respective editor.
- In the online view of the object within the implementation part of the FBD /LD/IL-editor.

A forced value is indicated with a F symbol.

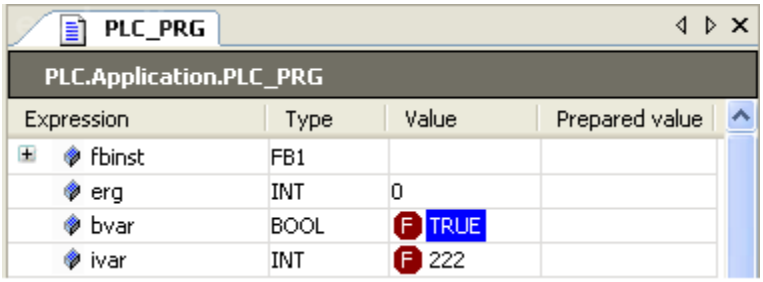


Abb. 199 Example of forced values in the declaration editor of a POU (online view)

Prepare Value Dialog

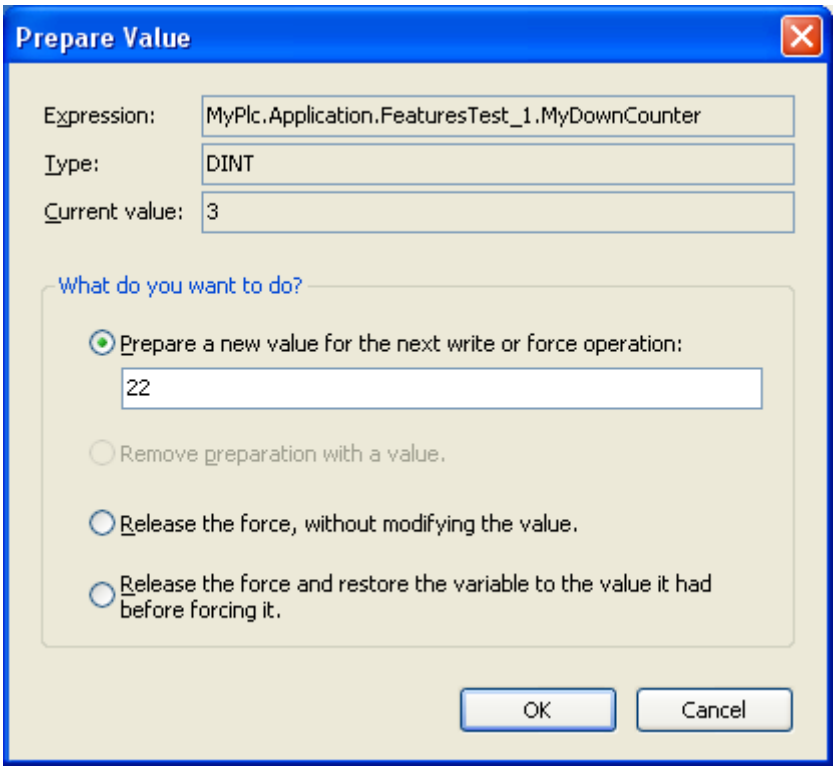
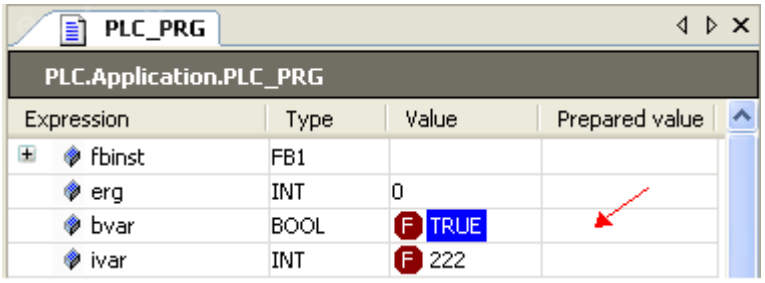


Abb. 200 Prepare Value dialog

This dialog is used to prepare a new value for a variable, to remove a prepared value, to release a forced variable or to release it and additionally reset its value to the one the variable was assigned to before forcing.

The dialog will open if you click in the 'Prepared Value' field of a currently forced value or in the inline monitoring field of the variable in the implementation part of the FBD/LD/IL-editor.



The following information on the currently concerned variable is displayed:

Expression: path of the variable: for example "PLC.Application.PLC_PRG.ivar"

Type: data type, for example "DWORD"

Current value: for example "TRUE" or "23"

Choose one of the following options concerning '**What do you want to do**' with the variable:

Prepare a new value for the next write or force operation: Depending on the data type of the variable you can enter a new number or string which you want to get assigned to the variable.

Remove preparation with a value. The prepared value for a variable will be removed.

Release the force, without modifying the value.: The variable will be marked as <Unforce> and thus is prepared to get the current value read from the PLC.

Release the force and restore the variable to the value it had before forcing it. The variable will be marked as <Unforce and restore> and thus is prepared to get the value it had before forcing.

According to the chosen option, after leaving the dialog with OK, in the 'Prepared value' field of the monitoring view the variable will show a new value or "<Unforce>" or "<Unforce and restore>". At the next 'Force values' resp. 'Write values' (for the first option) command the prepared values will be set.

12.22.19 Unforce values

Default Shortcut: <Alt> + <F7>

WARNING!
Extraordinary changes of variable values in an application currently running on a controller can cause unwanted behaviour of the controlled system.
Evaluate possible dangers before Writing or Forcing of variables and make appropriate security precautions.
Depending on the controlled system, damages to machines and parts could result, or even health and life of persons could be endangered.

This command (category Online) is available in Online Mode. It serves to release the forcing for all variables of the **currently active application**.

The variables will get the current value read from the PLC. This corresponds to option 'Release the force, without modifying the value.' which can be activated in the 'Prepare Value' dialog for a forced variable.

Note: Command "Unforce values [all applications]" which effects all applications of a project, is not per default inserted in any menu. (if needed, please see the Customization dialog)

12.22.20 Add All Forces to Watchlist

This command (category Online) is available in Online Mode when one of the watch views Watch 1, Watch 2, Watch 3 or Watch 4 is **active**. It serves to add all currently prepared or already forced variables of the active application to this watchlist. Notice however that this works only for **docked** watch views.

Notice also the possibility to use the watch view Watch all Forces, which automatically contains all currently prepared or forced values and additionally provides commands for releasing the forces.

12.22.21 Release Forcelist

WARNING!
Extraordinary changes of variable values in an application currently running on a controller can cause unwanted behaviour of the controlled system.
Evaluate possible dangers before Writing or Forcing of variables and make appropriate security precautions.
Depending on the controlled system, damages to machine and parts could result, or even health and life of persons could be endangered.

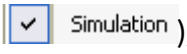
This command (category Online) is not per default available in any menu. It can be used to release the forcing for all variables of the **currently active application**.

The variables will get the current value read from the PLC. This corresponds to option 'Release the force, without modifying the value.' , which can be activated in the 'Prepare Value' dialog for a forced variable.

12.22.22 Simulation

This command (category Online) is available in Online Mode to switch on and off the **simulation mode** of the programming system. In simulation mode the application can be run and debugged on a "simulation target" which is always available within the programming system. So no real target device is needed to test the online behaviour of an application.

If the command is called from the Online menu, the currently **active application** will be concerned. If the command is called from the context menu when an application object is selected in the device tree, then the **selected application** will be affected, no matter whether it is set active.

When command **Simulation**' is activated (

), the device entry in the device tree will be displayed in italic letters and at the first login with the current active application you will be asked whether application "Sim.<devicename>.<applicationname>" should be created and loaded to the simulation target. No communication settings have to be done.

See the following picture for an example; command Online - Login has just been performed for the currently active application ExceptionTest:

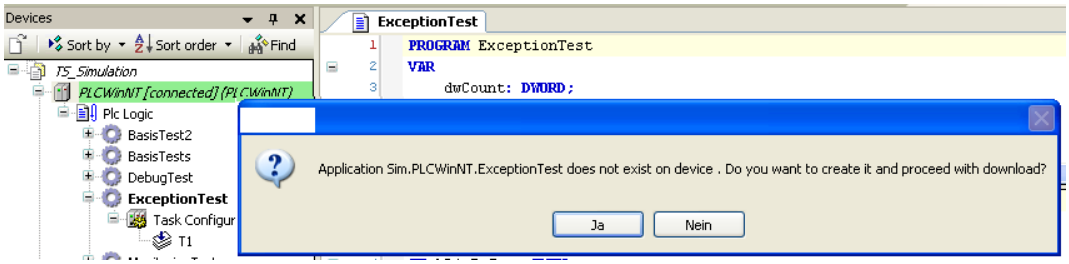


Abb. 201 Example, Login to the simulation target

After successful login you can use the respective online commands to test the application.

To switch off the simulation mode, first log out and then again perform command "Simulation". The checkmark in front of the command will disappear, the target entry in the device tree again will be displayed in normal letters and you can log in to a real device.

12.22.23 Flow Control

By default this command (category Online) is available in the Debug menu. It serves to switch on and off the Flow Control (Powerflow) function, which is supported for the language editors ST, FUP, LD and IL.

Activated Flow Control allows to track the execution of the application program.

1. The current values of the variables, thus also results of function calls and operations, will be displayed within the editor views, each at the exact position and time of execution. For comparison: The standard monitoring functionality just provides the value a variable has between two execution cycles.
2. Exactly those code lines resp. networks which are processed in the current cycle will be displayed color-coded.

Flow Control **can only be activated in online mode** and works exactly for the visible part of the currently active editor view. 'Flow Active' is displayed in the status bar only as long as the feature is activated and an editor view showing any flow positions (processed parts of the code) is opened.

Note: Activated Flow Control increases the run time of the application! If option Secure online mode is activated in the communication settings dialog, then a message box will appear, always when the user is going to activate Flow Control. One will be asked whether the function should really be switched on and one has the possibility to abort the activation.

Representation of Flow Control in the different language editors:

By default pale green is set as color for indicating flow control positions. (The color to be used can be modified in the ⇘ Text Editor options).

In all editors the current values of variables and of concerned inputs and outputs are displayed in little boxes like used for the standard monitoring. In processed code parts these boxes will be displayed in the color configured for flow control, in non-processed parts the monitoring boxes appear white with grey face. Note that at a non-processed code position the displayed value is a "normal" monitoring value, that is the value between two task cycles.

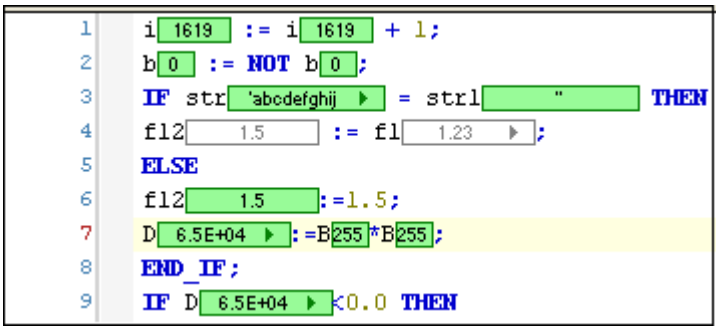


Abb. 202 Example: Flow Control in ST editor

In network editors the executed networks are marked by bars in 'flow control color' at the left margin.

In LD the currently processed connection lines basically are drawn in **green** (resp. the currently set 'flow control' color), others in grey. The current value on the connection is also indicated: TRUE by **thick blue**, FALSE by **thick black** lines, no known or analog values by thin black. This might result in dashed lines combining the respective information.

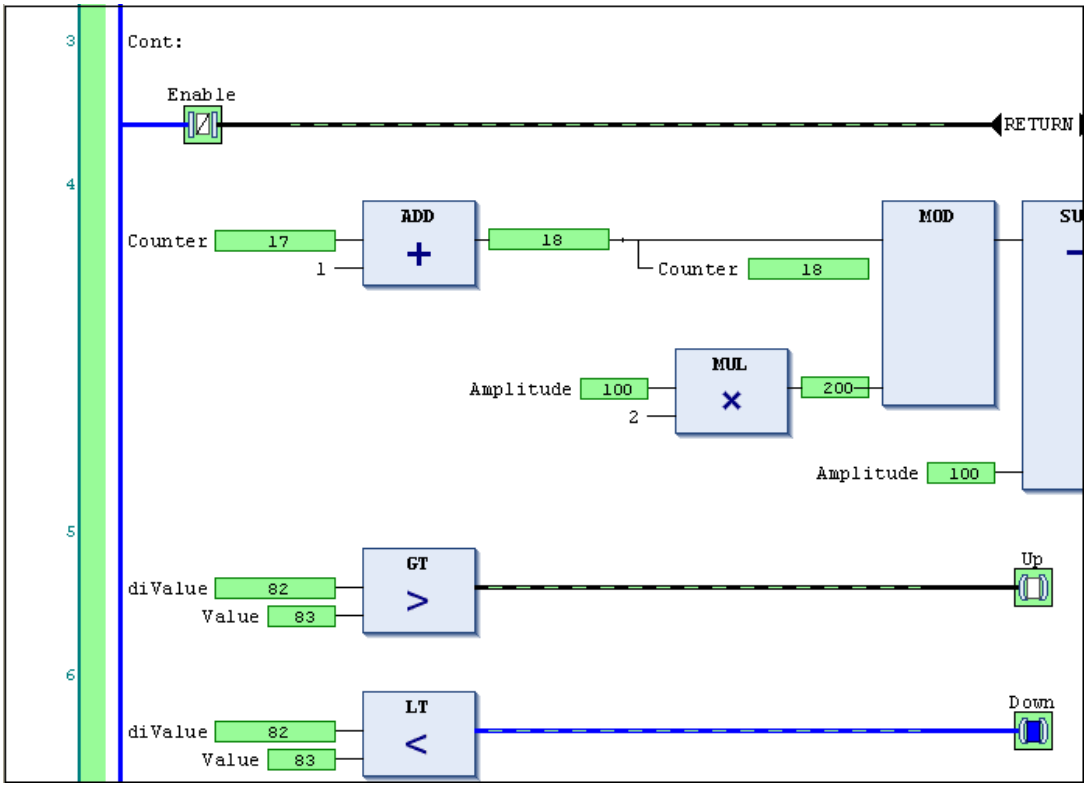
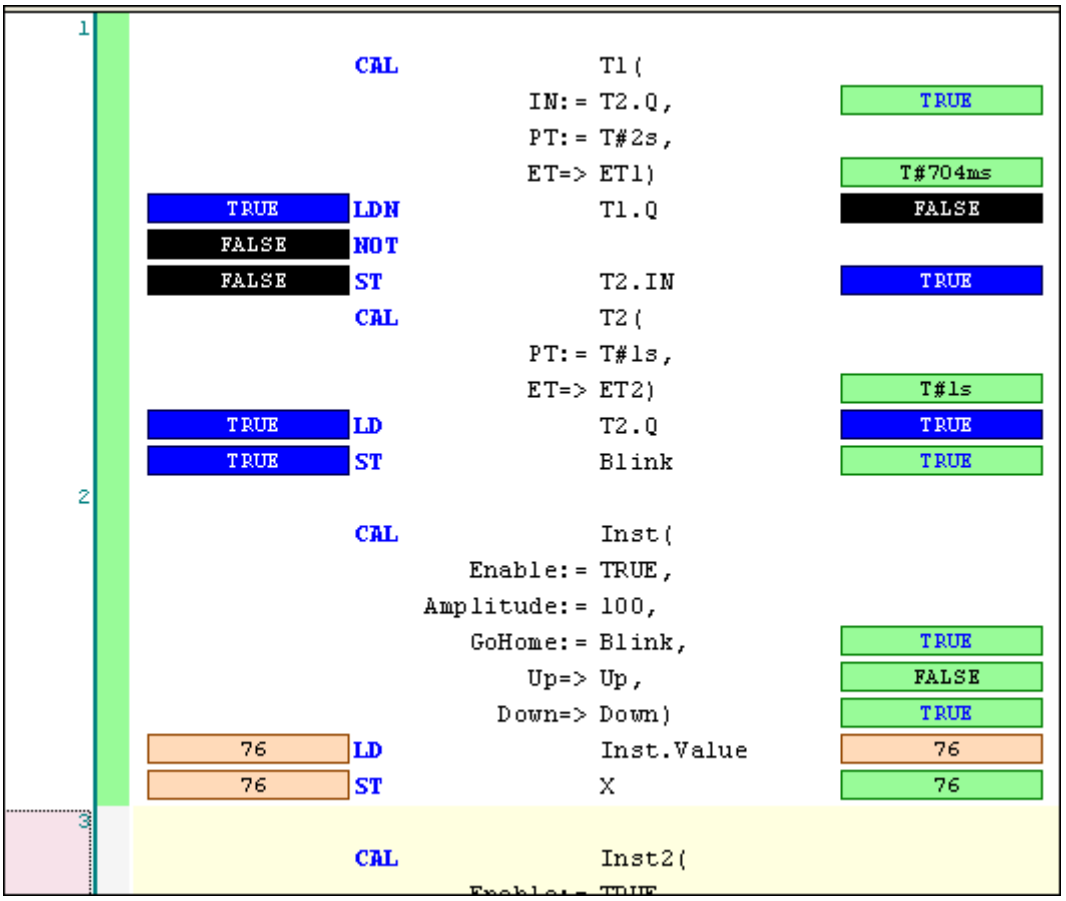


Abb. 203 Flow Control in LD editor

In IL for each instruction line two boxes are used to indicate the current values. One left to the operator showing the current accumulator value, one right to the operand showing the operand's value.



Example: Flow Control in IL editor

Regard the possibility to **write values also in flow control mode. Forcing values is not possible!** By a double-click on the value box the Prepare Value dialog opens, where you can enter the desired value.

12.22.24 Display Mode

This option commands (category Online) can be used to choose which of the following formats should be used for the monitoring display. Perform a mouse-click on the desired option in the Display Mode submenu. The currently set option is marked by a tick.

- Binary
- Decimal
- Hexadecimal

12.22.25 Connect to <device>

This command (category Online) can be used to set up a connection to the device which is currently selected in the device tree. As a precondition the communication settings of the device must be configured properly.

12.22.26 Disconnect from Device

This command (category Online) can be used to disconnect from the device which is currently connected and selected in the device tree.

12.22.27 Reset origin device

This command (category Online) is available in the context menu in offline and online state, when a programmable device is selected in the device tree. It sets back the device to the original ("shipping"`) state, as far as any application(s), boot project files and remanent variables on the device will be deleted.

12.23 Preferences

The commands of category 'Preferences' are available for customizing the user interface and set various »PLC Designer« and project options.

12.23.1 Preferences

Preferences commands - Frame

The 'Frame' plug-in provides for command category 'Preferences' commands (see below) which serve to customize the user interface, to configure certain options for user interface components which are provided via plug-ins also and to configure directories which are necessary for the project.

By default these commands are part of the Tools menu, but regard that the Tools menu itself also is configured in the Customize dialog (regard category 'Preferences').

The commands:

- Options
- Customize

Options

The Options dialog, provided with the Frame plug-in, contains sub-dialogs for configuring the behaviour and display of the »PLC Designer« programming interface. The default settings are defined by the current profile.

The dialog is opened via the command '**Options**' (category 'Preferences').

The current settings, confirmed with OK, will be saved on the local system, overwriting the defaults which are provided with an installation of the programming system.

The default categories:

- CFC Editor
- »PLC Designer« 2.3 converter
- Declaration Editor
- Device editor
- FBD, LD and IL editor
- Features
- International Settings
- Libraries
- Load and Save
- SFC
- SFC editor
- SmartCoding
- Source Control
- Syntax Highlighting
- Text Editor
- Visualization

Options, Load and Save

This sub-dialog of the options dialog allows settings concerning the behaviour of »PLC Designer« at loading and saving a project.

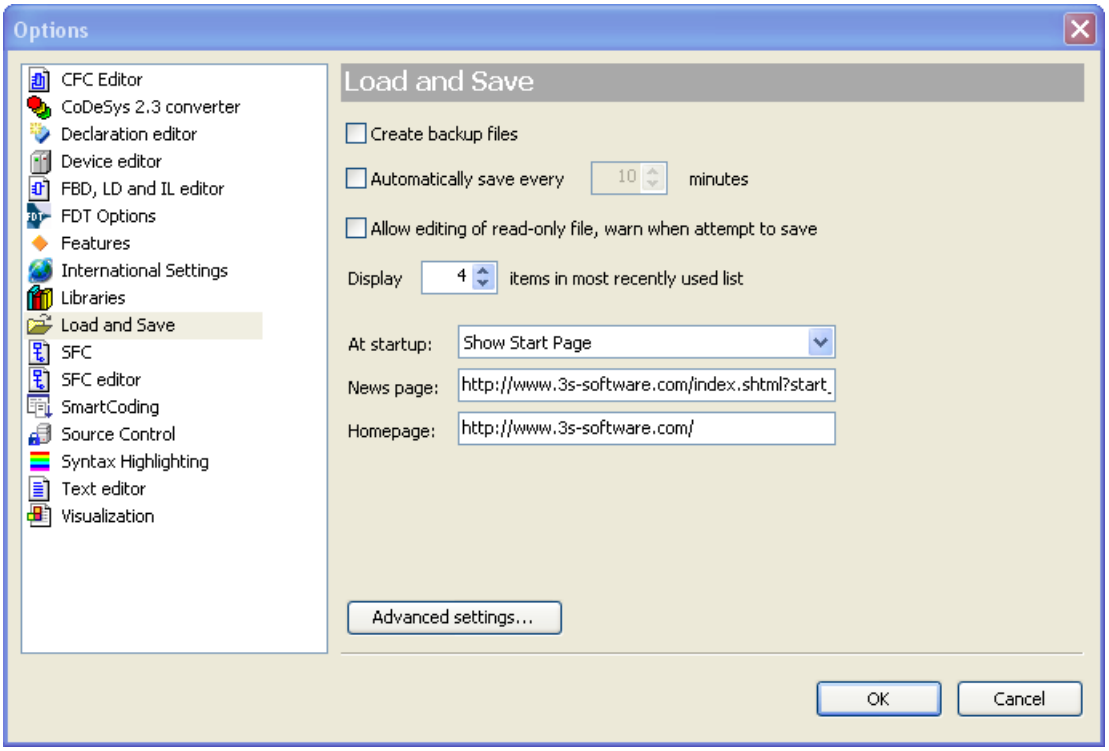


Abb. 204 Load and Save options dialog

Create backup files: If this option is activated, at each saving the project will not only be saved in <*projectname*>.*project*. but also copied to a file <*projectname*>.*backup*. If needed you can rename this backup-file and re-open in »PLC Designer«.

Automatically save every ... minutes: If this option is activated, the project will be automatically saved in the defined time intervals to a file <*projectname*>.*autosave* in the projects directory. After a non-regular termination of the programming system you might reload this autosave-file.

If you regularly save or close the currently opened file, the associated auto save file will be deleted, in case of an irregular abortion it will be kept.

If you reopen a project for which an appropriate auto save file is found, the **Auto Save Backup** dialog will appear, where you can decide whether to reopen the auto save project or the last version which was saved by the user:

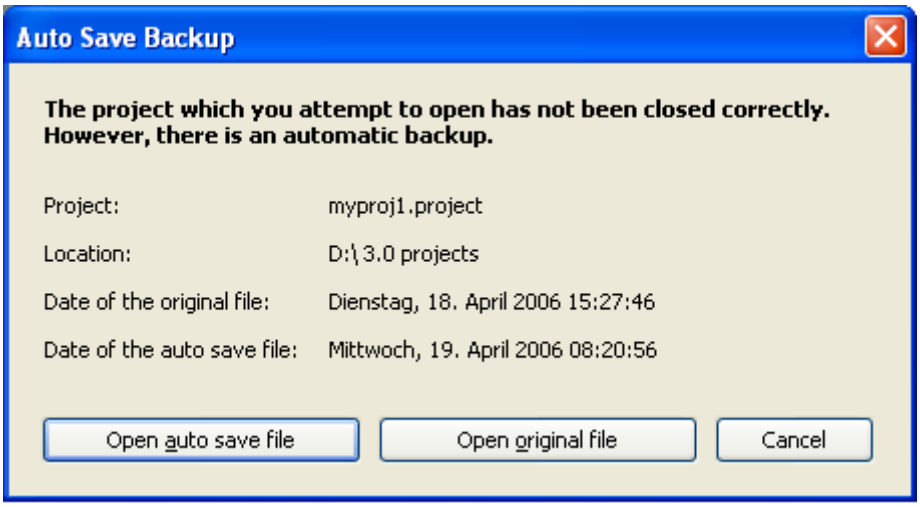


Abb. 205 Auto Save Backup dialog

Open auto save file: The auto save file <*projectname*>.*autosave* will be opened and marked as modified.

Open original file: The last version of the project saved by the user will be opened.

Allow editing of read-only file, warn when attempt to save: You can edit a project which is restricted to read-only access, but nevertheless an alert (warning) will be generated when you are going to save the file.

Display ... items in most recently used list: Define here how many of the recently opened projects should be displayed in the 'Recent Projects' list (File Menu).

At startup: Define here, what should happen when »PLC Designer« gets started:

- Load last loaded project: The project which was edited last, will automatically be opened.
- Show "Open Project" dialog box: Corresponds to command 'Open Project'.
- Show "New Project" dialog box: Corresponds to command 'New Project'.
- Show empty environment: The programming system will be started without opening or creating a project.
- Show Start Page: The 'Start Page' view will be displayed. Corresponds to command Start Page

News page: URL of the page which is displayed in the 'News' area of the Start Page

Homepage: URL of the page which will be displayed on command 'Visit the homepage' (Start Page) or on '3S Homepage' (Help menu).

Advanced Settings...: Button for opening the same-named dialog:

Project Compression: See the dialog for further information on the three possible compression levels. Per default the least compression is set.

Load behaviour: If this option is activated (default and recommended) the loading of libraries and compile information will be done in the background, while you already can start working on the project.

Options, Text editor

This sub-dialog of the options dialog allows settings concerning the editing in a text editor:

1. Editing: Definition of Undo, tabulator, indenting, folding etc. in the editing area of the text editor.

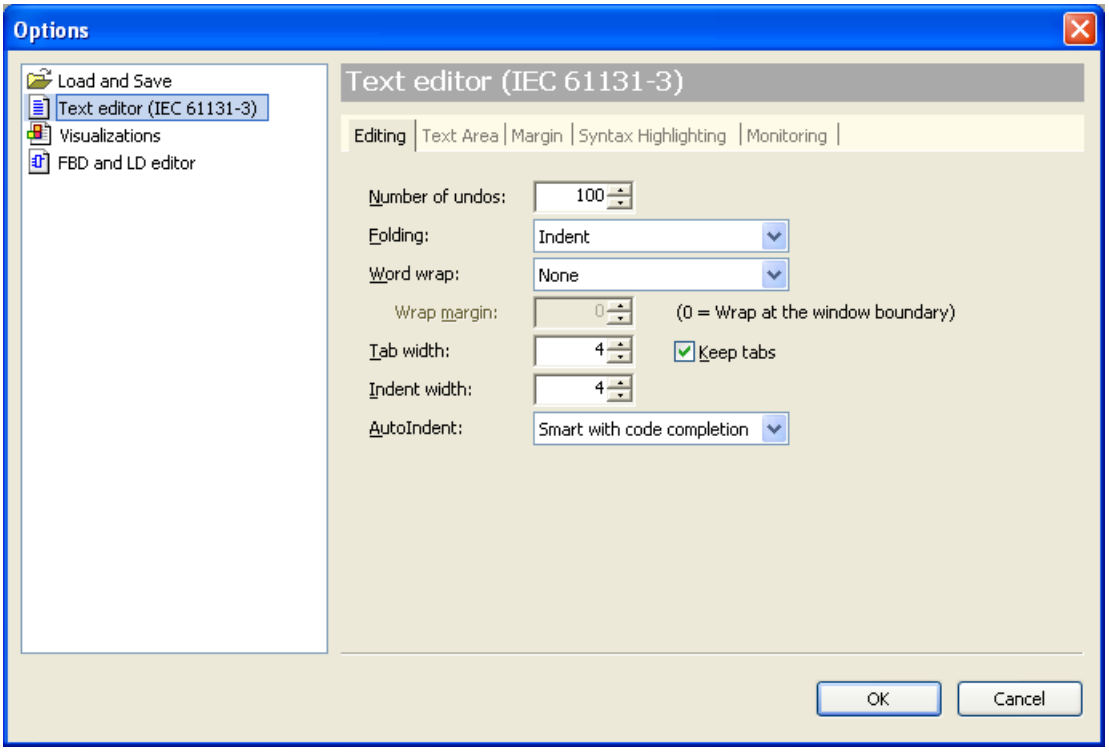


Abb. 206 Options dialog, category Text Editor, Editing

Number of undos:	Define how many editing steps should be saved, so that they can be made undone by the Undo function.

Folding:	Define whether the code should be viewed structured by folds. This means that sections specified by indented position resp. marked by a special comment are hidden in a fold which can be opened or closed via by a mouse-click on the plus- resp. minus-sign on the left side of the folds header line. The options:
	None: No folding is done.
	Indent: All lines indented to the previous one will be put in a fold, which is headed by the preceding line.
	Examples: The lines between VAR and END VAR resp. between IF and END IF are indented, thus are packed in a fold indicated by a minus-sign in open and by the plus-sign in closed state:
	fold opened
	`1 IF a:=1;`
	`2 THEN`
	`3 b:=0;`
	`4 ELSE`
	`5 b:=1;`
	`6 END_IF`
	`7 c:=c+1;`
	fold closed
	`1 IF a:=1; [4 lines]`
	`6 END_IF`
	`7 c:=c+1;`
	Abb. 207 Opened and closed folds
	Explicit: The code section to be packed in a fold must be marked explicitly by comment lines: Before the section in a comment enter three opening braces "{{{", after the section in a comment enter three closing braces "}}}". The comment may contain further text also.
	Examples for defining a folding section
	`1 IF b THEN`
	`2 /// {{{`
	`3 IF a THEN`
	`4 c:=TRUE;`
	`5 END_IF`
	`6 /// }}}`
	`7 END_IF`
	`2 /// commentcomment {{{ commentcomment`
	`3 IF a THEN`
	`4 c:=TRUE;`
	`5 END_IF`
	`6 /// }}}`
	`7 END_IF`
	Explicit folds defined by special comments

Word wrap:	**None**: The line can be filled endlessly. **Soft**: The lines will be broken at the boundary of the window. **Hard**: The lines will be broken after the number of characters defined by **Wrap margin.** (A value of 0 means that the lines are wrapped at the window boundary.)
Tab width:	Tab width in number of characters.
Indent width:	Indent width used in case of AutoIndenting (see below); that is the number of spaces at the beginning of a line.
AutoIndent:	**Block:** **None**: No automatic indenting. Editing always starts at the left margin of the editors text area. **Smart**: Code lines following a keyword (for example VAR or IF) are automatically indented according to the Indent width defined above. **Smart with code completion**: When you enter a keyword like for example VAR or IF automatically the associated code lines following below are indented according to the Indent width defined above and additionally the terminating keyword is added (for example END_VAR or END_IF).
Keep tabs	If activated: A space which has been entered using the tab key according to the defined Tab width (see above) will not be resolved to single character spaces afterwards. So if the tab width gets changed later, the existing tab spaces will be corrected correspondingly.

2. Text area: Color and font definitions for the editing area of the text editor.

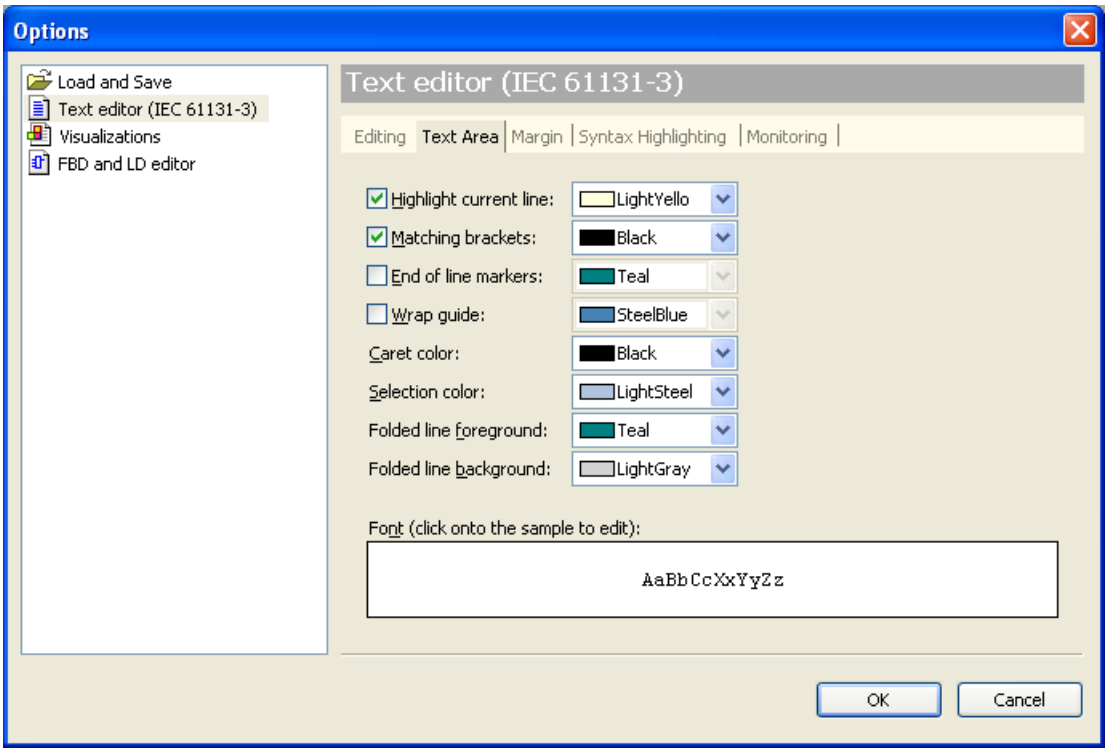


Abb. 208 Options dialog, category Text Editor, Text Area

You can activate the following options in order to have some visual assistance during editing:

Highlight current line:	The line where the cursor is placed, is highlighted with the color defined in the selection list.
Matching brackets:	If the cursor is placed before or after a bracket "(" or ")" in a code line, the corresponding closing resp. opening bracket will be indicated by a rectangle which is colored as defined in the color selection field. Also, if the cursor is placed in the first or last line of a bracket scope*, the corresponding line at the end resp. start of this scope will be indicated also by a rectangle of this color. * A bracket scope contains all lines between corresponding keywords (for example IF and END_IF, see also tab 'Margin')
End of line markers:	The end of each editor line will be marked by a small dash of the defined color which is placed behind the last position which is occupied by a character or a white-space.
Wrap guide:	If soft or hard word wrap is activated (see above, tab Editing), the defined wrap margin will be indicated by a vertical line in the defined color.
Caret color:	Color of the cursor caret
Selection color:	The currently selected area will be highlighted with this color.
Folded line foreground:	The header of a closed, folded section of code lines will be written in this color.
Folded line background:	The header of a closed, folded section of code lines will be highlighted in this color.
Font:	Choose the font to be used in the text editor. By a mouse-click on the sample field you get the standard dialog for setting the font.

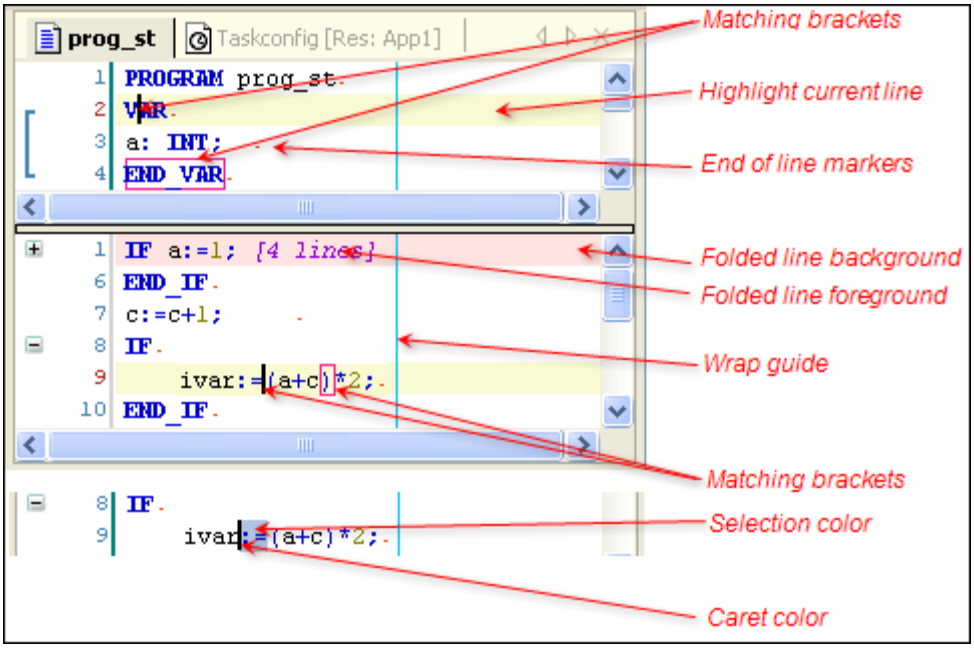


Abb. 209 Examples for the settings in the Textarea Dialog

3. Margin: Definition of colors, font, mouse-click definitions concerning the area left to the editing area of the text editor window.

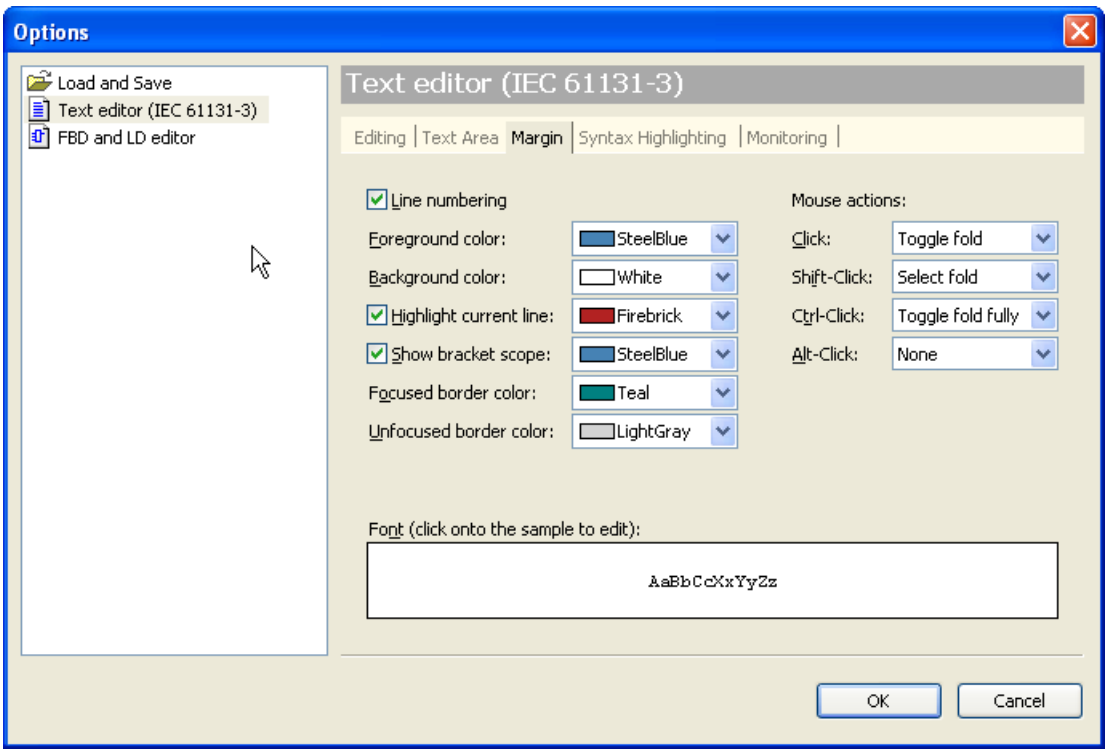


Abb. 210 Options dialog, category Text Editor, Margin

The following options concern the left margin of the editor window, which is separated by a vertical line to the editing area:

Line numbering:	Line numbers will be displayed in the declaration and in the implementation part of the window, each starting with 1 in the first line.
Foreground color:	Color of the line numbers.
Background color:	Color of the margin background.
Highlight current line:	Color of the line number of the currently edited line.
Show bracket scope:	A bracket scope contains lines between corresponding keywords, like e.g between IF and END_IF. You can activate this bracket scope function by option 'Matching brackets' in category 'Text area'. If the cursor then is placed before, after or within a corresponding keyword, the bracket scope will be indicated by a bracket in the margin area.
Focused border color:	Color of the vertical separator line on the right side of the margin area in the currently active part of the editor.
Unfocused border color:	Color of the vertical separator line on the right side of the margin area in the currently not active part of the editor.
Mouse actions:	Define which of the following actions should be performed when you perform the associated type of mouse-click on the plus- resp. minus sign before the header line of a fold: **None**: No action. **Select fold**: The lines of the fold area will be selected. **Toggle fold**: The first level of the fold will be opened resp. closed. **Toggle fold fully**: All levels of a fold will be opened resp. closed. Types of mouse clicks: Click, Shift-Click, Ctrl-Click, Alt-Click

4. Monitoring: Activating/Deactivating and configuration of the display of inline monitoring fields.

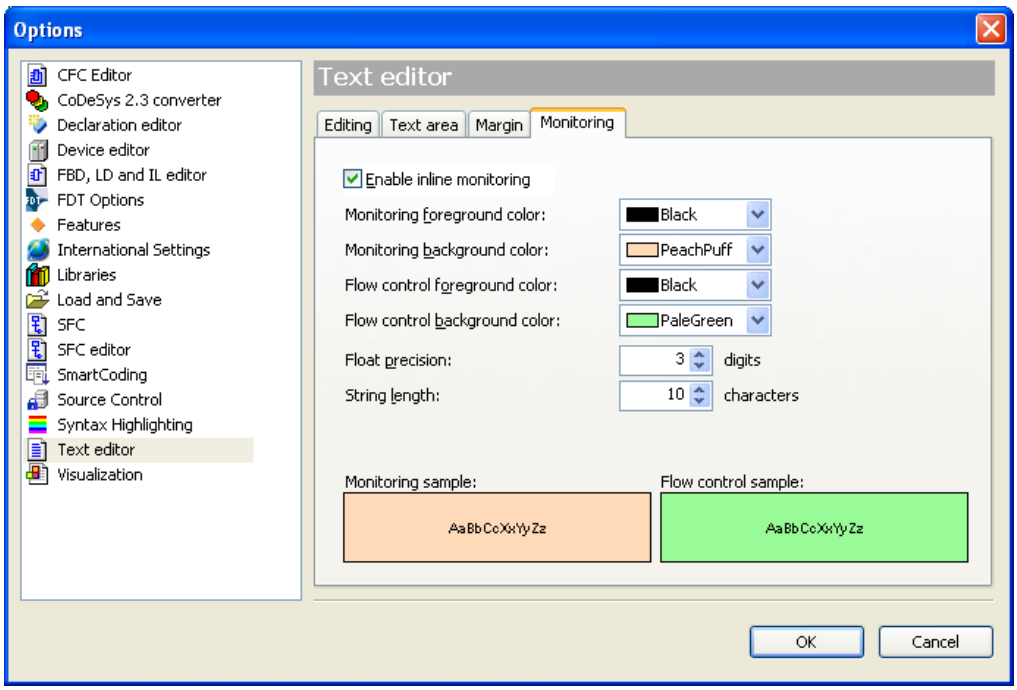


Abb. 211 Options dialog, category Monitoring

Enable inline monitoring:	Activate resp. deactivate inline monitoring
Monitoring Foreground color:	From the selection list choose a color for the writing in the monitoring fields.
Monitoring Background color:	From the selection list choose a color for the background of the monitoring fields.
Flow control foreground color:	From the selection list choose a color for the writing in the monitoring fields at flow positions.
Flow control background color:	From the selection list choose a color for the writing in the monitoring fields at flow positions.
Float precision:	Define how much decimal places (digits) should be maximum displayed in case of floating values.
String length:	Define the number of characters maximum displayed in case of a string value.
Sample:	Preview of the currently defined monitoring settings.

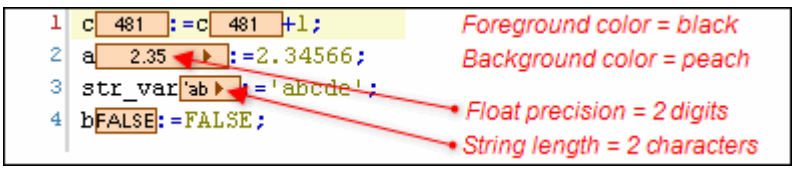


Abb. 212 Example of inline monitoring in ST editor:

Options, International Settings

This sub-dialog of the options dialog allows the following settings concerning the language used in the user interface and help system.

User interface language:

Same as Microsoft Windows: If this option is activated, the language used in the user interface will be that which is currently set by Microsoft Windows on your computer.

Specific culture: If this option is activated, the language currently selected from the list will be used.

Note: Changing the user interface language will not be effective until »PLC Designer« is restarted. Some components may not be available in the selected language and will appear in their default culture, which typically is English.

Online help language:

Same as user interface language: This option is set by default. The Online Help will be displayed in the language which is set for the user interface. If no help is available in this language, English will be used.

Specific culture: If this option is activated, the language currently selected from the list will be used.

Options, SmartCoding

This sub-dialog of the options dialog allows some settings for making coding more pleasant. This concerns the Smart Coding functions like for example AutoDeclaration or Input Assistant.

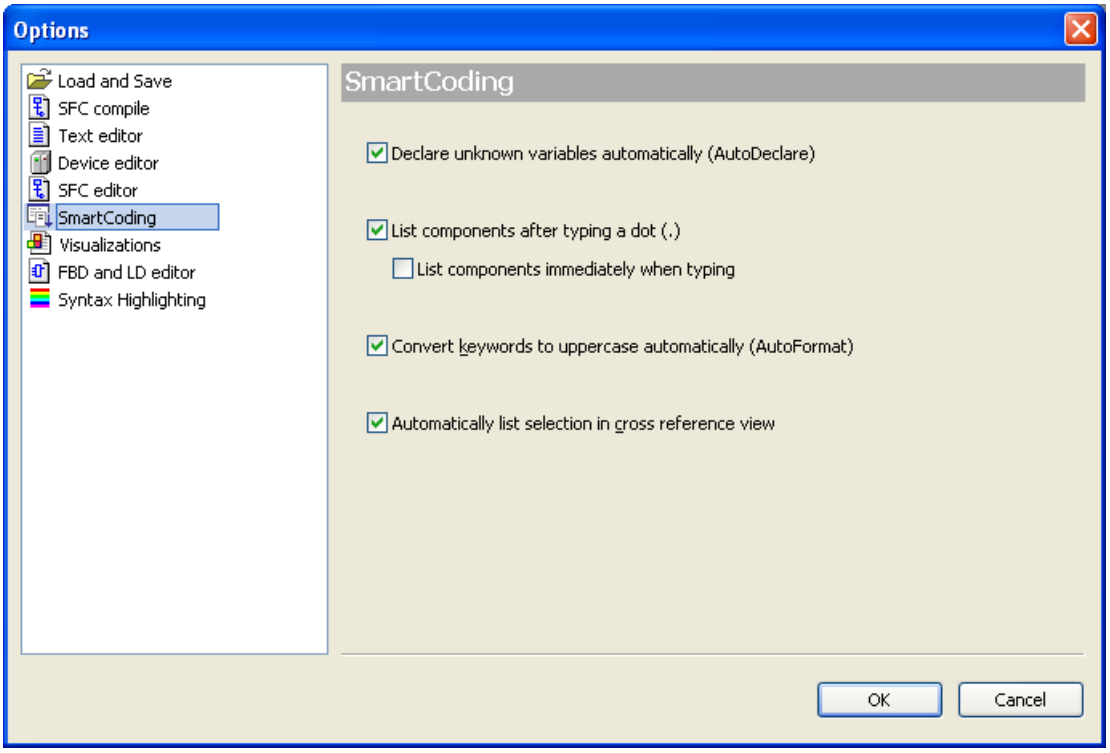


Abb. 213 SmartCoding options dialog

Declare unknown variables automatically (AutoDeclare): If this option is activated, the Auto Declare dialog will open automatically when you enter a not yet declared identifier in any programming language editor.

List components after typing a dot (.): If this option is activated, the "List Components" is activated. This means that when entering a dot (".") in an editor at a position where an identifier is expected, you will get a selection list with possible entries.

List components immediately when typing: If activated: As soon as you enter any character in the editor, a list will open which contains all available identifiers and operators. This is also a kind of "List Components" functionality. Depending on which character sequence you have entered, automatically the first entry of this list matching this character sequence will be selected. Anyway you can select each of the list items and by <Return> place it at the cursor position.

Convert keywords to uppercase automatically (Autoformat): If this option is activated, all keywords used in text parts of editors automatically will be written in capital letters. Example: If you enter "bVar:bool;" this will be converted to "iVar:BOOL;"

Automatically list selection in cross reference view: If this option is activated, the Cross Reference View always automatically lists the references of the variable currently selected in the active editor.

Differences to the Cross Reference List in »PLC Designer« V2.3:

- The Cross Reference View can be displayed parallel to the editor view.
- No compilation of the project is necessary in order to get the references displayed.
- Also component accesses, enumeration constants and user-defined types are found.
- An automatic display of the references for the currently selected item can be activated.

Options, Syntax Highlighting

This sub-dialog of the options dialog allows the definition of colors and font style for the particular text items (for example operands, pragmas, comments etc.) in an editor window.

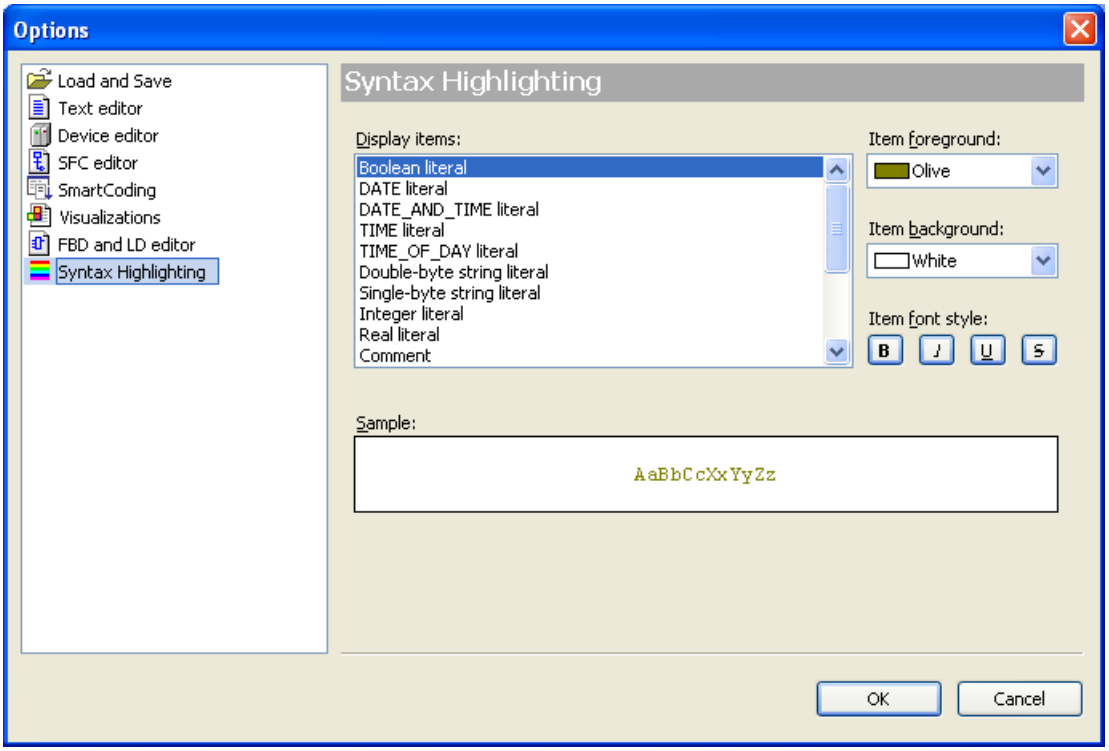


Abb. 214 Options dialog, category Syntax Highlighting

Define here which color and font style should be used for displaying the particular items in the text editor window. Select a item type from the **Display items** selection list, then select the desired colors for the **Item foreground** and **Item background**) and define the **Item font style** by a mouse-click on one or several of the respective buttons for bold (**B**), italic (*I*), underlined (U) and strike through (S).

A sample view of the current settings will be displayed in the **Sample** window.

Options, Libraries

This sub-dialog of the options dialog serves to manage the "**mappings**" of **library references** which will be applied during the conversion of an "old" project. As long as no mapping is stored for a particular library here in the options dialog, you will be asked for one each time when you open an old project including this library.

See Open Project for a description on opening and converting old projects.

See Library Management for a description on library management.

A "mapping" defines how the library reference should look after the conversion of the project. There are three options:

- The library reference should be kept, that is the library gets converted to the new format (*.library) and installed in the local library repository
- The library reference should be replaced by another one, that is one of the currently installed libraries is selected to get included instead of the previous one.
- The library reference should be removed, the library will not be included in the converted project.

If you decide to save the mapping, it will be added to the Options dialog in category 'Libraries':

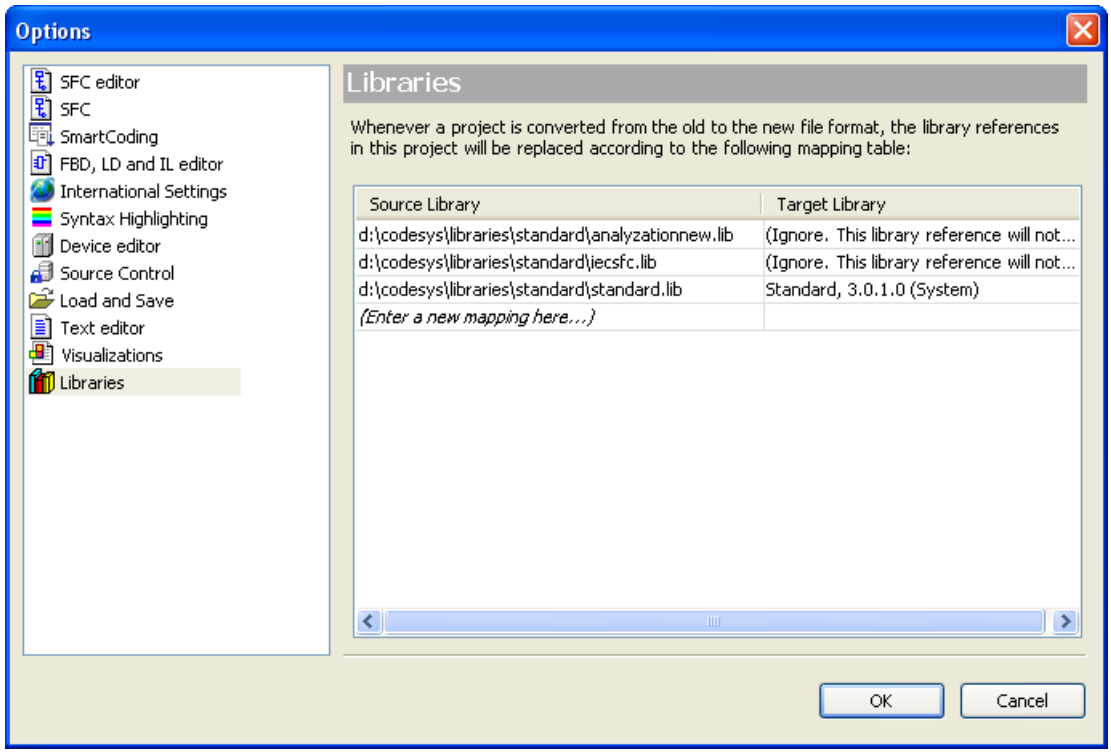


Abb. 215 Options dialog, category Libraries

All mappings which are listed here, will be applied at the next conversion of an old project and its library references. Thus you do not need to repeat the mapping definition when the same library is referenced in multiple projects to be converted.

The "mappings" are displayed in a line each:

Source Library: path of the library which was referenced in the project before conversion,

Target Library: name and location of the new library which should be referenced in the converted project. If the reference should be removed, this is indicated by the string "(Ignore. This library reference will not be converted.)".

You can edit the mappings list: For this purpose perform a double-click on the respective field, or select the field and press <Enter>.

If you edit a Source Library entry, the ... button will open the standard browse dialog, where you can browse the file system for an "old-formatted" library.

If you edit a Target Library entry, the **Set Target Library** dialog will open.

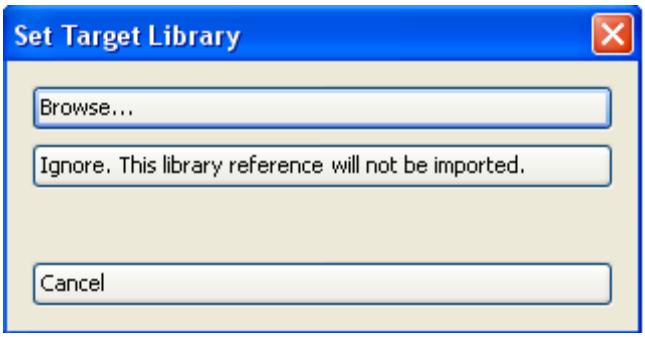


Abb. 216 Set Target Library dialog

To select a "new-format" library use the **Browse** button. The **Select Library** dialog will open, where you can search one of the currently installed libraries from the local library repository. You can sort by company and category as known from the Library Repository dialog.

To define that the given Source Library always should be removed from the project, when this gets converted to the new format, use button **Ignore...**

If you want to **add a new entry** to the mappings list, you can do this also directly here in the dialog by editing edit the last line "*(Enter a new...)*".

12.23.2 Options, »PLC Designer« V2.3 converter

In this sub-dialog of the options dialog the "conversion mapping" for »PLC Designer« V2.3 projects is defined.

Tab 'Devices' describes the **conversion of the used devices** (target systems):

As long as no "mapping" is stored for a particular device here in the options dialog, you will be asked for one each time when you open an old project using this device.

See Open Project for a description on opening and converting old projects.

See Device, Device Tree for a description on device handling.

A "device conversion mapping" defines how the device reference should look after the conversion of the project. That is it assigns a "**Target device**" (device in new project) to the "**Source device**" (device in old project). There are two options:

- The source device will be replaced by the target device, that is another, currently installed and device, in the device tree of the new project.
- The source device will not be converted (will be ignored), that is will not be available in the device tree of the new project. Thus also application-specific objects, like for example the task configuration, will not be transferred to the new project.

A mapping can be defined and stored ...

- ... during the conversion of an appropriate (device-including) old project (see the description of the 'Open Project' command)
- ... or it can be directly entered in the converter options dialog in the Devices tab:

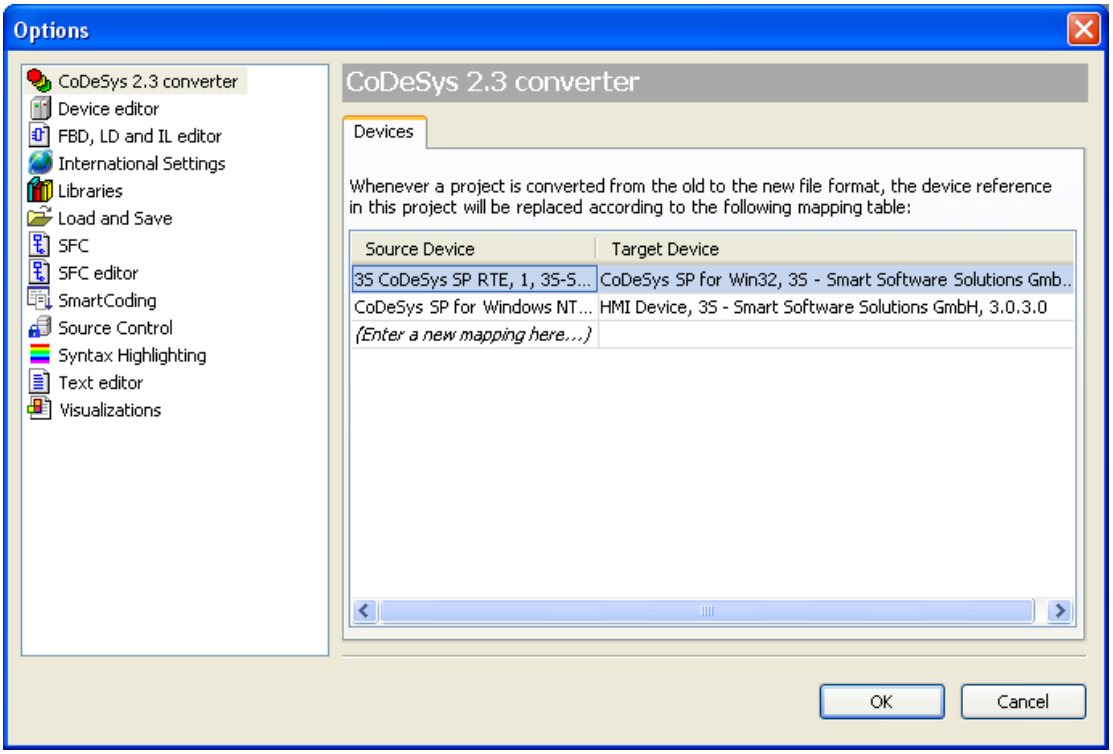


Abb. 217 Options dialog, category »PLC Designer« 2.3 converter

To edit a field in the mapping table, perform a double-click on this field or select the field and press <Enter>. Assign a target device to each source device:

For entries in the 'Source device' column the dialog '**Select Source device**' will open, where all devices are listed which can be handled by the converter. Select the desired one to get it entered in the source device field.

For entries in the 'Target device' field the '**Select Target device**' dialog will open, where you can choose one of the currently installed devices. The usage of the dialog (sorting by vendor and categories) corresponds to that of the device repository dialog. (Also see the description of the device conversion during 'Open Project') . If you choose **'None'**, the source device will **not be available in the new project**, that is also application specific objects like for example the task configuration will not be converted!

All device mappings which are listed here in the options dialog, will be applied at the next conversion of an old project and its device references. Thus you do not need to repeat the mapping definition when the same device is referenced in multiple projects to be converted.

12.23.3 Options, CFC Editor

This sub-dialog of the options dialog provides settings concerning the editing in a CFC (Continuous Function Chart) editor.

The settings will immediately be applied to the currently opened editor views as soon as they have been confirmed (and the dialog closed) with **OK**.

Settings:

Enable AutoConnect: If this option is activated the following is true: When you drop CFC elements somewhere on the canvas, unconnected pins that are touching each other, will be connected automatically. This might be useful for quick editing, however take care not to create connections accidentally when moving elements around!

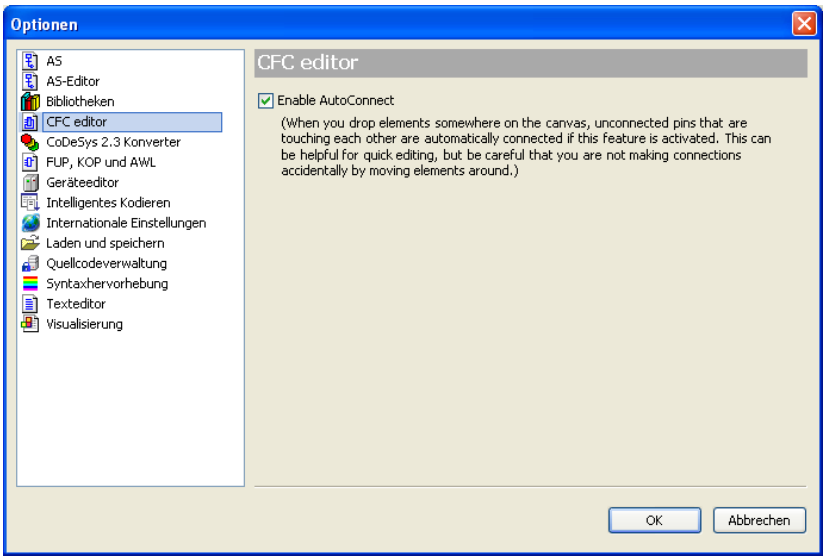


Abb. 218 Example of step time displayed in online mode

12.23.4 Options, Declaration Editor

This sub-dialog of the options dialog contains some default settings on the appearance of the declaration editor.

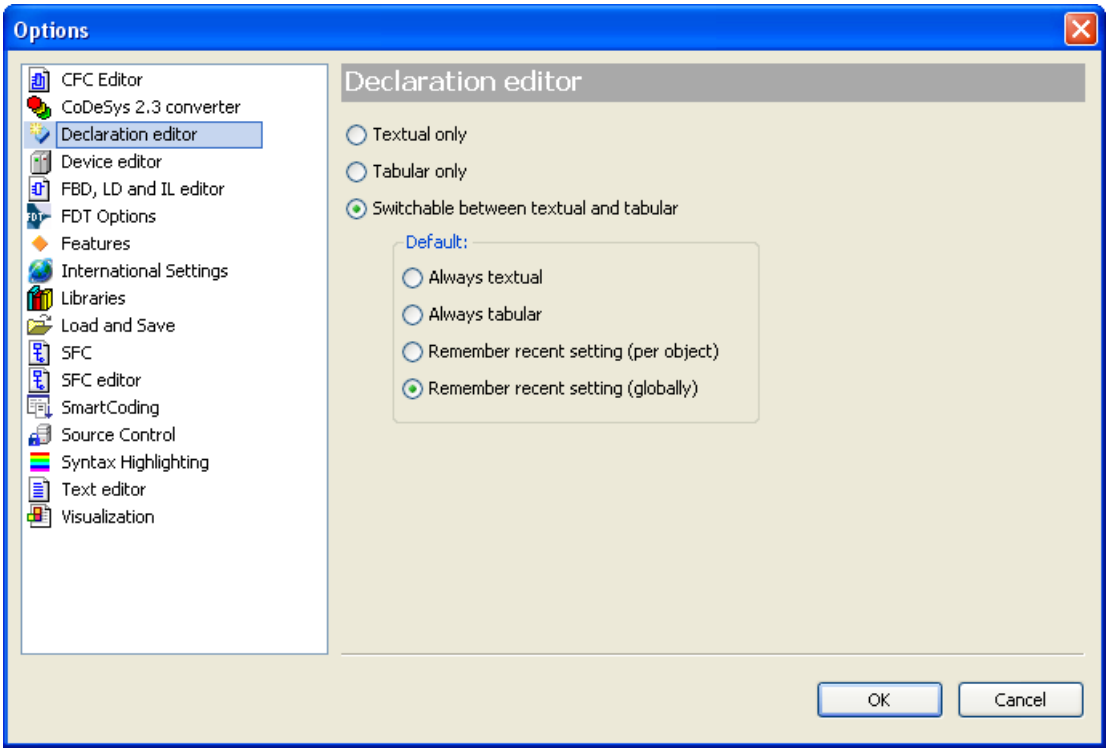


Abb. 219 Declaration Editor options dialog

Define, which views of the declaration part should be available when an object is opened for editing:

Textual only

Tabular only

Switchable between textual and tabular: If this setting is activated, two buttons will be available in the declaration editor window for switching between the textual and the tabular view of the declaration part. In this case choose also one of the following settings, which defines in which of the both formats by **Default** the declaration editor at first will appear, when an object is opened for editing:

- **Always textual**
- **Always tabular**
- **Remember recent setting (per object)**: When reopening an object, the declaration editor will be opened in the same format as it had for this object during last editing.
- **Remember recent setting (globally)**: When opening any object, the declaration editor format recently used in any object will be used.

12.23.5 Options, FBD, LD and IL editor

This sub-dialog of the options dialog allows settings concerning the editing in the FBD and LD editor:

Category View:

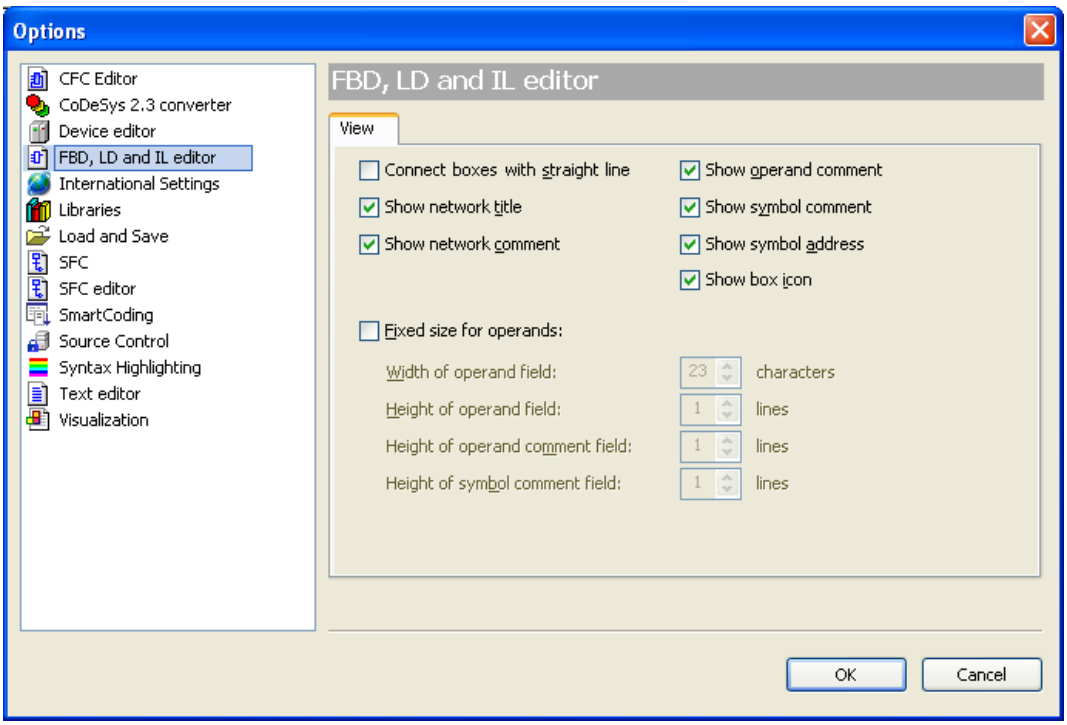
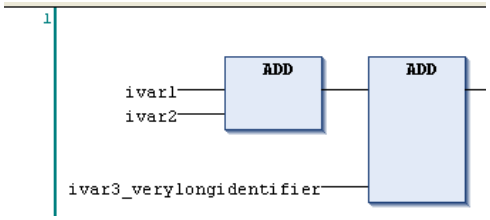
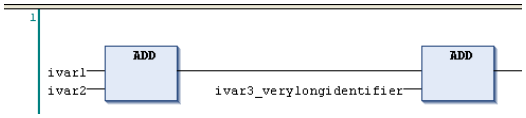
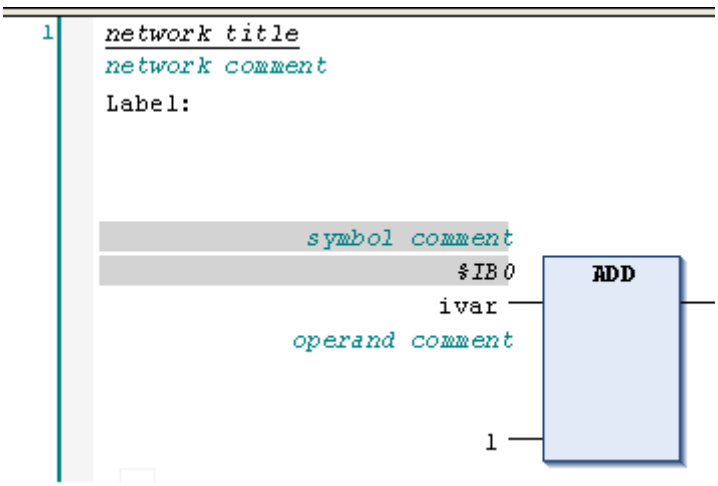
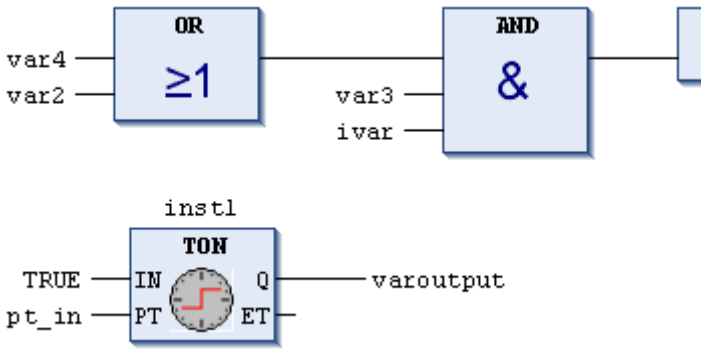


Abb. 220 FBD, LD and IL view options dialog

Connect boxes with straight lines:	If this option is activated, the network components will be arranged in a way that the lines between are of a fix, short length, thus as far as possible reducing the horizontal space required for displaying the networks. This might effect that boxes are enlarged in height, e.g. to provide enough space for input and output elements. If the option is not activated, the elements will keep their standard size and the connection lines adapt to the need of space. Example Option activated:  Option deactivated: 
Show network title	The network title - if defined - is displayed in the upper left corner of a network.
	 Components in the editor view, example: FUP network
Show network comment	The network comment - if defined - is displayed in the upper left corner line of a network. If the network title is visible also, the comment will appear in the line below the title.
Show operand comment	The comment which might be assigned to a variable in the implementation part of the editor will be displayed. The operand comment only refers to the current place of usage of the variable, in contrast to the "symbol comment", which is defined at the declaration of a variable.
Show symbol comment	For each symbol (variable) which has got a comment in its declaration, this comment will be displayed above the symbol's identifier. Regard that additionally or instead of also a local "operand comment" can be assigned.
Show symbol address	For each symbol (variable) the assigned address will be displayed above the symbol's identifier.

Fixed size for operands	Not yet implemented, currently except for the network comment only single-line display is possible: If this option is activated, the following parameters determine the display size of the information fields assigned to an operand: **Width of operand field [characters]:** Maximum number of characters which are displayed of an operand name. **Height of operand field [lines]:** Maximum number of lines displayed of an operand name. **Height of operand comment field [lines]:** Maximum number of lines available for an operand comment. **Height of symbol comment field:** Maximum number of lines available for an operand symbol.
Show box icon	If a function block or a function available via a library or via the object properties is provided with an icon (bitmap), this icon will be displayed in the box in the FBD and LD editor. Also the standard operators are provided with icons.  Boxes with icons for standard operators in a FBD network:

12.23.6 Options, Features

This subdialog in the Options dialog provides the possibility to adapt the selection of available features in the currently used programming system environment. In the upper part you can switch on and off the particular features, in the lower part you can switch between the predefined sets on a whole.

Note: Some settings restrict the selection of object types that can be newly created in a project. However, if the user opens a project that already contains objects of currently not insertable types, these nevertheless will be visible and fully functional.

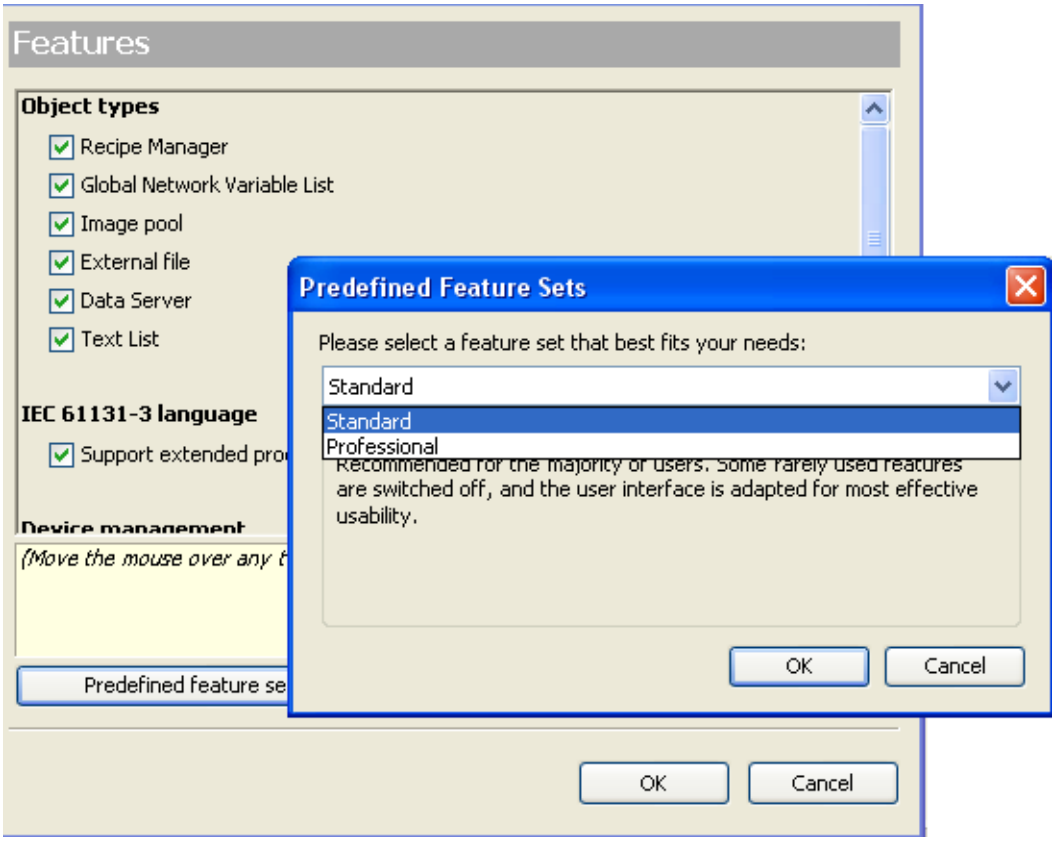


Abb. 221 Dialog Options, Features

Predefined Features Sets

Currently there are two predefined sets of features, which can be exchanged via button **Predefined Features Sets:**

Standard: Some rarely used features are switched off and the user interface is adapted for most effective usability. Recommended for the majority of users.

Professional: All features are available and the user interface is more complex in order to unveil all possibilities of the system.

Particular Features

Alternatively, each of the features which are only available with the Professional set, can be activated (on) or deactivated (off) in particular. For this purpose add or remove a checkmark in the respective option box. The concerned features belong to the categories described in the following table. In the last two columns each see the default setting for the Standard resp. Professional set.

Category	Description	Standard	Professional
Object types:			
Data server Image pool External file Recipe manager Text list Global network variable list	Only if the checkmark is set for an object type, that type will appear in the "Add Object " context menu. Text list and Image Pool objects also will be created automatically by the visualization if required.	off	on
IEC 61131-3 language:			
Support extended programming features:	Only if this feature is activated, the following object types will appear in the “Add object” context menu: - Interface - Method - Property Furthermore, when creating a new POU, the wizard will offer to extend FBs or implement interfaces. The new tabular declaration editor also will offer to extend FBs, implement interfaces, and decorate the entire POU or single declarations with compiler pragmas. The input assistant will offer declaration keywords that are used together with object oriented programming (EXTENDS, IMPLEMENTS, etc.)	off	on

Category	Description	Standard	Professional
Device management			
Enable simplified device handling	If activated, the user will only see the newest device description version for a particular device; the version itself is only displayed at some places for informational purposes.	on	off
Enable repository dialog	If activated, the menu command Tools / Device Repository... will be available. Otherwise, the command 'Tools / Install Device...' will appear instead, providing access to the Install Device dialog. Another possibility is to install new device descriptions using the package mechanism.	off	on
Library management			
Enable simplified library handling	If activated, the following simplifications will apply: - The user will not be able to save a project as a library. - The user will not be able to install a project to the library repository. - The user will not be able to open library projects for editing. - When adding a library in the Library Manager, there is no possibility to add placeholder references. - When editing the properties of library reference, only the namespace can be adapted. - In the Library Manager list itself, the version column is not displayed. - When adding a library in the Library Manager, only the newest version per particular library is displayed; the version information itself is omitted.	on	off
Hide system libraries	If activated, system libraries (“gray libraries”) will not be displayed in the Library Manager. Exception: if such a library could not be loaded, it will be displayed so that the user can remove the reference manually.	on	off
Enable repository dialog	If activated, the menu command Tools / Library Repository... will be available. Otherwise, the command 'Tools / Install Library...' will appear instead, providing access to the Select Library dialog. Another possibility is to install new libraries using the package mechanism.	off	on

12.23.7 Options, Device editor

This sub-dialog of the options dialog allows settings concerning the editing in a device editor.

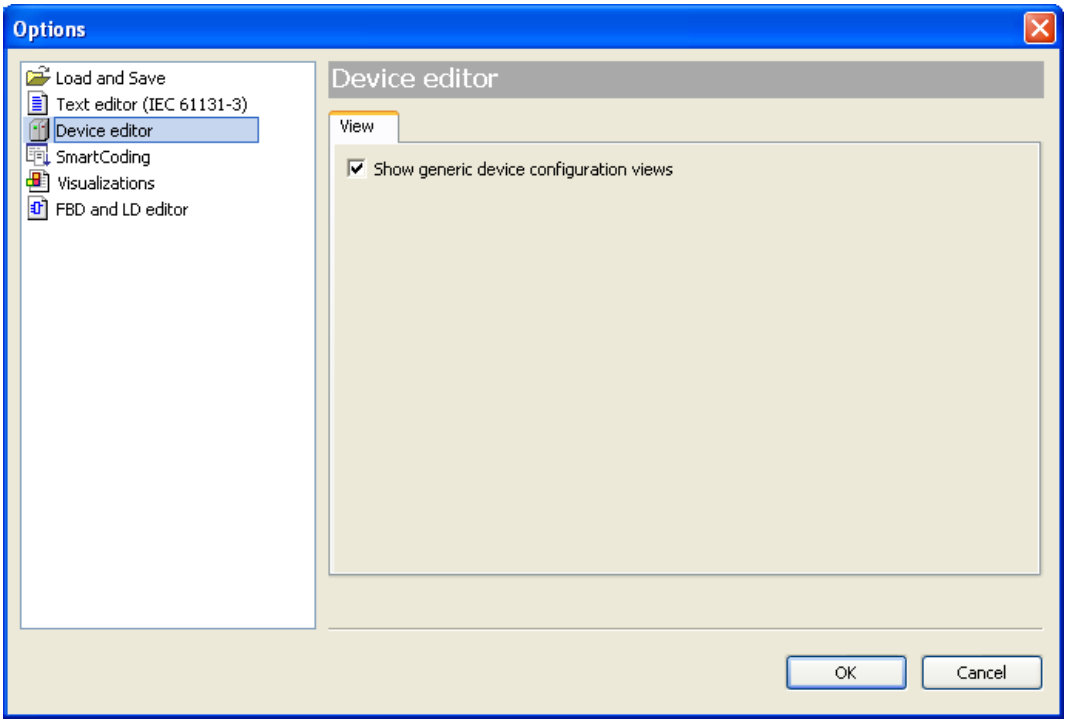


Abb. 222 Device editor options dialog

The View tab provides option **Show generic device configuration views**. Only if this option is activated the generic dialog 'Configuration' will be available in the device configuration editor of the respective device.

12.23.8 Options, SFC Editor

This sub-dialog of the options dialog provides sub-dialogs for settings concerning the editing in a SFC (Sequential Function Chart) editor.

The settings made in each of these dialogs will immediately be applied to the currently opened editor views as soon as they have been confirmed (and the dialog closed) with OK.

Layout

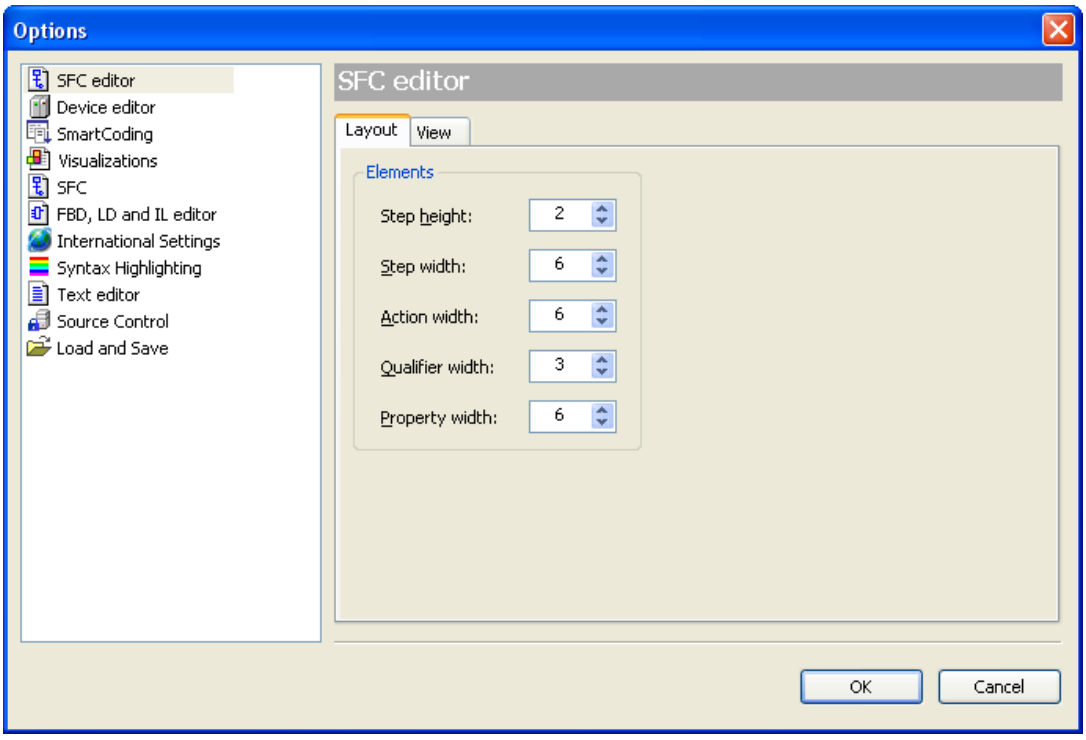


Abb. 223 SFC editor options dialog, category Layout

The settings are to be made in "grid units". One grid unit = font size currently set in the Text editor options (Text area / Font).

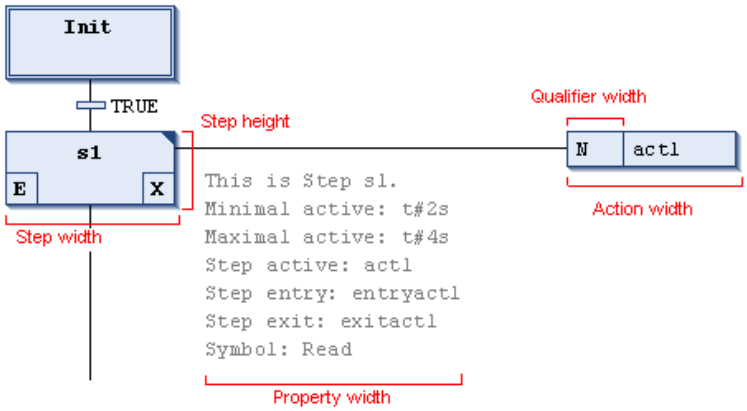


Abb. 224 Layout options

Step height: Step element height in grid units. Possible values: 1-100.
Step width. Step element width in pixels in grid units. Possible values: 2-100.
Action width: Action element display width in grid units. Possible values: 2-100.
Qualifier width: Qualifier display width in grid units. Possible values: 2-100.
Property width: Property display width in grid units. Possible values: 2-100.

View

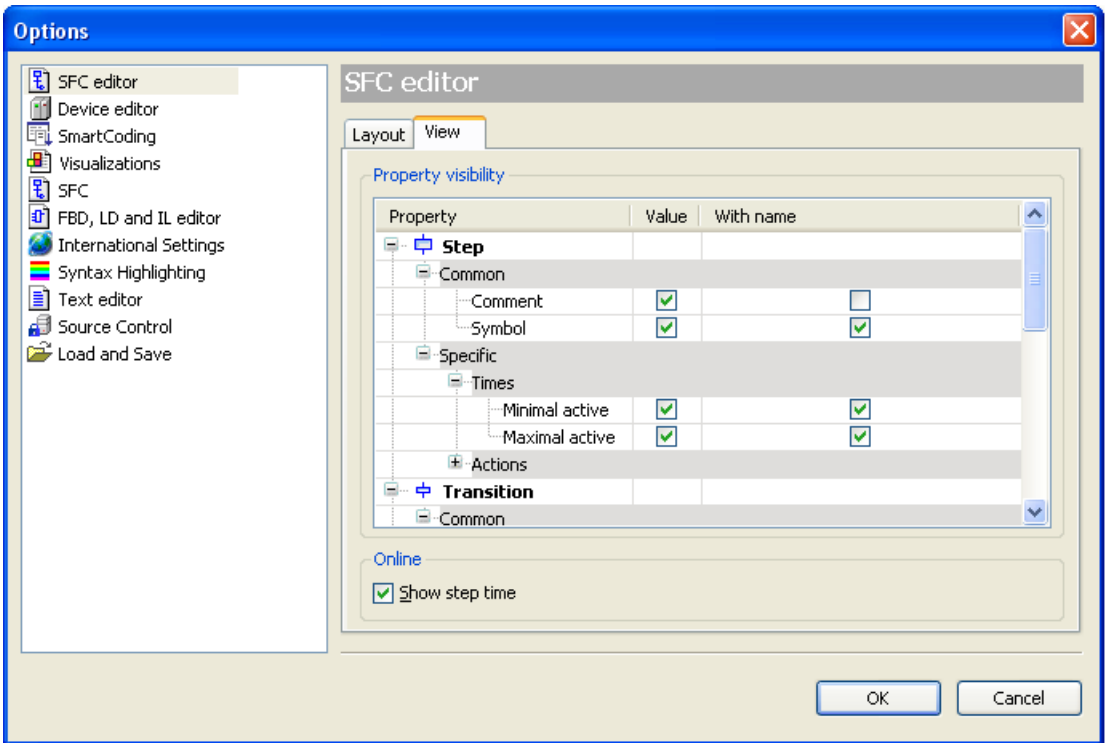


Abb. 225 SFC editor options dialog, category View

Property visibility:

Here you can define which of the element Properties (step attributes) should be displayed next to an element in the SFC editor view: The 'Common' and the 'Specific' properties for each element type are shown in the table and you can each activate the checkbox in the **Value** field in order to get displayed the property value right to the element in each SFC object. If you additionally activate the checkbox in the **With name** field, the value will be preceded by the name of the property.

Online:

If option **Show step time** is activated, in online mode the current step time will be displayed right to each step element for which time Properties are set.

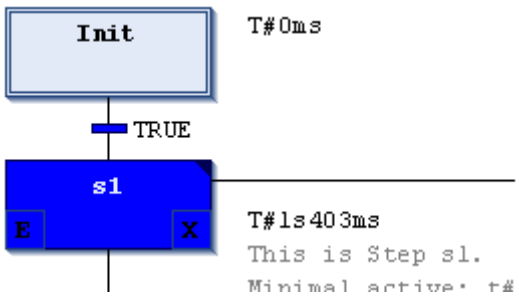


Abb. 226 Example of step time displayed in online mode

12.23.9 Options SFC

This sub-dialog of the options dialog provides the possibility to set **default settings for SFC objects**. Each **new** SFC object will get these defaults in its Properties.

Note: Notice that any modifications in the default settings will not be applied to existing SFC objects until you use the 'Set defaults' button in the SFC Settings (Properties) of the particular object. The object properties have higher priority than the defaults.

Notice also that the pure editor options, that is settings for layout and view are handled in a separate options dialog.

Build

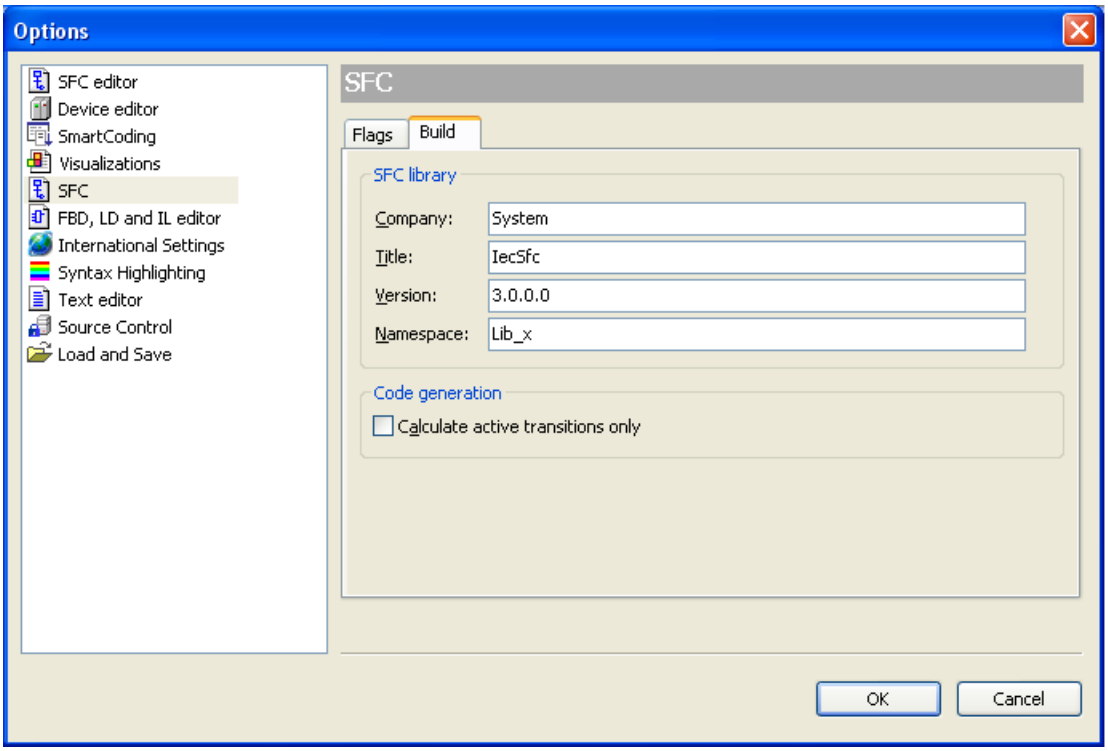


Abb. 227 SFC Options dialog, category 'Build'

SFC library:

Here the SFC library, which should be used by default, is defined by **Company**, **Title**, **Version** as defined in the Project Settings of the library. A **Namespace** can be specified in order to uniquely address the library. This might be necessary if different versions of the library are available on the system. Concerning namespace always keep aware, that there is no mismatch between a namespace defined in the Library Manager and the one specified for the particular object.

The default settings are those of the **SfcIec.library** provided with the standard profile.

Code generation:

Calculate active transitions only: If this option is activated, by default code will be generated only for the currently active transitions.

Flags

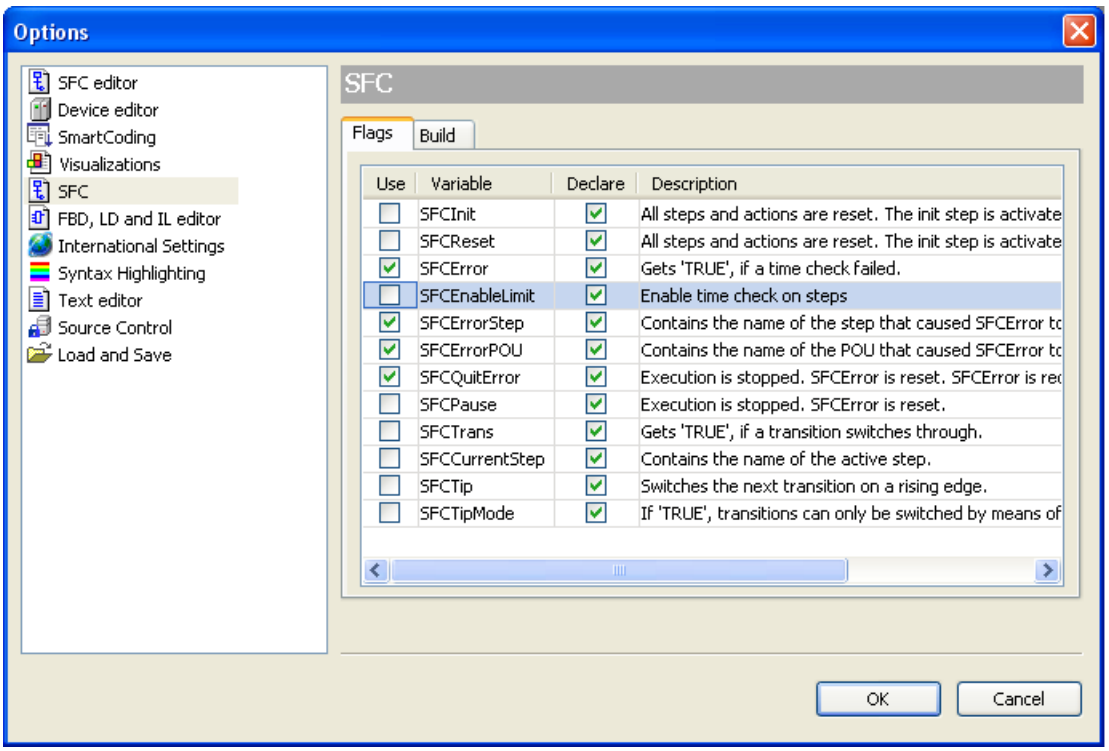


Abb. 228 SFC Options dialog, category 'Flags'

All possible 'flags', that is implicitly created variables for watching and controlling the processing of a SFC, are listed in this dialog.

By a mouse-click in the corresponding checkbox you can activate the automatic declaration (**Declare**) and **Use** of a flag. These settings will be applied as defaults to new SFC objects. If 'Declare' is activated, but 'Use' is deactivated for a flag, the declaration will be done, but the flag will be without any effect during processing.

Notice that an automatically declared flag variable will **only be visible in online mode** in the declaration part of the SFC Editor!

A timeout has been detected in step 's1' in SFC object 'POU' by flag SFCError.

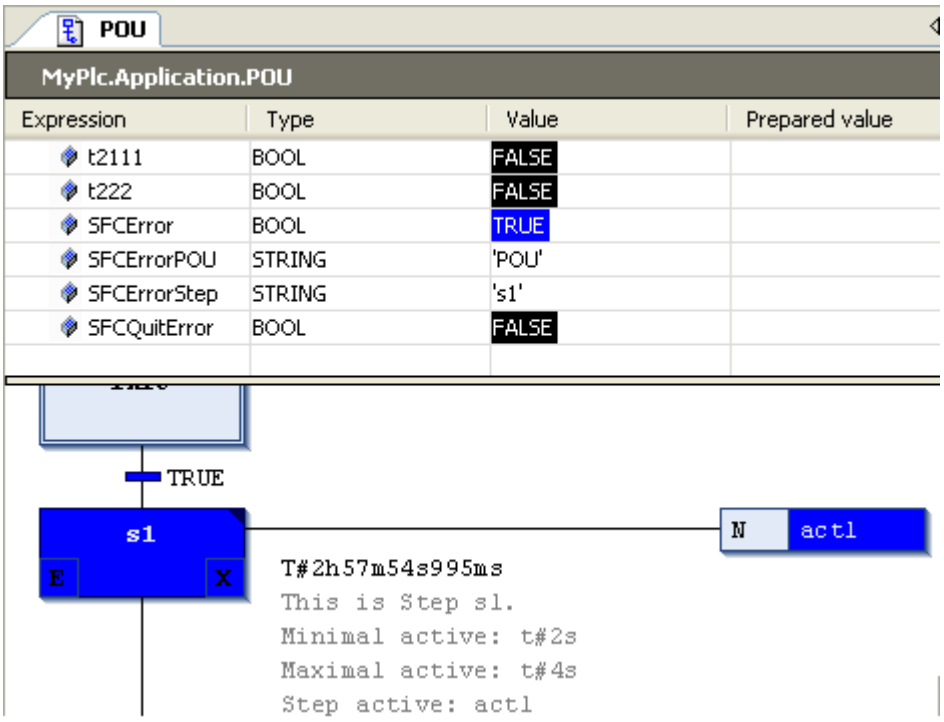


Abb. 229 Example of some SFC error flags in online mode of the editor

12.23.10 Options, Source Control

In this sub-dialog of the options dialog you can do some general settings concerning the Source Control of the project.

Notice that Source Control must be explicitly activated for a project (in the 'Change Source Control' dialog) and that in order to be able to use it you must be logged in to the ENI Server which is connecting to the respective external project data base.

Here in the options dialog it is defined, which source control actions should be done automatically. For each action choose whether it should run without any further prompt to the user (**Yes**), whether before execution the user should be asked (**Prompt**), or whether it should not be executed at all (**No**).

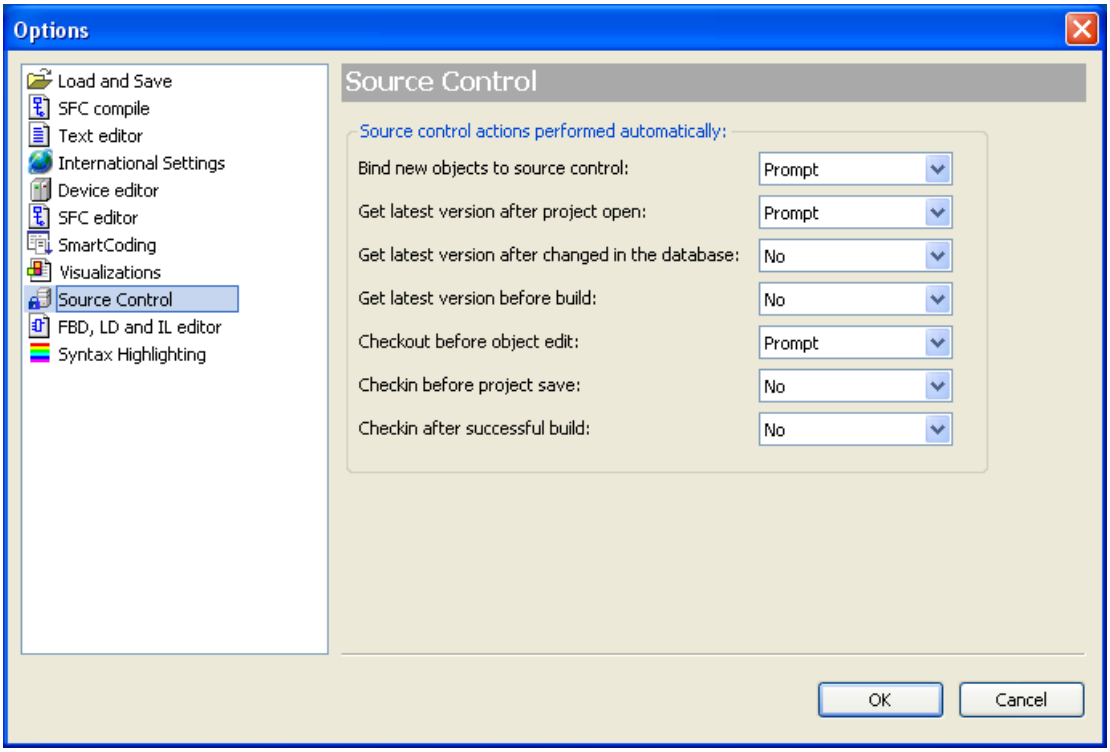


Abb. 230 Source Control options dialog

The actions:

Bind new objects to source control	If you add a new object to the project, this will be bound to the default data base location.
Get latest version after project open	When you open the project, the latest version of all objects which are under source control, will be copied from the above defined data base folder to the project, whereby the local version of objects will be overwritten.
Get latest version after changed in the database	As soon as a newer version of an object is checked in to the data base (for example by another user), the user will be informed.
Get latest version before build	If a build process is to be done for the current project, primarily the latest version of all objects which are under source control, will be copied from the above defined data base folder to the project, whereby the local version of objects will be overwritten.
Checkout before object edit	As soon as you start to edit an object which is under source control, the object will be checked out. If the object is currently already checked out by another user (indicated by a red cross before the object name in the »PLC Designer« object organizer), then an appropriate message will pop up.
Checkin before project save	When the project is going to be saved, previously all objects, which are under source control, will be checked in.
Checkin after successful build	After an error-free build of the project all objects, which are under source control, will be checked in.

12.23.11 Options, Visualizations

This sub-dialog of the options dialog consists of the tabs **General** and **Grid** allowing common settings for working on visualizations in the programming system.

General

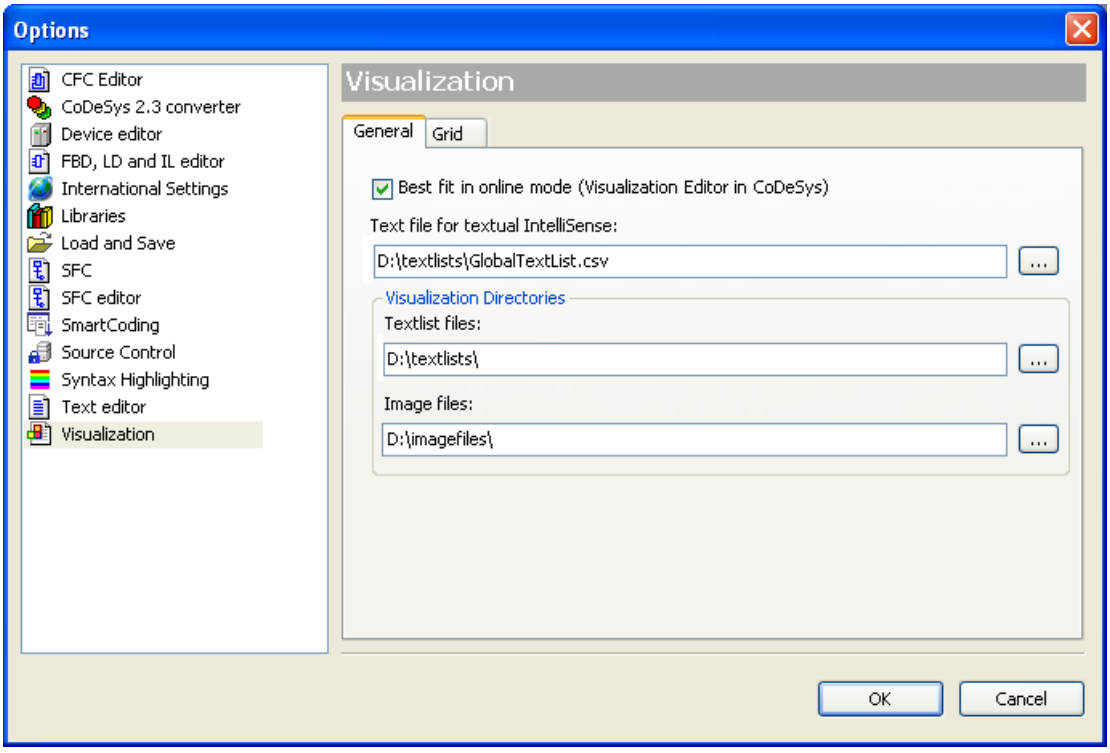


Abb. 231 Visualization options dalog, General

Best fit in online mode: Activate this checkbox for the visualization being scaled optimally according to the actual size of the online visualization window within the programming system.

Note: The scaling of a target visualization can be defined within the visualization manager editor.

Text file for textual "List Components" functionality: In the related edit field you may enter the name (including the path) of the file (of type .csv) providing the proposals that will be made by the "List Components" function, when you enter a string in the edit field of the item 'Texts\Text' within the properties of a visual element. This text file may stem for example from the export of a global textlist. The text file has to be designed as table, its structure must follow the structure of textlists, as described on the Textlists help page.

Visualization Directories:

Here you can define default directory paths, which however still might be modified project-specifically in the Project Settings, category Visualization. Finally valid for the project are the Project Settings!

Textlist files: Use button ... to open the standard dialog for specifying an existing or new directory for language files (textlists), which should be available for text and language configuration within the visualization.

Image files: Use button ... to open the standard dialog for specifying one or several existing or new directory for image files (image pools), which should be available for the use in the visualizations. If entering several directories, separate by semicolons.

Grid

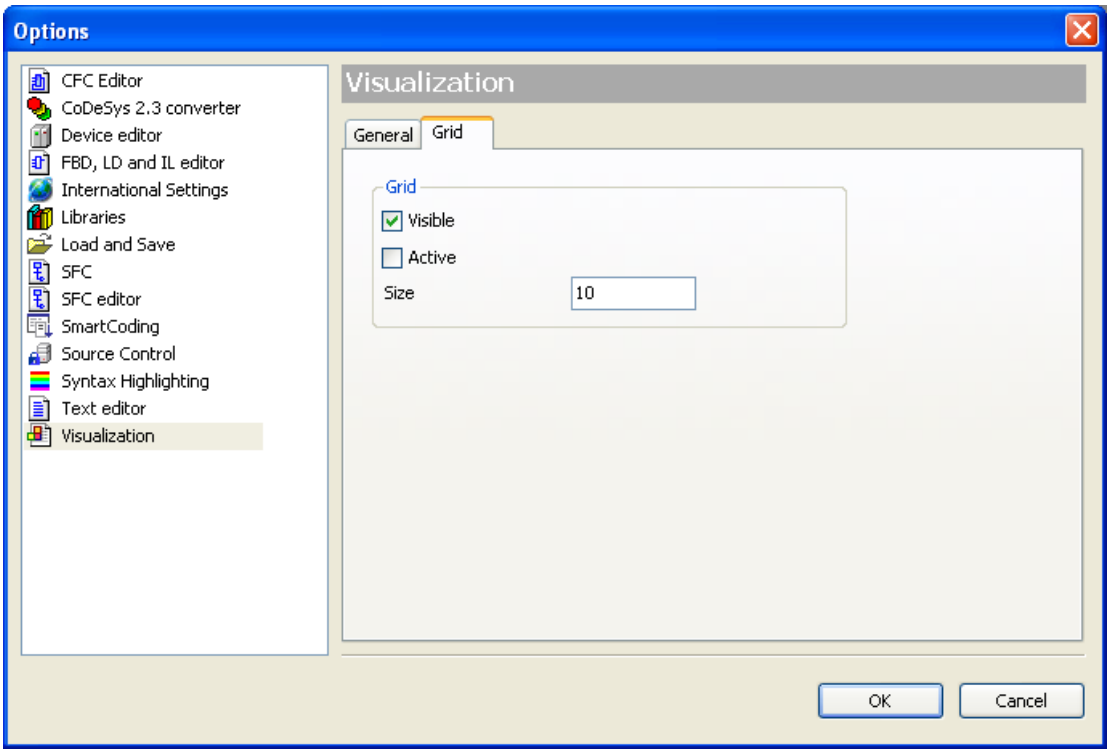


Abb. 232 Visuialization options dalog, Grid

Visible: If this option is activated, a grid according to the Size defined below will be displayed in the visualization editor window in order to ease optic orientation.

Active: If this option is activated, visualization elements can only be positioned according to the grid points defined below in "Size", no matter whether the grid is "visible" or just "active". When inserting or moving an element this means that the center of the element can only be placed on a grid point. When modifying an element, it means that the small rectangle at an edge of the element, which gets shifted in order to change the size or form of the element, can only be placed on a grid point.

Size: Vertical and horizontal distance between the grid points in pixels.

12.23.12 Customize Dialogs

The Customize dialog, provided with the Frame plug-in, contains sub-dialogs (tabs) for the configuration of the user interface concerning structure and contents of menus and toolbars as well as of the keyboard functions.

The dialog is opened via the command '**Customize**' (category 'Preferences').

The default settings are defined by the provider and can be restored at any time via button **Reset to default**. This will open a dialog where you can select which of the categories (Menu, Keyboard, Toolbars) you want to get restored.

The current settings confirmed with OK get saved on the local system but you also have the possibility to save and reload specific configuration files, see below.

The following sub-dialogs are available by default:

- Menu
- Keyboard
- Toolbars

How to save and reload the dialog settings:

Press button **Save** to store the settings of the currently opened subdialog in a file. The standard dialog for saving a file will open, where you have to enter a filename without extension and confirm with 'Save'. This can be done for each sub-dialog and the following files might be stored:

"<filename>.opt.menu" for the menu settings

"<filename>.opt.keyb" for the keyboard settings

"<filename>.opt.tbar" for the toolbar settings

To reload the settings from one of those files, use button **Load**. A Load Menu dialog will open, where you can browse for the desired menu file, whose settings then will be taken into the respective Customize dialog.

Customize Toolbars

This sub-dialog of the Customize dialog allows to

- - activate resp. deactivate the available toolbars, which means to make them visible or invisible in the user interface.
- - add new toolbars which are defined in the 'Edit Toolbar..' dialog.
- - delete toolbars
- - save the current toolbars settings to an external file resp. reload previously saved settings to the current project
- - rename the toolbar

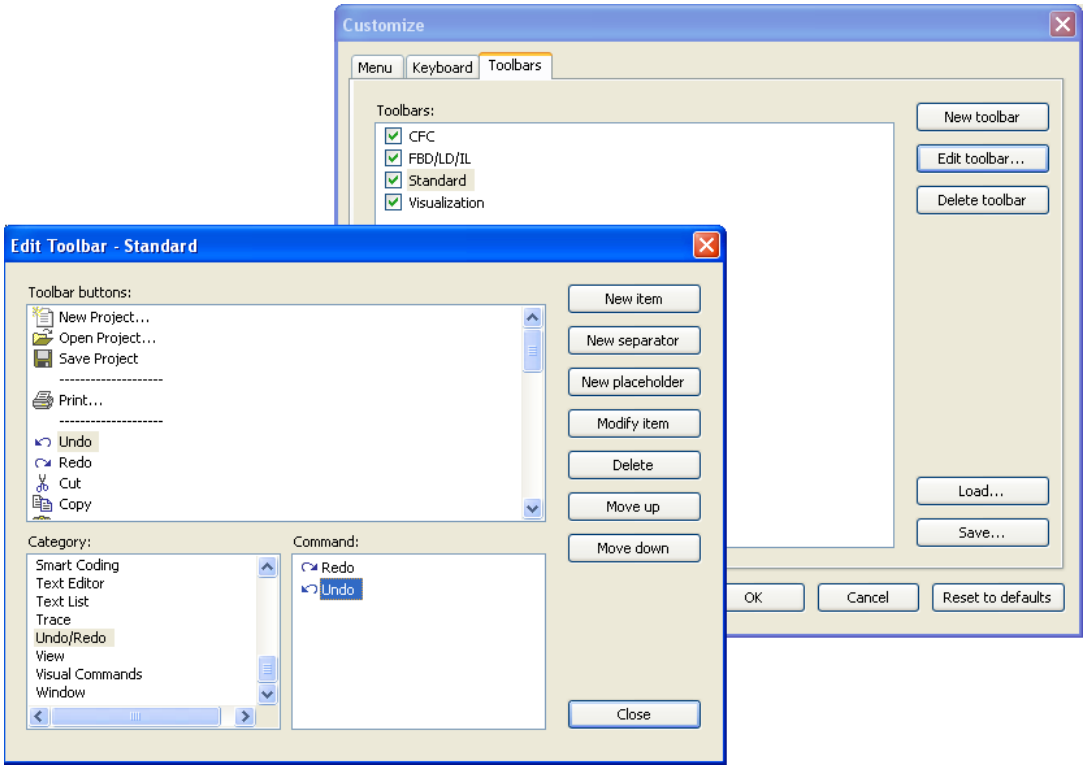


Abb. 233 Customize Toolbars dialogs

How to activate a toolbar:

The 'Toolbars:' list shows the names of the toolbars which are currently defined.

To activate or deactivate a toolbar, which means to make it visible or invisible in the user interface, click on the checkbox. If a check is shown, the toolbar will be activate.

How to define a new toolbar:

To define a new toolbar press button **New toolbar**. An edit field preceded by a checkbox will be opened at the topmost position of the 'Toolbars:' list. Enter a name for the new toolbar.

To edit the new toolbar or to edit an existing one press button **Edit toolbar...** . The **dialog 'Edit Toolbar - <current toolbar>'** will open where you define which commands should be represented in the toolbar:

Select a command from the Command:' list which offers all commands belonging to the command currently selected in the 'Category:' list. Then press button **New item** to add the command to the 'Toolbar buttons:' list. In this way select and add all commands you want to assign to the toolbar. The order of the commands in the toolbar will be the same as that in the 'Toolbar buttons:' list. To reorganize the order use buttons Move up, Move down and Delete. Further on you can add separator lines by pressing button New separator.

How to delete a toolbar

To delete a toolbar, select the respective entry in the 'Toolbars:' list and press button **Delete toolbar**.

How to rename a toolbar:

If you want to rename a toolbar within the configuration, you can directly edit its name in the toolbar list. In order to get into edit mode, select the respective entry in the list and use the <space>-key or perform another mouse-click on the seleted entry.

Customize Menu

This sub-dialog of the Customize dialog allows defining the structure and commands of the»PLC Designer« menus.

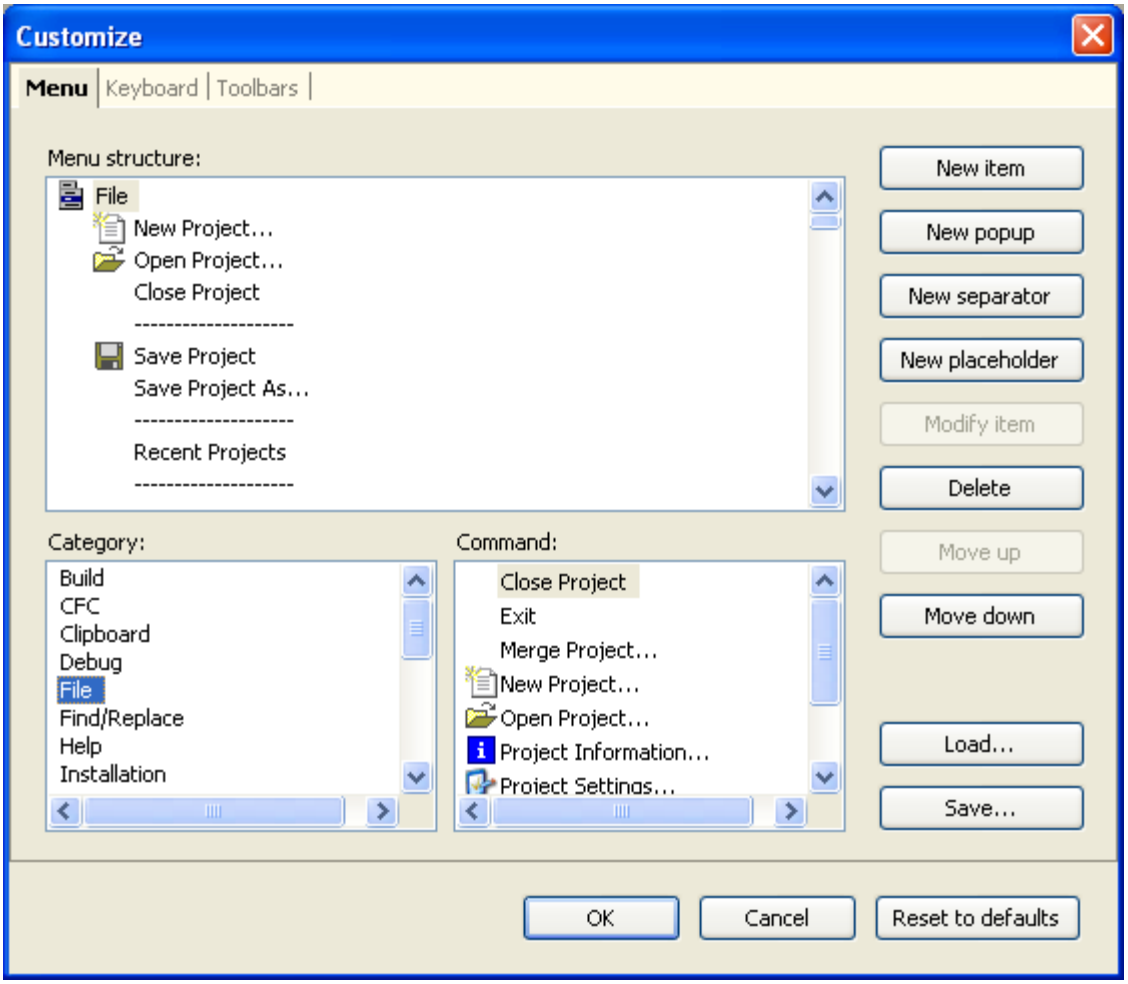


Abb. 234 Customize dialog, Menu

Menu structure: The currently defined menus, sub-menus and the assigned commands are displayed. A menu/sub-menu title is indicated by an icon . Indented below the respective commands and sub-menus are listed in the same top-down order as they will be displayed in the»PLC Designer« menu. They might be separated by a separator line.

Category window: Here you find the available categories of commands provided with the current profile. Each category offers several commands which are displayed in the **Command** window.

How to add a new command or a separator to an existing menu:

In the Menu structure window select the title of the menu to which you want to add a command. In the Category window select the desired category and then the desired command in the Command window. If you press button New item now, the selected command will be inserted below the menu title in the Menu structure window. To place it somewhere else in the menu's order of commands, use buttons Move down or Move up, which will shift the command one line down or up.

Alternatively you can select an existing command in the Menu structure window. In this case the new command will be entered directly above.

Use button New separator to insert a separator line above the currently marked command in the Menu structure window.

How to add a new Menu or Sub-Menu:

In the **Menu structure** window select that line above which you want to insert a new menu resp. sub-menu. Notice that the order of menus from top to bottom in the Menu structure window corresponds to the order of the menus in the menu bar from left to right. Press button **New popup** to open the **Edit text** dialog, where you can define a menu title in various languages:

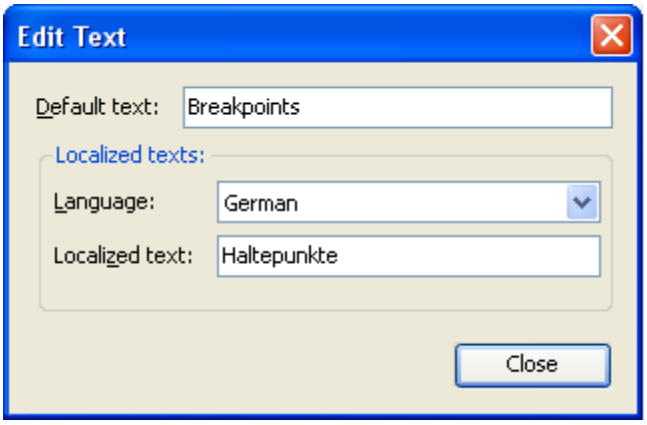


Abb. 235 'Edit text' dialog for defining a menu title

The **Default text** will be displayed as menu title, if for the currently set user interface language no localized text is defined. Replace "<Popup>" by the desired menu title. For localization purposes you can choose other languages from the selection list at the **Language** field and each enter a corresponding menu title in the field **Localized Text**. You can enter the definitions for multiple languages one by one. All definitions will be stored at closing the dialog.

The new menu will be displayed in the menu bar only if you have assigned at least one command. The menu title will be displayed in the currently set user interface language, if a localized text has been defined for that language.

How to add a placeholder:

A placeholder in the menu structure definition describes a particular command category. This is useful for later extensions by additional components. If a new component brings a menu command of the same category as defined by the placeholder, this command will be added to that menu, where the placeholder is located.

Example: In the Menu structure window select menu 'Edit', in the Category window select 'Clipboard'. Press button New placeholder. The placeholder <Clipboard> will be inserted below entry Edit in the Menu structure:

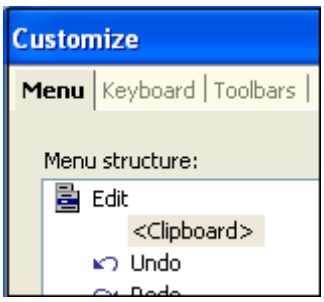


Abb. 236 Example, placeholder added to menu

This will not effect the current view of the Edit menu in your project resp. in new projects, until you add any new plug-in providing a command (for example "Select none") which is assigned to category 'Clipboard'. In this case command "Select none" will be inserted in the Edit menu at the top position.

Customize Keyboard

This sub-dialog of the Customize dialog allows to define keyboard functions:

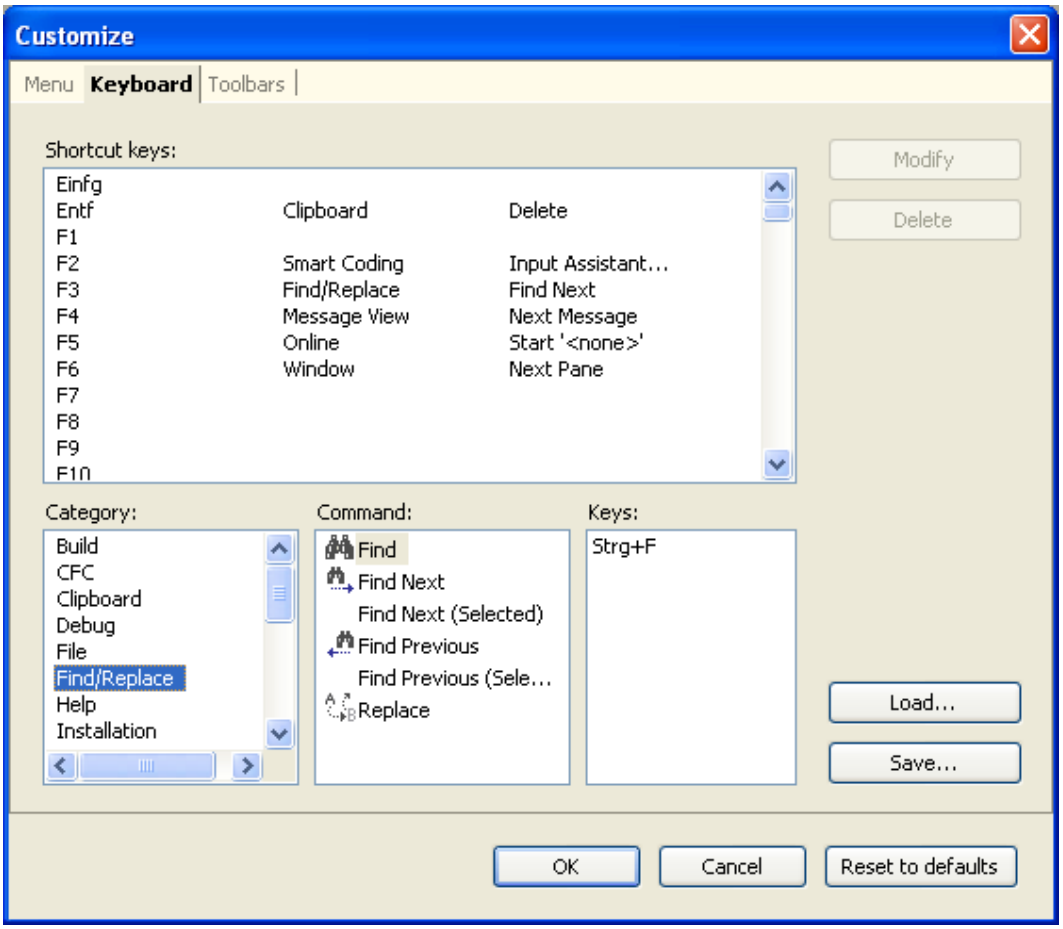


Abb. 237 Customize dialog, Keyboard

For defining shortcut keys the following lists are available:

Shortcut Keys: List of all possible keys and currently assigned commands. First column: key; second column: command category; third column: command; If a key has not yet got assigned a command, just the key is displayed.

Category: List of command categories.

Command: List of commands belonging to the currently in 'Category:' selected category.

Keys: Key currently assigned to the currently in 'Command:' selected command.

How to assign a new command to a key resp. to modify an existing assignment ?:

- Select the key in the 'Shortcut keys:' list by a mouse-click, the key's line will get highlighted.
- From the 'Command:' selection list choose the desired command. It depends on the currently selected category in the 'Category:' selection list which commands currently are offered.
- Now press button **Modify**. The chosen command will be added to the key's line in the 'Shortcut keys:' list.

To remove an assignment select the key in the 'Shortcut keys:' list and press button **Delete**.

12.24 Print

The 'Print' command category provides commands for creating print-outs for the current project.

12.24.1 Print commands

The 'Print Engine' plug-in for command category 'Print' provides commands (see below) for producing printouts of the current project, on a printer or to a file.

By default these commands are part of the 'Print' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

The commands:

- Document...
- Page Setup...
- Print...

12.24.2 Document...

This command (category 'Print') opens the **Document Project** dialog, where you configure the documentation of the project.

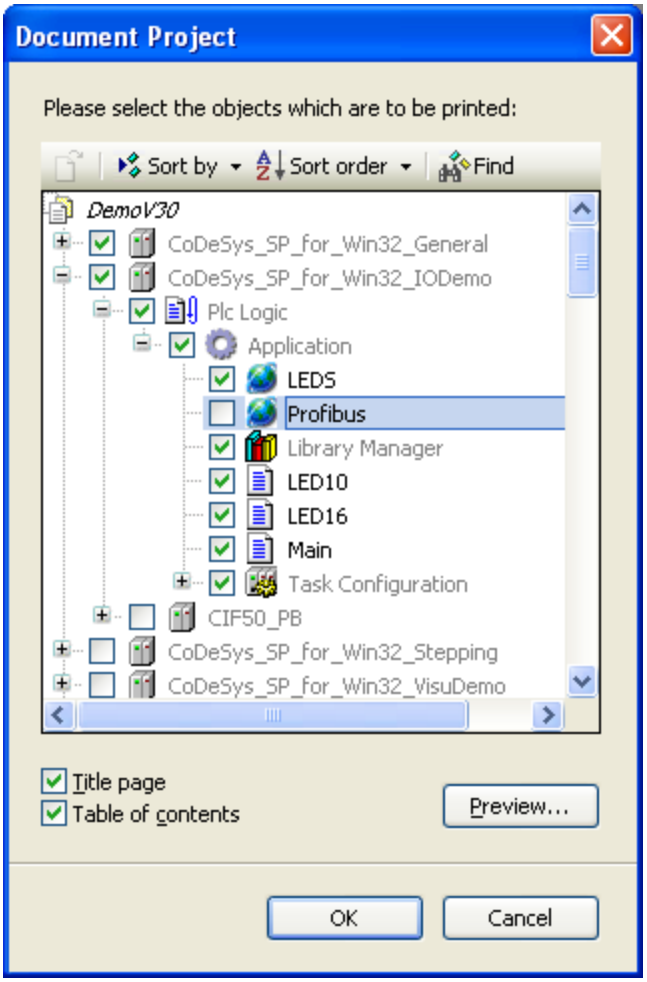


Abb. 238 Document Project dialog

The project objects as defined in the POUs and Devices trees are presented in a tree also and you can select which object(s) should be documented in the form of a printout. The contents of an object which is displayed grey-shaded, currently are not printable, but if a tick is set for that object, it will be considered in the documentation in so far as the object name is noticed in the structure and the contents tree.

For sorting and searching in the objects tree a toolbar, also known from other view windows, is available above the tree.

If option **Title page** is activated, the documentation will be extended by a title page as defined in the Page Setup.

If option **Table of Contents** is activated, the documentation will be extended by a contents page showing all objects and the corresponding page numbers in the documentation. The header levels to be noticed are defined in the Page Setup, category Document.

The current configuration will be stored as **default** for the project. If currently all project objects are selected, then later by default always all objects, that is also any newly created, will be selected. If in the current settings some objects are excluded from selection, then by default exactly the remaining selection will be set and thus newly created objects not be selected.

12.24.3 Page Setup...

Symbol:

This command (category 'Print') opens the Page Setup dialog for configuring the layout of a printout page which can be produced via the 'Print...' or 'Document...' command.

Button **Preview** always is available to check the currently configured page setup in a preview. The dialog 'Seitenansicht' will open where you can browse through the configured pages.

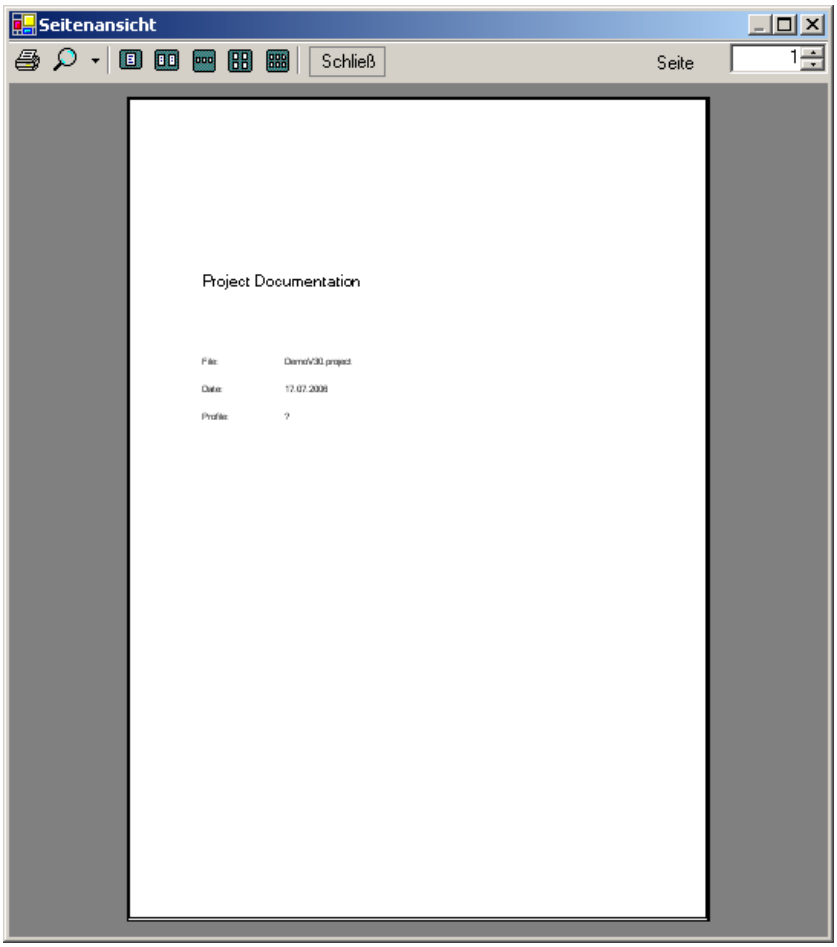


Abb. 239 Dialog 'Page Setup', Preview'

Select the desired page number in the 'Seite' field in the upper right corner. Set the desired zoom factor by the menu resp. choose one of the view modes 1 page, 2 pages, 4 pages, 6 pages ( ). To print the preview on the default printer use button . By 'Schließen' the preview will be closed.

The following subdialogs are available on tabs of the Page Setup dialog:

Paper

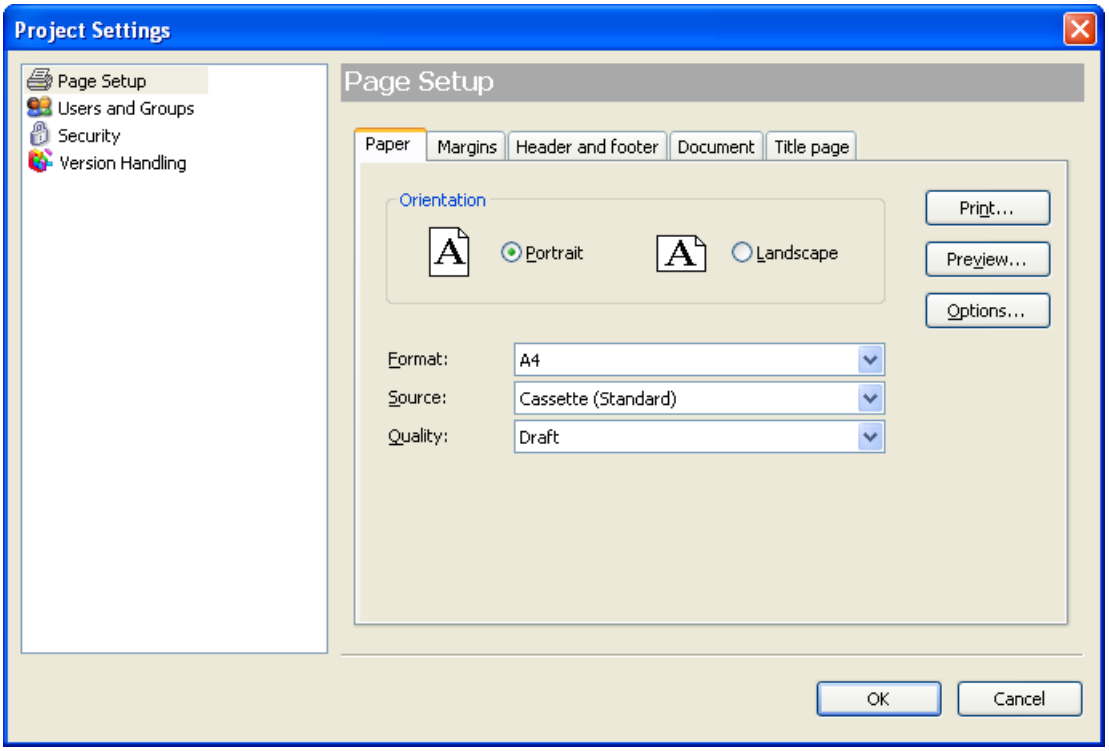


Abb. 240 Dialog 'Page Setup', category 'Paper'

Orientation: Choose whether the page should be in Portrait or Landscape orientation.

Format: Choose from the selection list the desired format, for example "A4".

Source: Choose from the selection list which source should be used for the paper feed, for example "Cassette (Standard)".

Quality: Choose the desired print quality, for example "Draft" for preliminary output.

Margins

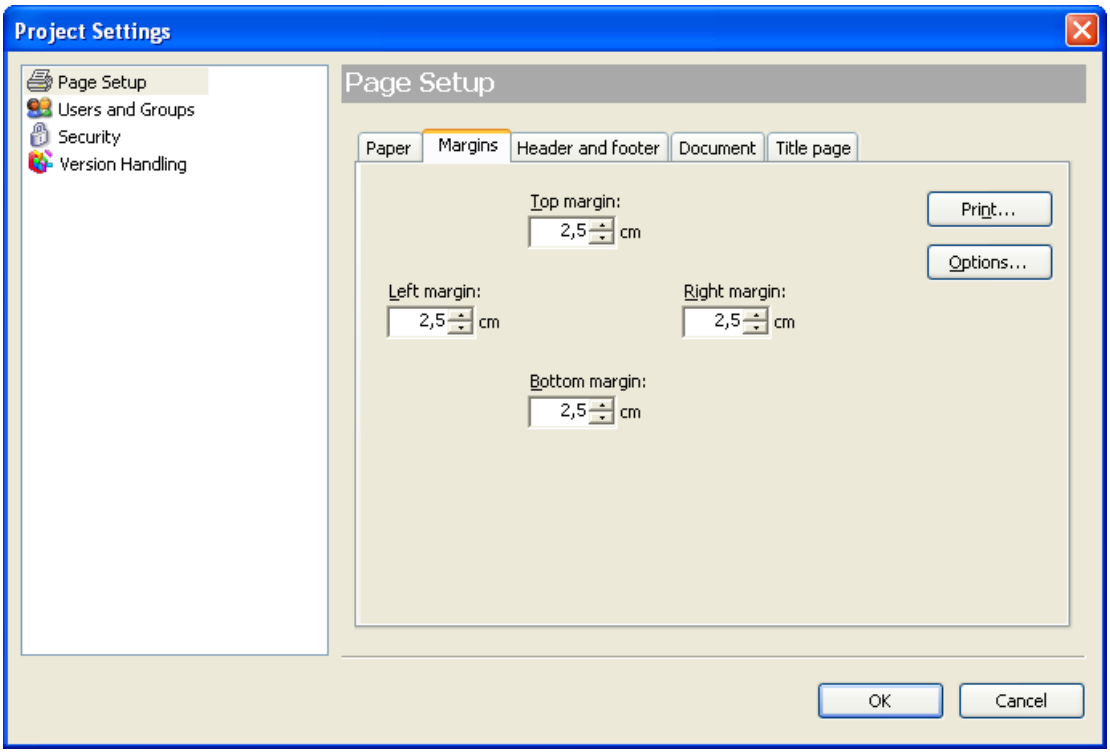


Abb. 241 Dialog 'Page Setup', category 'Margins'

Define here the top, bottom, left and right page margins of the printout page in centimeters. Use the arrows to set the value down or up in 0,5 cm steps or directly edit the value fields.

Header and footer

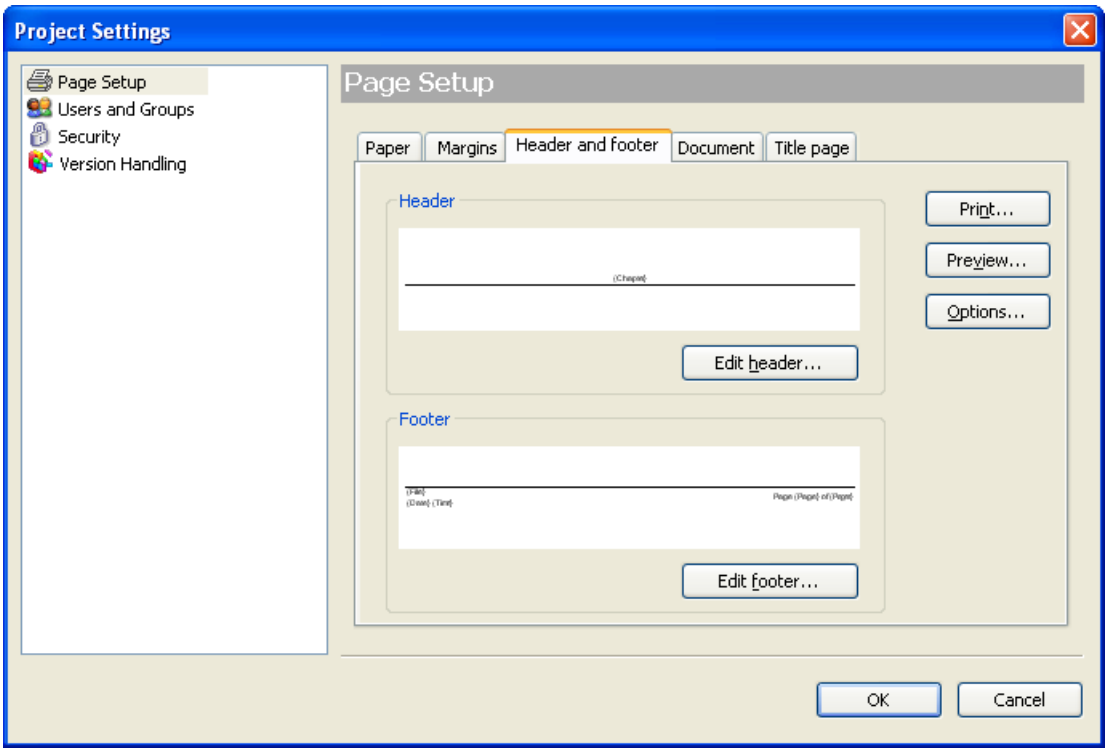


Abb. 242 Dialog 'Page Setup', category 'Header and footer'

A header and a footer area can be defined. Use the Edit header resp. Edit footer button to open a dialog where the layout details are configured:

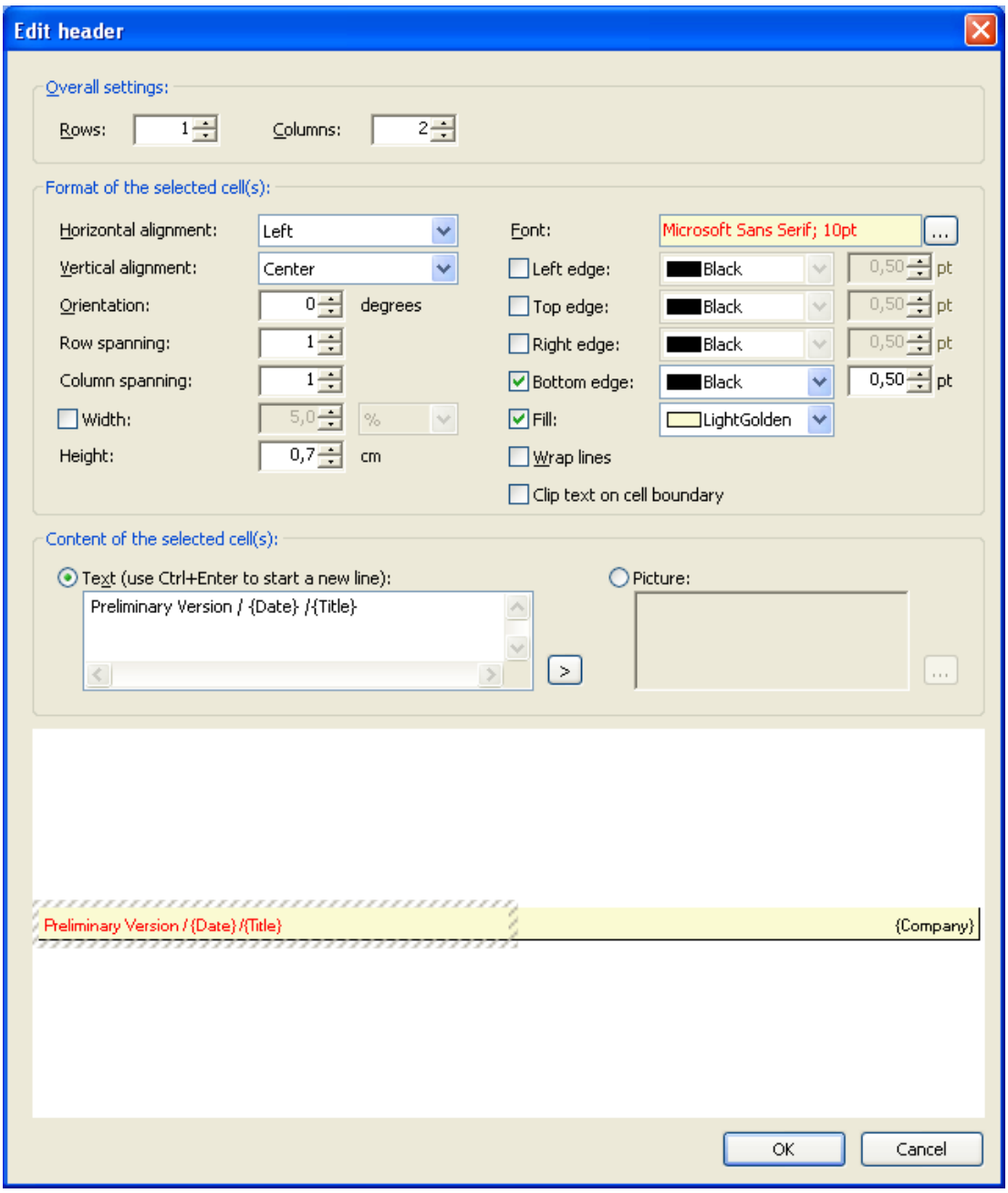


Abb. 243 Dialog 'Edit Header' (corresponds to 'Edit Footer')

Overall settings:

Header and footer area are each organized as a table. So particular rows and columns can be configured and the resulting cells be filled with text or picture. A cell gets selected by a mouse-click. on the respective area. A **selected cell** is indicated by a grey striped ( ) frame:

Rows: Number of rows; if the number is increased, the additional rows will be inserted below the existing one(s).

Columns: Number of columns, if the number is increased, the additional columns will be inserted right to the existing one(s).

Format of the selected cell:

Select the cell to be configured by a mouse-click in the layout field in the lower part of the dialog.

Horizontal alignment: Choose whether the content of the currently selected cell horizontally should be aligned right, center or left. Default: Center.

Vertical alignment: Choose whether the content of the currently selected cell vertically should be aligned with the top, center or bottom. Default: Center.

Orientation: Enter the number of degrees to which the content of the currently selected cell should be rotated around its base, whereby the "base" is determined by the horizontal alignment (for example alignment = left -> the text or picture will be rotated around its leftmost, vertically centered point). The rotation always is counter-clockwise, valid values are 0-359. Default: 0.

Row spanning: Enter the number of rows which should be merged to a single one in the current column. Example: If you enter "3" the currently selected cell will be merged with the two cells below. Default: 1.

Column spanning: Enter the number of columns which should be merged to a single one in the current row. Example: If you enter "3" the currently selected cell will be merged with the two cells to its right. Default: 1.

Width: If this option is deactivated, all columns automatically get the same width and the sum of the column widths corresponds to the currently set page width. If you want to change the width for a particular column, select any cell of that column, activate the Width option and enter the desired width in cm resp. in percents of the complete header resp. footer width.

Height: Defines the height of the row, where currently any cell is selected, in centimeters. Default: 0,5.

Font: The name and size of the font, currently defined for the text content of the selected cell, is displayed. To modify the font parameters use the standard font configuration dialog which can be opened by button ...

Left edge, **Top edge**, **Right edge**, **Bottom edge**: Here the visibility, color and line width of the borders of the currently selected cell are defined. By default they are black and have a width of 0,5 points. Also by default in headers the bottom edge, in footers the top edge are activated, that is set visible.

Fill: If this option is activated, a fill color for the currently selected cell can be chosen from the selection list. By default a cell is not filled.

Wrap lines: If this option is activated, the contents of the currently selected cell will be wrapped at the end of a line if they exceed the cell width. Per default the option is not activated, that is the contents will be cut by the cell borders.

Content of the selected cell(s):

The currently selected cell is can be filled with text or a picture. Activate the desired option.

Text: In the edit window enter any text which should be displayed in the currently selected cell. Use <Ctrl>+<Enter> to start a new line. Placeholders can be used, which will be replaced by the actual values when the page is printed. Button [>] opens the **list of available placeholders**:

Placeholders for common information:

{Page}	Running page number, for example "3"
{Pages}	Total number of pages, for example "23"
{Date}	Current date dd-mm-yyyy, for example "07.07.2006"
{Time}	Current time hh-mm, for example "17:55"

Placeholders for information as defined in the Project Information:

{Path}	Location of the project file, for example "D:\projects"
{File}	Name of the project file, for example "mach1.project"
{Chapter}	Chapter number + Chapter name: The chapter number results from the position of the object in the "Documentation tree" as defined via the Document Project dialog. Header level "1" is used for objects which in the POUs or Devices tree are directly positioned below the root. Objects indented below get header levels 2 and higher. Example: "3.1 POU PLC_PRG.action1" PLC_PRG is the third object from the POUs view to be documented, action1 is assigned to PLC_PRG.
{Size}	Size of the project file, for example "65,42 KB (66.989 bytes)"
{MS-DOS Name}	MS-DOS name of the project file, for example "D:\COD~1.0PR\UN2A48~1.PRO"
{Created}	Creation date of project file, for example "07.07.2006 17:36"
{Changed}	Date of last modification of project file, for example "07.07.2006 17:36"
{Last Access}	Date of last access of project file, for example "07.07.2006 17:36"
{Attributes}	File attributes: R=read only / H= Hidden / A=Archive / S=System
{Profile}	Name of the currently used profile, for example "»PLC Designer« V3.x"
{Company}	Company name
{Title}	Project title
{Version}	Project version number
{Default Namespace}	Default namespace, relevant if project is used as library
{Author}	Author name
{Description}	Description

Picture: Use button [...] to get the standard dialog for browsing for a file. Select the desired image file to get it inserted in the currently selected cell.

Document

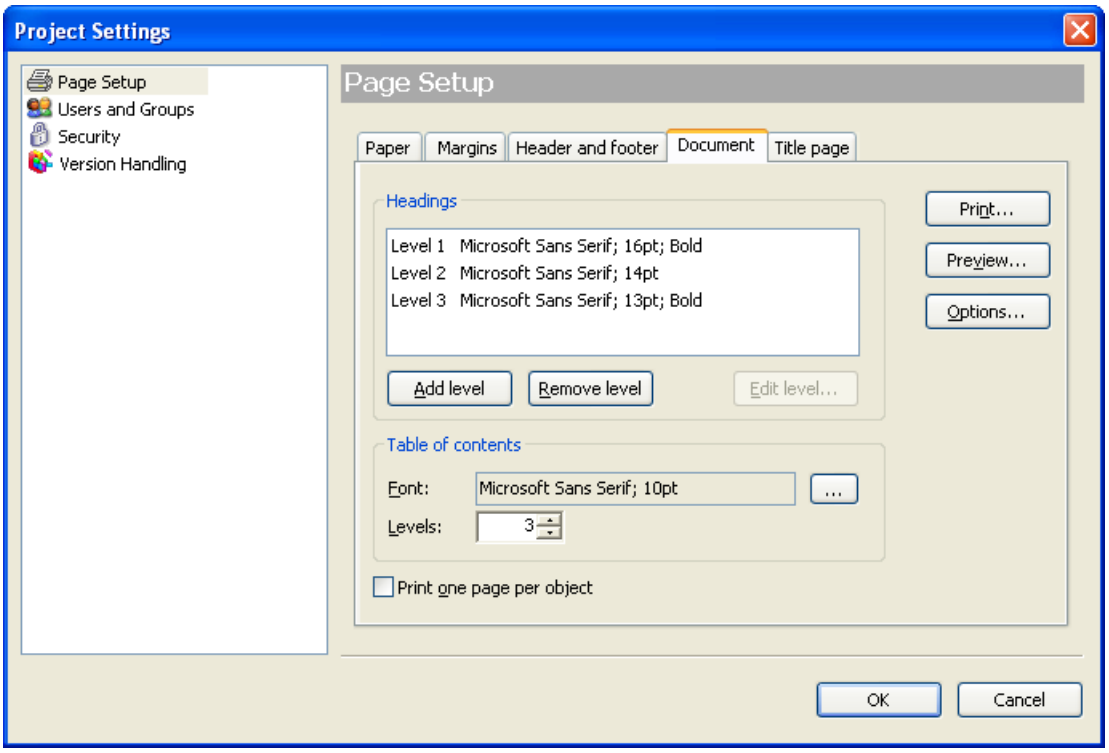


Abb. 244 Dialog 'Page Setup', category 'Document'

Here you define the format of the headings and which heading levels are to be noticed in the table of contents.

Note: What is a heading level? The heading level an object gets assigned in the documentation, is determined by the position of the object in the POUs resp. Devices tree. All objects directly below the root of the tree will get heading level 1 in a print set. Objects further indented get header levels 2 and higher. The numbering within one level is according to the vertical sequence of the objects within the tree.

Headings:

By default heading levels 1-3 are already configured with default **font settings** (as displayed in the picture above). They are shown in the Headings list.

Further levels, which are possibly used in the project documentation, automatically will get displayed according to the settings defined for the last level described in the list (for example in default case level 4 and higher will look like level 3). If you however want to configure those higher levels explicitly, use button **Add level** to add them to the headings list, because:

All heading levels shown in the list can be **modified** concerning their font settings. For this purpose select the list entry and use button **Edit level...**. The standard dialog for font configuration will open where you can do the appropriate settings.

The last entry in the headings list can be removed by button **Remove level**.

Table of contents:

Define the font settings (**Font:** use button ... to get the font configuration dialog) and how many heading levels (**Levels:** set the desired number) the should be noticed in the Contents of the project documentation.

Print one page per object: Activate this option, if each object should be printed on a separated page. By default no page breaks are inserted.

Title page

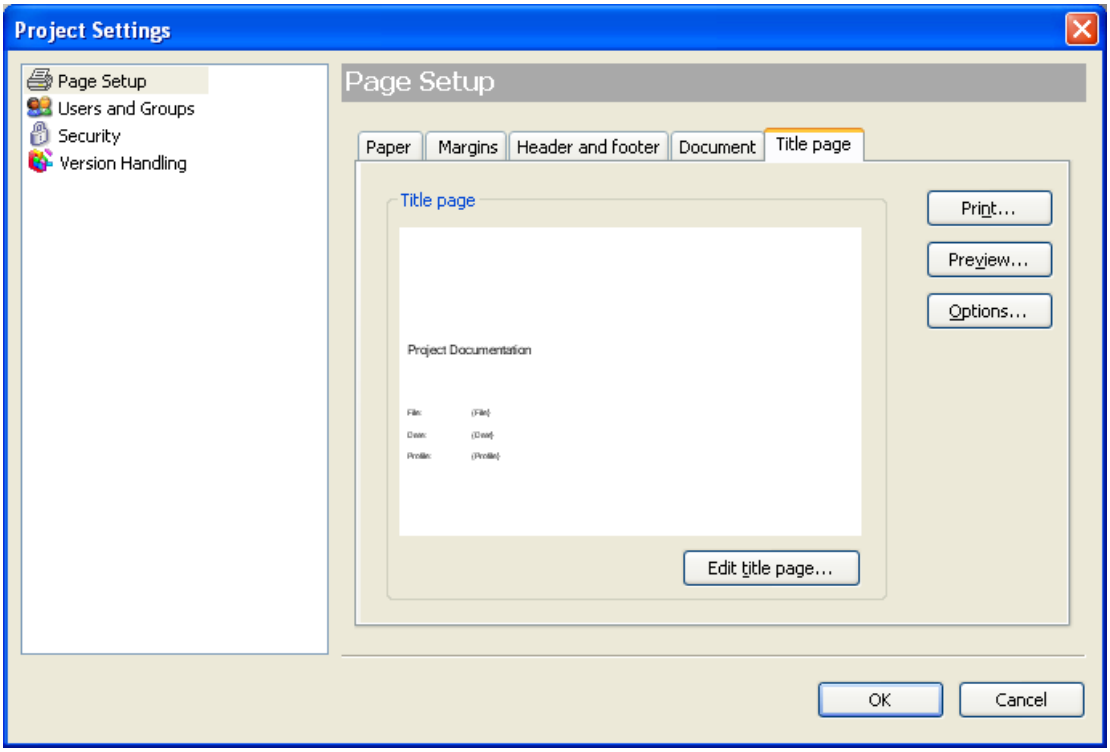


Abb. 245 Dialog 'Page Setup', category 'Title Page'

Here you define the layout of the title page of the project documentation. This page will only be added to a printout , if option 'Title page' is activated in the Document Project dialog, where you set up the contents of the current documentation set.

Button **Edit title page...** will open the corresponding dialog for defining the layout. For a description of the dialog items please see above: Edit Header or Edit Footer dialog.

12.24.4 Print...

Symbol:

This command (category 'Print') serves to print the currently defined part of the project on a defined printer according to a defined layout.

The standard print dialog will open where you define the desired print range, the printer properties and number of copies. If option 'Ausgabe in Datei' is activated, you can define an appropriate file instead of getting an output on a printer.

To define the print page layout use command Page Setup....

When you close the dialog with OK while you are already printing, the printing will stop after the next page.

12.25 Recipe

This command category provides commands for creating recipe definitions in the Recipe Manager editor.

12.25.1 Recipe Commands

The 'RecipeManagement Editor' plug-in provides commands of category 'Recipe commands' for working in the Recipe Manager editor.

These commands are available in the context menu when a recipe definition is currently opened in an editor window. By default then also the 'Recipe' menu provides the commands. If applicable, use the Customize dialog to modify the current menu configuration.

The commands:

- Add a new recipe
- Remove recipes

12.25.2 Add a new recipe

Symbol:

This command (category 'Recipe commands') is used to add a new recipe to the currently active recipe definition table. The dialog **New Recipe** will open where you have to enter the recipe name as a string. If you want to **copy the values or an existing** recipe, choose the desired one from the selection list.

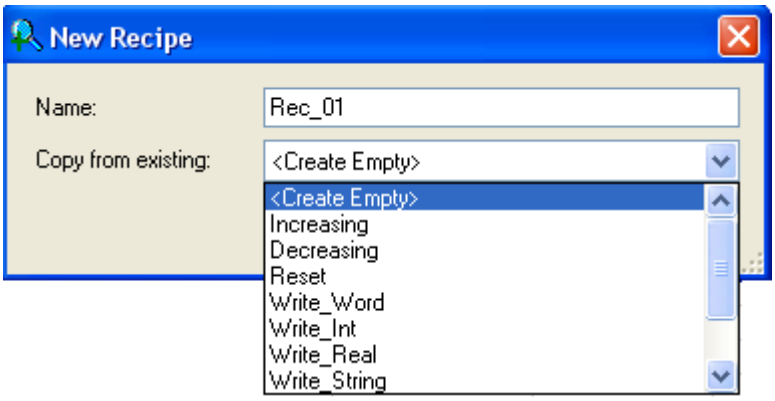


Abb. 246 New Recipe dialog

Appropriately an additional column will be inserted in the recipe definition.

Note: A new recipe also might be created in online mode via an appropriately configured visualization element (Input configuration, category 'Internal command').

12.25.3 Remove recipes

Symbol:

This command (category 'Recipe commands') is used to remove an existing recipe from the currently active recipe definition table.

Select one of the fields in the column describing the respective recipe and perform the command. The column will be removed.

Note: A recipe also might be removed in online mode via an appropriately configured visualization element (Input configuration, category 'Internal command').

12.26 SmartCoding

The command category 'Smart Coding' provide functions to support coding, like for example Input Assistant and Autodeclaration dialogs.

12.26.1 Input Assistant

Smart Coding commands - Input Assistant

The 'Input Assistant' plug-in provides for command category 'Smart Coding' a command for calling the Input Assistant dialog.

By default the command is part of the 'Edit' menu. If necessary open the Customize dialog to view resp. modify the menu configuration.

Input Assistant

Symbol:

The Input Assistant dialog resp. the command 'Input Assistant' (command category Smart Coding) only will be available if currently the cursor is placed in a text editor window. The dialog offers all project items available for being inserted at the current cursor location.

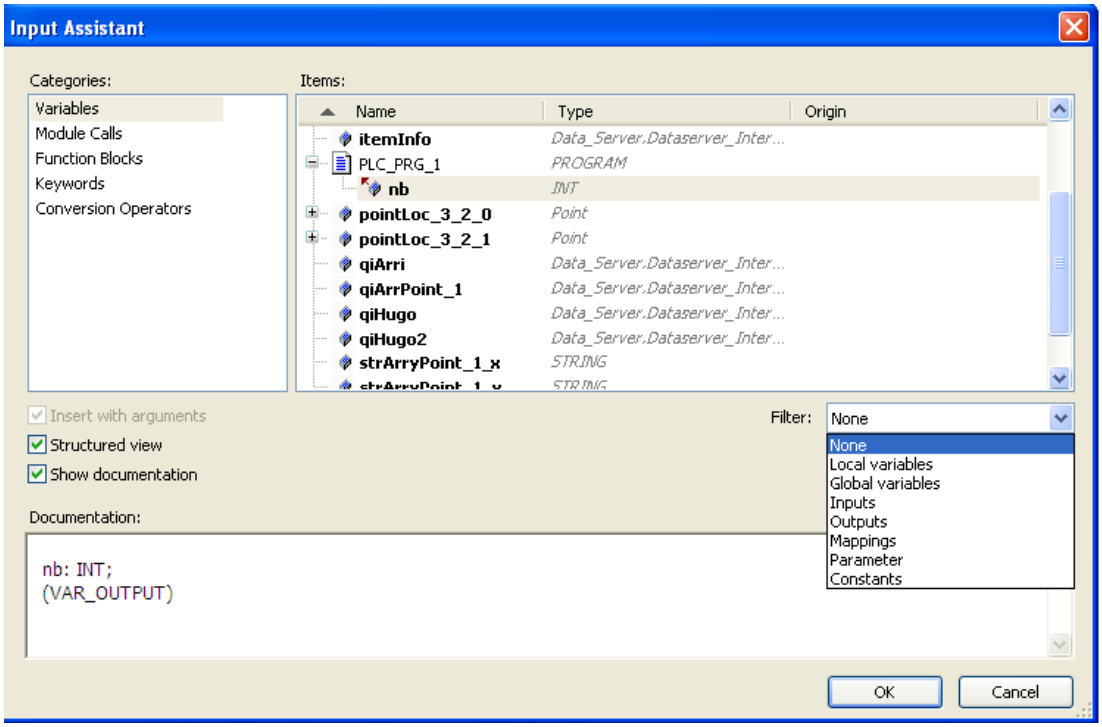


Abb. 247 Input Assistant dialog

The items are sorted by **Categories**. For category "Variables" additionally a **Filter** can be set to a certain type of variables like e.g. Local variables, Global variables, Constants etc.

For the currently selected category the available **items** and - depending on the category - also their data Type and Origin are displayed in the Items window. The origin will be shown for I/O variables (path within device tree) and library-defined variables (library name and category). If option Structured view is activated, the items will be displayed in a structure tree supplemented with icons, otherwise they will be arranged 'flat', but each showing the POU belonging to (for example 'GVL1.gvar1'). The items can be sorted by Name, Type or Origin in ascending or descending alphabetic order by a mouse-click on the respective column header (up-arrow resp. down-arrow symbol).

Note: If there are objects with the same name available in the global area (POUs tree) as well as below an application (device tree), only 1 entry will be offered in the input assistant, because the usage of the object is determined by the usual call priorities (first the application-assigned object, then the global one).

If option **Show documentation** is activated, the dialog will be extended by the Documentation field. If the selected element is a variable and an address is assigned to this variable resp. a comment has been added at its declaration, those will be displayed here.

Insert with arguments: If this option is activated, items which include arguments, like for example functions, will be inserted with those arguments. For example: If function block FB1, which contains an input variable fb1_in and an output variable fb1_out, is inserted with arguments, the following will be written to the editor: fb1(fb1_in:= , fb1_out=>)

12.26.2 Auto Declare...

Smart Coding commands - Auto Declare

The 'AutoDeclare' plug-in provides the command Auto Declare.... for command category 'Smart Coding'.

By default the command is part of the 'Edit' menu. If necessary open the Customize dialog to view resp. modify the menu configuration. Notice that some options can be set concerning the SmartCoding functions in the Options dialog.

Declare...

Default Shortcut: <Shift> + <F2>

This command - provided by the AutoDeclare plug-in - opens the 'Auto Declare' dialog for the declaration of a variable. For this purpose the cursor must be placed in a line of the implementation part of the editor, which contains an undeclared variable or an already variable must be selected. If the dialog should open automatically as soon as a line containing a not yet declared variable is left, the respective option in the SmartCoding Options must be activated.

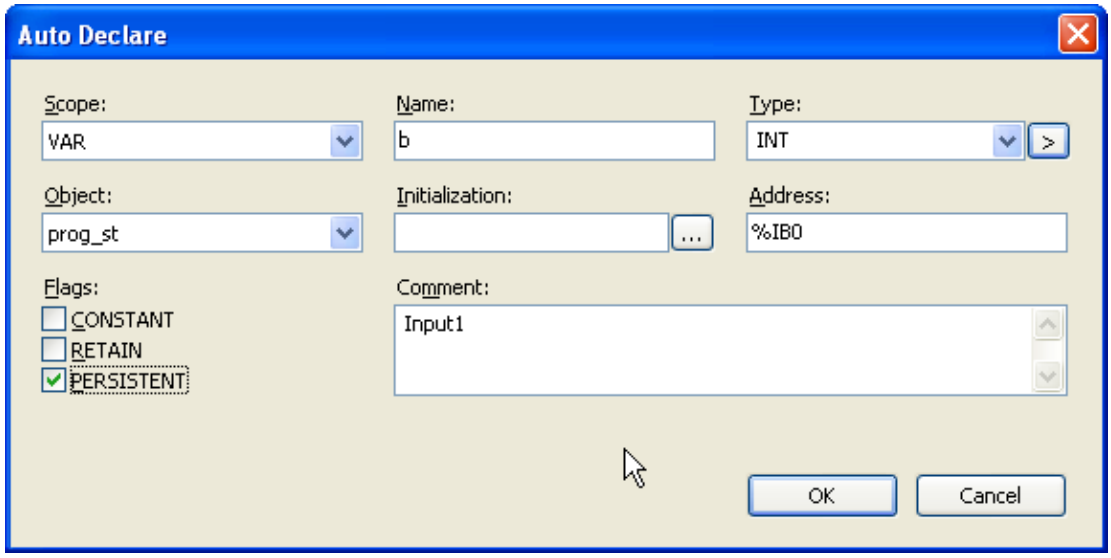


Abb. 248 Dialog Box for Declaration of Variables

Some fields will be filled automatically with default values, but still can be edited:

Name: By default: the name of the new variable which you have entered in the editor.

Object: By default: the name of the currently edited object. To define another object where the variable declaration should be performed, select one of the available objects. For example, if you are going to declare a global variable (Scope: VAR_GLOBAL), here you will get all global variables lists already defined within the project.

Type: By default: INT, if this is the first variable in the line: INT, but, if there is already a declared variable in the line, the type of this variable will be pre-set. For modifying this entry you can press button to get the Input Assistant dialog which allows you to select one from all possible data types. In case you want to declare an array variable you might use the array wizard which is offered also via the arrow button. See below for a description.

Scope: By default: VAR (local variable). Alternatively set another scope from the selection list.

Initialization: Here you can enter an explicit initialization value for the variable. If nothing is entered here, the variable will be initialized with the default value. Not yet implemented: Press button to get assistance by.....

Address: The variable being declared can be bound to an IEC address ("AT" declaration). Example: variable "b" of type "INT" and Address "%IB0" -> declaration: "b AT %IB0 : INT;"

Comment: If applicable, enter a comment. The comment text can be formatted with line breaks by using the key combination <Ctrl> + <Enter>. It will appear in the declaration part of the object in the line above variable declaration.

Flags: CONSTANT, **RETAIN**, **PERSISTENT**: Activate the desired option to define whether you are dealing with a constant or a remanent variable. The respective attribute will be added to the keyword VAR, for example "VAR CONSTANT" initiating the declaration part for the variable. The PERSISTENT option only will be available if a Persistent Variables list exists.

Autodeclaration of arrays:

If you want to use the wizard for the declaration of array variables, use the arrow button > behind the Type field and select command **Array Wizard**. The **Array** dialog will open:

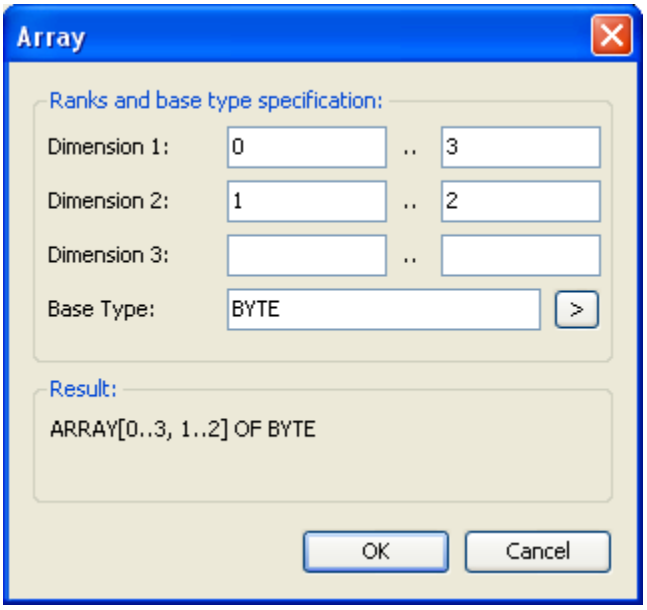


Abb. 249 Array dialog, example

At least the fields marked with an exclamation mark must be filled. Define the **dimension**s by entering the lower and upper limits and the **Base Type** of the variable, whereby the arrow button can be used to get to the input assistant.

In the lower part of the dialog (Result) a preview of the currently configured array declaration will be viewed.

See also the help page on arrays.

By pressing **OK** the declaration dialog will be closed and the variable declaration will appear in the declaration editor in accordance to the IEC syntax.

12.27 SFC

The command category 'SFC' provides commands for working in the SFC Editor.

12.27.1 SFC Editor commands

The commands of category 'SFC' are provided by the 'SFC Editor' plug-in and are available for programming in the SFC editor.

By default they are to be found in the "SFC" menu. If necessary, open the Customize dialog, category 'Menu', to view or modify the current menu configuration.

The commands:

- Init step
- Insert step-transition
- Insert step-transition after
- Parallel
- Alternative
- Insert branch
- Insert branch right
- Insert action association
- Insert action association after
- Insert jump
- Insert jump after
- Insert macro
- Insert macro after
- Zoom into macro
- Zoom out of macro

12.27.2 Init step

This command (category 'SFC') is used in the SFC editor to transform the currently selected step to an init step.

Thus the frame of the step element will change to a double-line. The step previously having been the init step will automatically change to a normal step and get displayed now with a single line frame. This transformation might be useful if you want to reconstruct an existing chart.

When creating a new SFC POU automatically an init step element will be inserted followed by a transition (TRUE) and a jump back to the init step.

Notice the possibility to set back the SFC to the init step by using variables SFCInit resp. SFCReset.

12.27.3 Insert step-transition

Symbol:

Batch command: file open

This command (category 'SFC') is used in the SFC-Editor to insert a step and a transition **before** the currently selected position.

The positioning (sequence) of the new step and transition depends on whether a step or transition have been selected when performing the insert command. Automatically the sequence step-transition-step-transition-... will be kept. See the following picture for examples:

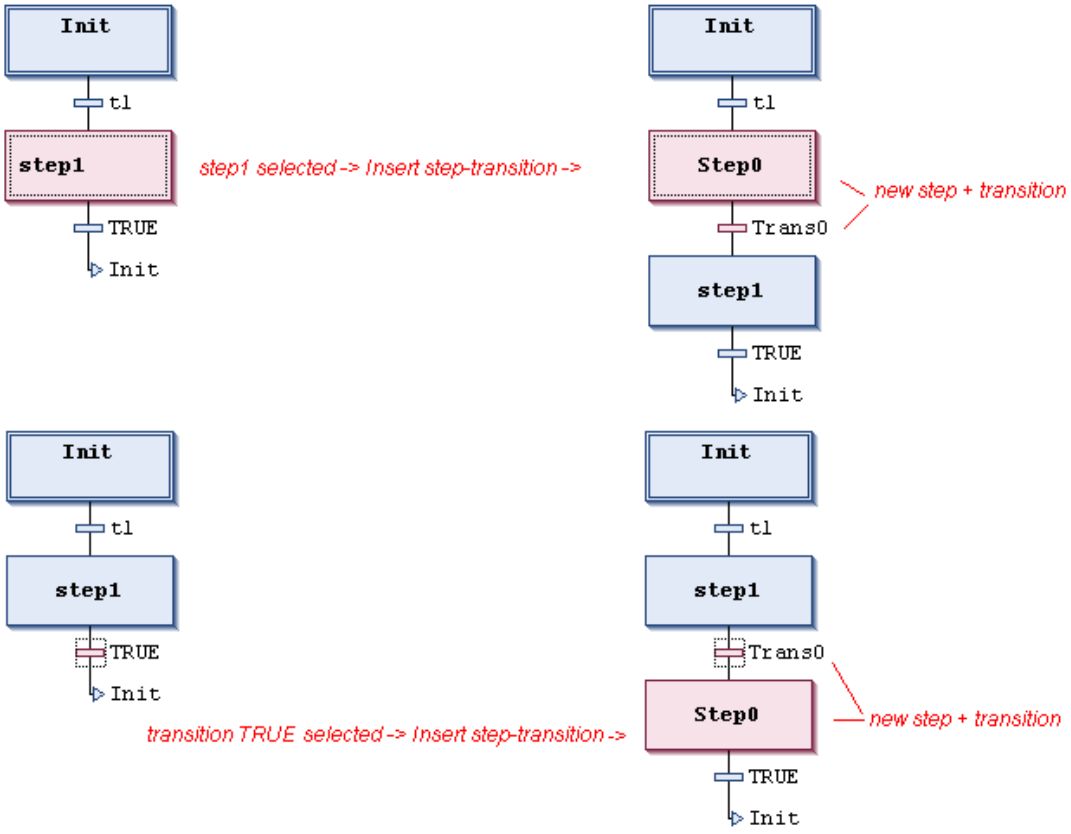


Abb. 250 Step and Transition inserted

The new step by default is named "Step<n>". n is a running number starting with "0" for the first step which is inserted in addition to the init step.

The new transition correspondingly by default is named "Trans<n>".

To modify the default names perform a mouse-click on the name string to get it editable.

12.27.4 Insert step-transition after

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a step and a transition **after** the currently selected step or transition.

The positioning (sequence) of the new step and transition depends on whether a step or transition have been selected when performing the insert command. Automatically the sequence step-transition-step-transition... will be kept.

In this example the new step and transition are placed after transition TRUE, which had been selected when performing the insert command.

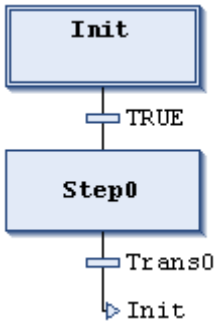


Abb. 251 Step and Transition inserted, example

The new step by default is named "Step<n>" whereby n is a running number starting with "0" for the first step which is inserted in addition to the init step.

The new transition correspondingly by default is named "Trans<n>".

To modify the default names perform a mouse-click on the name string in order to get into the edit mode.

12.27.5 Parallel

Symbol:

This command (category 'SFC') available in the SFC Editor transforms the currently selected alternative branch to a parallel branch.

Notice that after a branch transformation you must check and adapt the chart appropriately, that is you must arrange steps and transitions as required for the respective type of branching.

See also: Alternative

12.27.6 Alternative

Symbol:

This command (category 'SFC') available in the SFC Editor transforms a parallel branch to an alternative branch.

Notice that after a branch transformation you must check and adapt the chart appropriately, that is you must arrange steps and transitions as required for the respective type of branching.

See also: Parallel

12.27.7 Insert branch

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a branch **left** to the currently selected element(s).

See Insert branch right for further information. The difference is just that the branch is inserted to the right of the currently selected step.

12.27.8 Insert branch right

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a branch **right** to the currently selected element(s). (To insert it each left to the currently selected step, use command Insert branch.)

- If the uppermost element of the current selection is a transition or an alternative branch, an alternative branch will be created.
- If the uppermost element of the current selection is a step, a macro, a jump or a parallel branch, a parallel branch with **label** "Branch<x>" will be inserted. This is a default label name where x is a running number. You can edit the label name. The branch label might be used as a jump target.
- If currently a common element of an existing branch is selected (horizontal line), the new branch will be added to the existing branches on the right most position. If currently a complete arm of an existing branch is selected (horizontal line), the new branch will be added directly right to that one.

Note: Notice that branches can be transformed by commands 'Alternative' and 'Parallel'.

Examples:

Parallel branch:

In the following picture see a new parallel branch, created by command 'Insert branch right' when step11 was selected. Automatically a step ("Step2" in the example) gets inserted.

Processing in online mode: When t2 is TRUE, Step2 will be executed immediately after step11 before t3 is noticed. So both branches will be executed, in contrast to alternative branches.

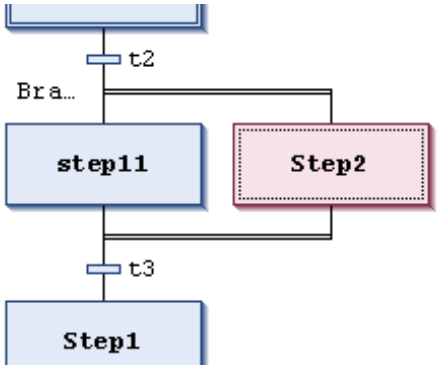


Abb. 252 Parallel branch

Alternative branch:

In the following picture see a new alternative branch, created by command 'Insert branch right' when transition t4 was selected. Automatically a step ("Step32") and a preceding and a subsequent transition (t41, t42) get inserted.

Processing in online mode: When Step3 is active, the following transitions (t4, t41) will be checked from left to right. The first branch whose transition is found to be TRUE, will be executed. Thus only one branch is executed, in contrast to parallel branches.

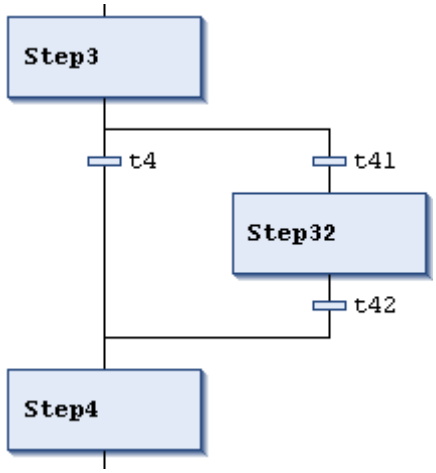


Abb. 253 Alternative branch

12.27.9 Insert action association

Symbol:

This command (category 'SFC') is used in the SFC-Editor to associate an action to a step.

Select the desired step and perform the command. The action box will be inserted right to the step box.

If already one or several action(s) are associated to a step, the new action element will be placed

- as first action (upper position) for the step, if the step has been selected when performing the insert command

- directly before the action, which was selected when performing the insert command.

See also Insert action association after.

The left part of an action box contains the action **qualifier**, by default "N", in the right part an **action name** must be entered. For this purpose click on the field to open an edit frame. The action must be available in the project. See 'Add Object' for creating a new action.

The qualifier also can be edited inline.

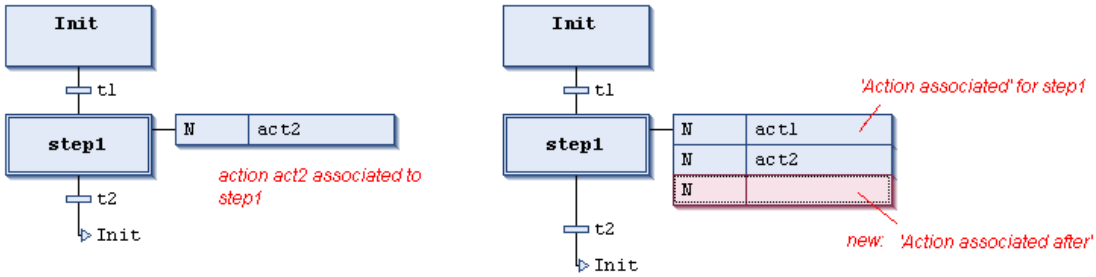


Abb. 254 Actions associated to a step, examples

12.27.10 Insert action association after

Symbol:

This command (category 'SFC') is used in the SFC-Editor to associate a further action to a step after an existing one.

Select the desired step and perform the command. The action box will be inserted right to the step box.

If already one or several action(s) are associated to a step, the new action element will be placed

- as last action (lowest position) associated to the step, if the step was selected when performing the insert command

- directly after the action, which was selected when performing the insert command.

See also and for a picture of examples: Insert action association.

The left part of an action box contains the action **qualifier**, by default "N", in the right part an **action name** must be entered. For this purpose click on the field to open an edit frame. The action must be available in the project. See 'Add Object' for creating a new action.

The qualifier also can be edited inline.

12.27.11 Insert jump

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a jump element **before** the currently selected element.

The new jump automatically is provided with "Step" specifying the target of the jump. Replace this string by the name of a step or by the label of a parallel branch which should be jumped to.

12.27.12 Insert jump after

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a jump element **after** the currently selected element.

Jumps may only be used at the end of an alternative branch.

The new jump automatically is provided with "Step" specifying the target of the jump. Replace this string by the name of a step or by the label of a parallel branch which should be jumped to.

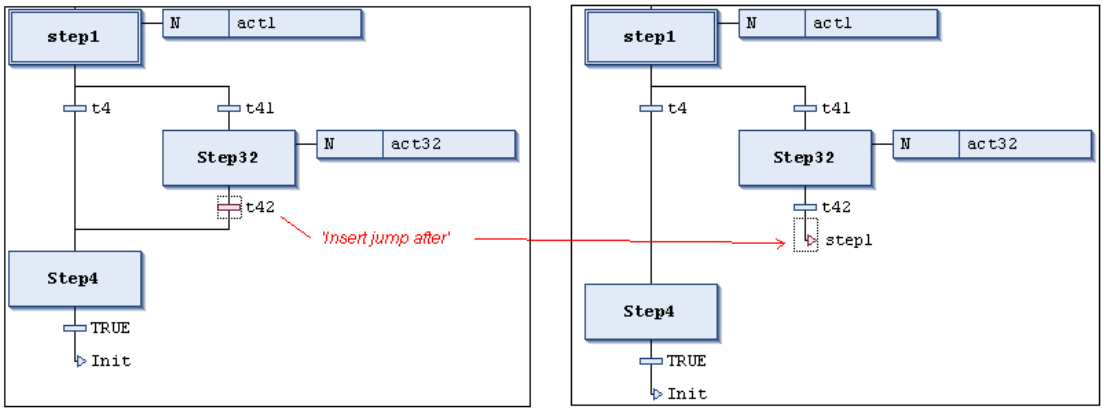


Abb. 255 Example, Insert jump after

12.27.13 Insert macro

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a macro box **before** the currently selected position in the diagram. By default the macro name "Macro<x>" will be entered in the box, whereby x is a running number. You can edit the macro name.

To edit or view a macro the macro editor can be opened via command Zoom into macro.

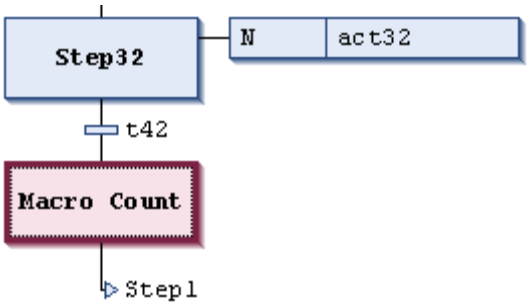


Abb. 256 Example, Macro (currently selected) in SFC diagram

See also: Insert macro after

12.27.14 Insert macro after

Symbol:

This command (category 'SFC') is used in the SFC-Editor to insert a macro after the currently selected position in the diagram. See Insert macro (which is used to insert a macro before the current position) for further information.

12.27.15 Zoom into macro

Symbol:

This command (category 'SFC') is used in the SFC-Editor to zoom into a macro, that is to open the **macro editor** view.

The command can be used in offline and online mode.

Select the macro box in the SFC diagram and perform the command. The main SFC editor view will disappear and instead the macro editor will be opened. Here you can edit or just view the section of the chart which is just represented by the macro box in the main SFC view. The zoom menu as usual for editor views is available in the lower right corner.

To return to the SFC standard view by command Zoom out of macro.

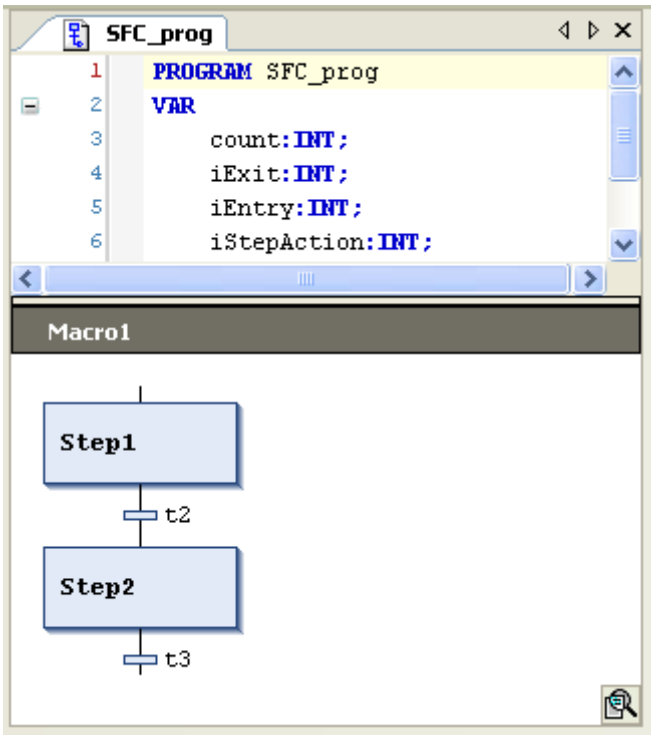


Abb. 257 Macro editor, example

12.27.16 Zoom out of macro

Symbol:

This command (category 'SFC') is used in the SFC-Editor to close the macro editor which is currently opened (via Zoom into macro) in order to return to the main SFC editor view. The command can be used in offline and online mode.

12.28 Source Control

The command category 'SourceControl' contains commands for handling the objects of a project in an external data base via the ENI (Engineering Interface).

12.28.1 Source Control commands

Commands of category 'Source Control' are provided by the 'Source Control' plug-in and are available for handling the objects of a project in an external data base via the ENI (Engineering Interface).

By default they are to be found in the "Project" menu in a submenu "Source Control". If necessary, open the Customize dialog, category 'Menu', to view or modify the current menu configuration.

Notice that as a precondition for working on a database project you must have a valid user account in the ENI Server and be logged in to the respective ENI Server. By default the credentials entered at the last login will be stored until you log in again.

The commands:

- Change Source Control...
- Open From Source Control...
- Add Objects From Source Control
- Get Latest Version (Entire Project)
- Get Latest Version (Selected Objects)
- Checkout...
- Checkin...
- Undo Checkout
- Show Differences
- History...
- Project History...
- Label...
- Refresh Status
- Login...

Notice also the command 'Pending Checkins' (category 'View').

12.28.2 Change Source Control...

Symbol:

This command (category 'Source Control') opens the **Change Source Control dialog**. Here the connection to an external database via ENI can be set up and activated. Further on the bindings of the project objects to the database locations might be viewed and modified.

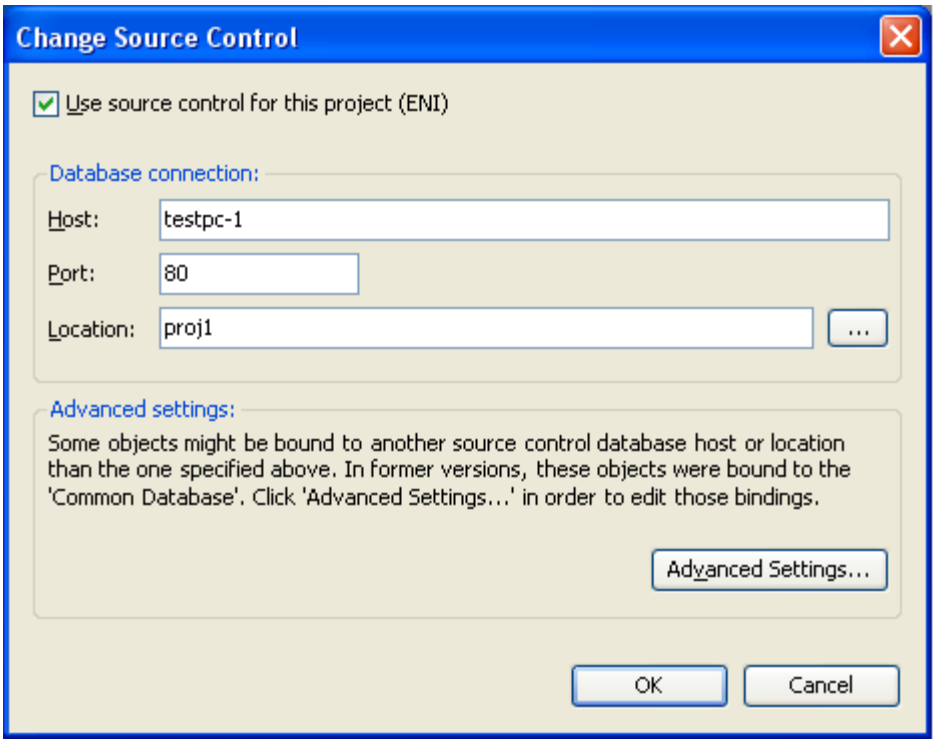


Abb. 258 'Change Source Control' dialog

Use source control for this project (ENI): When this option is activated, the project will be set under source control. That means that the project objects are handled in an external data base, the communication to which is done by an ENI Server. The connection to the ENI Server is defined in the 'Database connection' field:

Database connection:

Host: Address of the computer where the ENI Server is running; for example "192.168.100.18", "testpc-1", "localhost".

Port: This port number must be the same as set in the configuration parameters of the ENI Server; default: 80.

Location: Name of the data base folder where the project objects should be stored by default.

If you want to use an **already existing location**, you might use button ... to get the **Browse Location dialog** which shows a folder tree of all currently available data base locations. Select the desired location and confirm with OK to get it entered in the 'Location' field. When going to close the Change Source Control dialog with OK, you will get the message that the chosen location is already in use and that the project stored there might differ to the current one. At this time you can finally decide to use the location anyway or to cancel.

If you want to use a **new database location** for the current project, just enter a name for this location. When going to close the Change Source Control dialog with OK, you will be asked whether you want to create this new location, that is create a new database folder with the given name.

After having defined the existing or new location definitely, a dialog will appear asking whether all objects of the project should be bound to this location.

If Yes, the objects will be indicated with database symbols in the POUs and Devices view. The objects bound to the default location will be indicated by yellow symbols: PLC_PRG (PRG) . If an object is already found in the database, you will be asked whether it should be overwritten there.

If No, the objects will stay "unbound", that is will not be stored in the database. You might add them resp. edit the binding status still later via the Object Properties dialog or in the 'Advanced Settings' dialog as described in the following:

Advanced Settings: This button opens the 'Advanced Source Control Settings' dialog. An overview on the bindings of all project objects is provided in a table view.

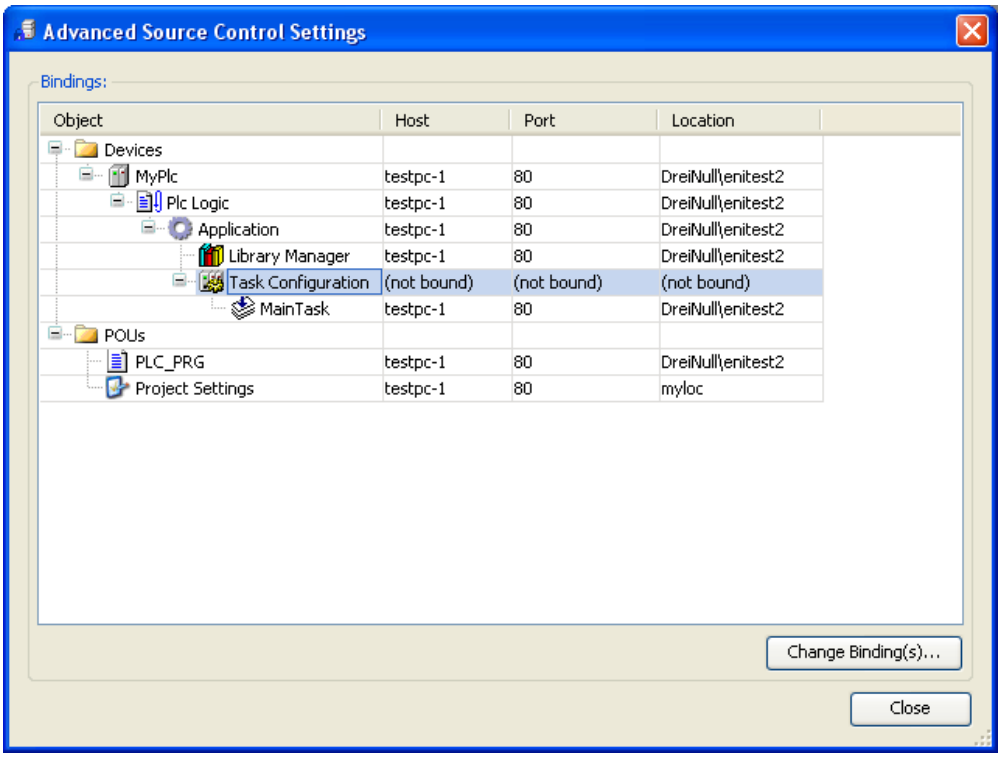


Abb. 259 Advanced Source Control Settings dialog

For each object you see the currently defined database location (Host, Port, Location). In order to edit one or several bindings, select one or several objects and use button Change binding(s)... to open the Change binding(s) dialog. If you want to change the binding of only one object, it is sufficient to perform a double-click on the respective line.

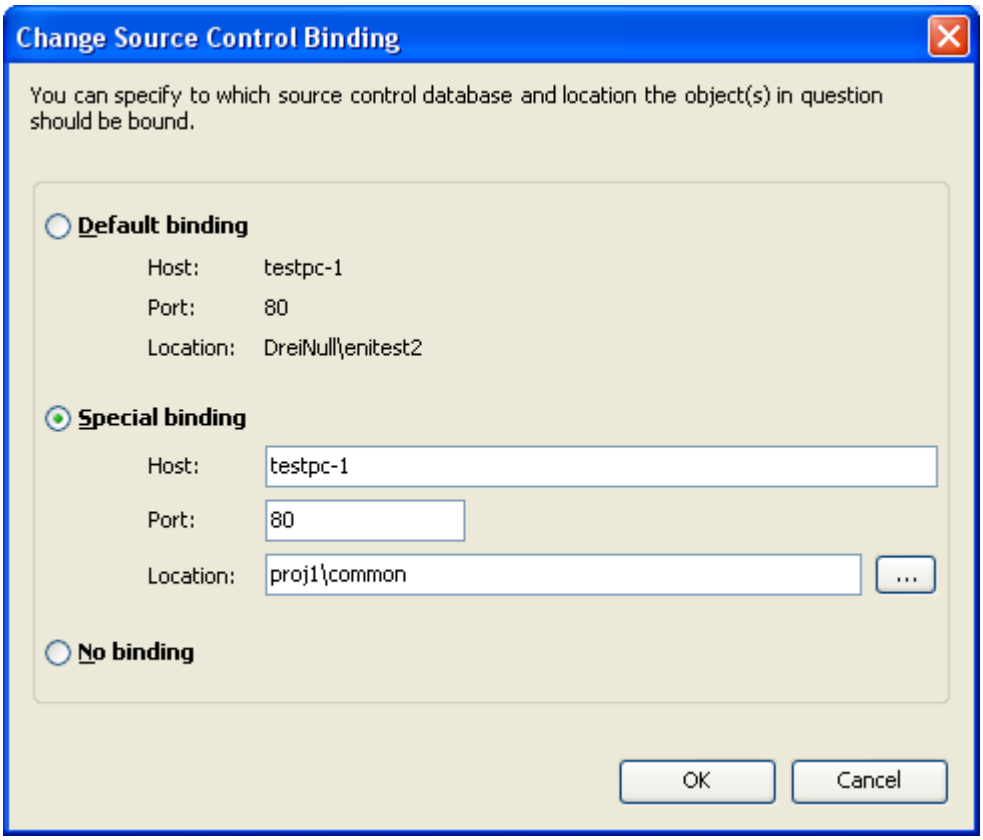


Abb. 260 Change Source Control Binding dialog

In the Change Source Control Binding dialog activate one of the following binding options:

Default binding: Activate this option, if the object(s) should be bound to the default database location which has been defined in the upper part of the dialog (Database connection). Notice: The object will also be kept in the current version at the previous location, that is will be **copied** to the default location!. An object with default binding will be indicated in the POUs or Devices tree by a yellow symbol, for example:
PLC_PRG (PRG)

Special binding: Activate this option, if the object(s) should be bound to another than the default location (default binding) and define this location. For information on how to fill the Host, Port and Location field see above, 'Database connection'. This option for example might be used to replace the "Common" database function ("Shared objects") which was used in »PLC Designer« V2.3. If already an object with the same name exists at the chosen new location, you will get an appropriate dialog showing the possible processings. Notice: If an object gets bound to another location, it will be deleted from the default location, that is **shifted** to the new location! An object with a special binding will be indicated in the POUs or Devices tree by a blue symbol, for example:
control1 (PRG)

No binding: Activate this option, if the object(s) should not be under source control. This means that the object(s) only will be stored within the project, which corresponds to object category "Local" used in »PLC Designer« V2.3.

12.28.3 Open From Source Control...

Use this command (category 'Source Control') if you want to open a project which is stored in a ENI project database. The **Open From Source Control dialog** will appear, where you have to enter the appropriate database connection parameters and a file location where the project should be stored locally.

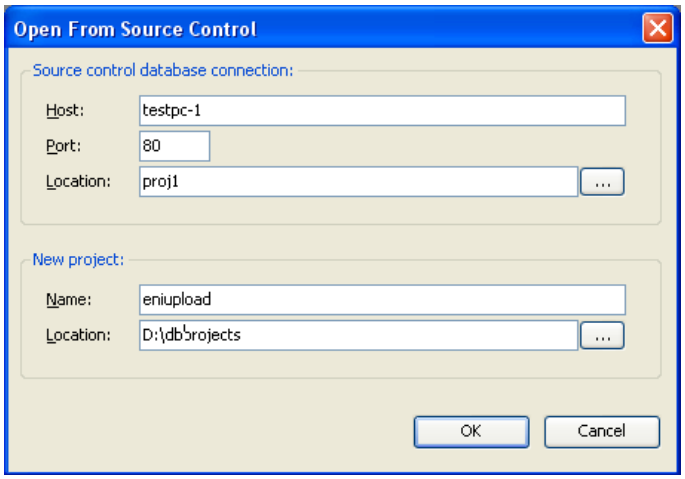


Abb. 261 'Open From Source Control...' dialog

Source control database connection:

Concerning the database connection settings (**Host**, **Port**, **Location**) see the respective description here: Change Source Control... If the specified location does not contain a project, an appropriate message will appear.

New project:

Enter a **Name** for the project as it should be stored on your local system and define the path where it should be stored. You might use the ... button for browsing for a **Location**.

If the specified project already exists, an appropriate message will appear.

12.28.4 Add Objects From Source Control

This command (category Source Control) by default is part of the submenu 'Source Control' in the 'Project' menu.

It serves to add particular objects to the currently opened project. The **Add Objects From Source Control dialog** will open, where you can browse the database locations for the desired object(s). Multiple selection of objects is possible.

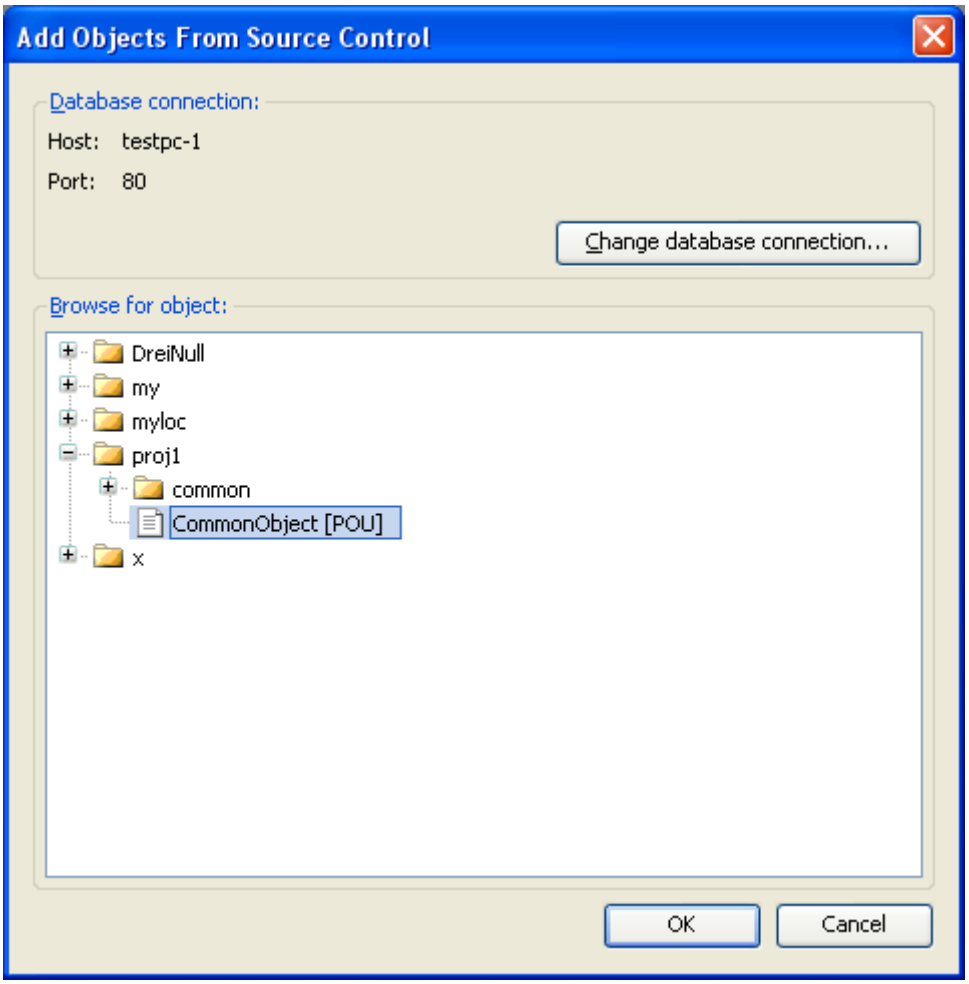


Abb. 262 'Add Objects From Source Control' dialog

If you want to add objects from another ENI database than that you are currently connected to, use button **Change database connection...** where you can enter the Name (**Host**) and **Port** of the respective ENI Server (**Change Database Connection dialog**). After having logged in, the Add Objects.. dialog will provide the database locations tree of the other database.

The selected objects will be added to the current POUs resp. Devices tree. The default database location automatically will be that one, from where they were loaded and all objects (exceptions: Library Manager objects) will be bound to this location. To change the bindings use the Object Properties resp. Change Source Control dialogs.

12.28.5 Get Latest Version - Entire Project

Symbol:

This command (category Source Control) copies the current version of all project objects from the database and each overwrites the local version in the current project. In contrast to the Check Out action the object will not be locked for other users in the data base.

12.28.6 Get Latest Version - Selected Objects

Symbol:

This command (category Source Control) copies the current version of all project objects which are currently selected in the POUs or Devices window from the database and each overwrites the local version. In contrast to the Check Out action the object will not be locked for other users in the data base.

12.28.7 Checkout...

Symbol:

This command (category Source Control) checks out the object(s), currently marked in the POUs or Devices window, from the data base. From then on the object(s) will be locked for other users.

The Checkout dialog will open where a **comment** can be entered to be stored in the version history of the object in the database. Line breaks can be inserted by <Ctrl>+<Enter>.

If you activate option **Do not get the latest version**, the object will be marked as "checked-out" in the database, but the latest version will not be copied automatically to the local project. This might be useful if you just want to lock the object for other database users, but do not want your current local object version to get overwritten by the database version.

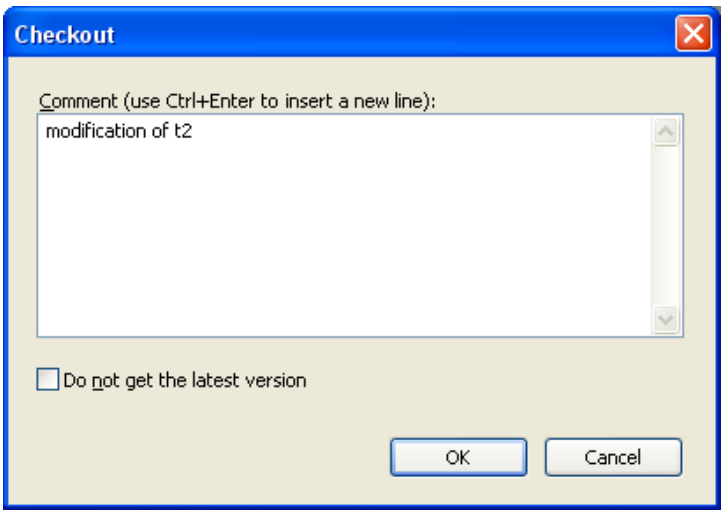


Abb. 263 'Checkout' dialog

If the database version of the object to differs from that in the local project, an appropriate message will be displayed and the user can decide whether the object should be checked out anyway.

After the dialog has been closed with OK, the checked-out object will be marked with a green check in thePOUs or Devices window of the local project. For other users it will be appear marked with a red cross and will not be editable by them. To check in an object use the Checkin... command.

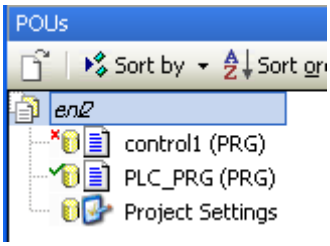


Abb. 264 Checkout symbols

12.28.8 Checkin...

Symbol:

This command (category Source Control) checks in the checked-out object(s), currently marked in the POUs or Devices window, to the data base. Thereby a new version of the object will be created in the data base. The old versions will be kept anyway.

The Checkin dialog will open where a **comment** can be added which will be stored in the version history of the object in the data base. Line breaks can be inserted by <Ctrl>+<Enter>.

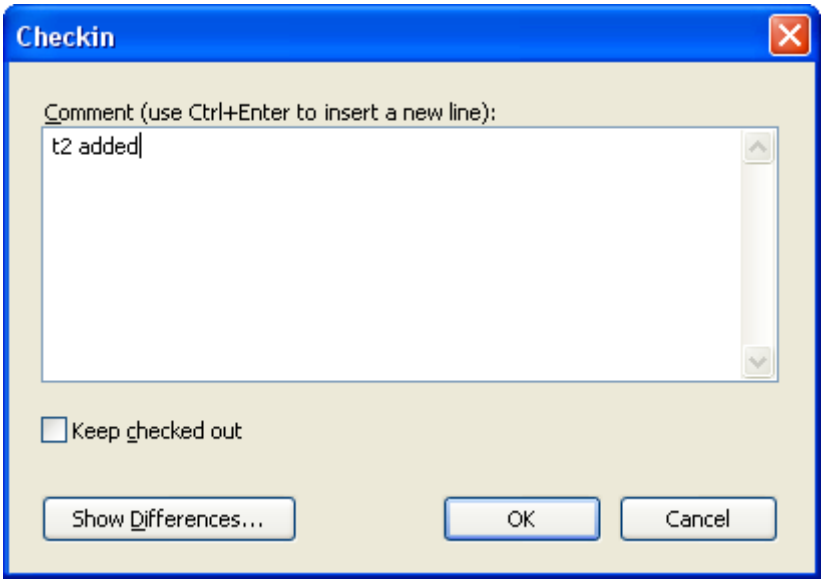


Abb. 265 'Checkin' dialog

Keep checked out: If this option is activated, the current version of the object will be written to the database, but the object will remain in status "checked-out".

Show differences: If you use this button, you will get information on whether the local object version differs from the latest version in the database. (Currently the differences will not be shown in detail, but only the information whether the object versions are different or not.)

After the dialog has been closed with OK the green check in front of the object name in the Object organizer will be removed, except the option 'Keep checked out' is activated.

Notice also the dialog 'Pending Checkins' dialog, available for listing and handling the currently checked-out objects.

12.28.9 Undo Checkout

Symbol:

This command (category Source Control) cancels the Checking out of the object which is currently marked in the POUs or Devices window.

If you have made any modifications in the local object version, you will be asked whether you want to get the local version overwritten by the latest database version, which means that the local modifications would be canceled. The object version in the database will remain unmodified and the object will be accessible again for other users. The red cross in front of the object name in the POUs or Devices window will disappear.

12.28.10 Show Differences

Symbol:

This command (category Source Control) gives information on whether the local version of the object differs to the database version.

(Currently the differences will not be shown in detail, but only the information whether the object versions are different or not.)

12.28.11 History...

Symbol:

This command (category Source Control) opens the History dialog providing a list of the database actions (for example Check-in, Label etc.) which have been performed on the currently selected object. (See also the 'Project History' dialog provided for the entire project.)

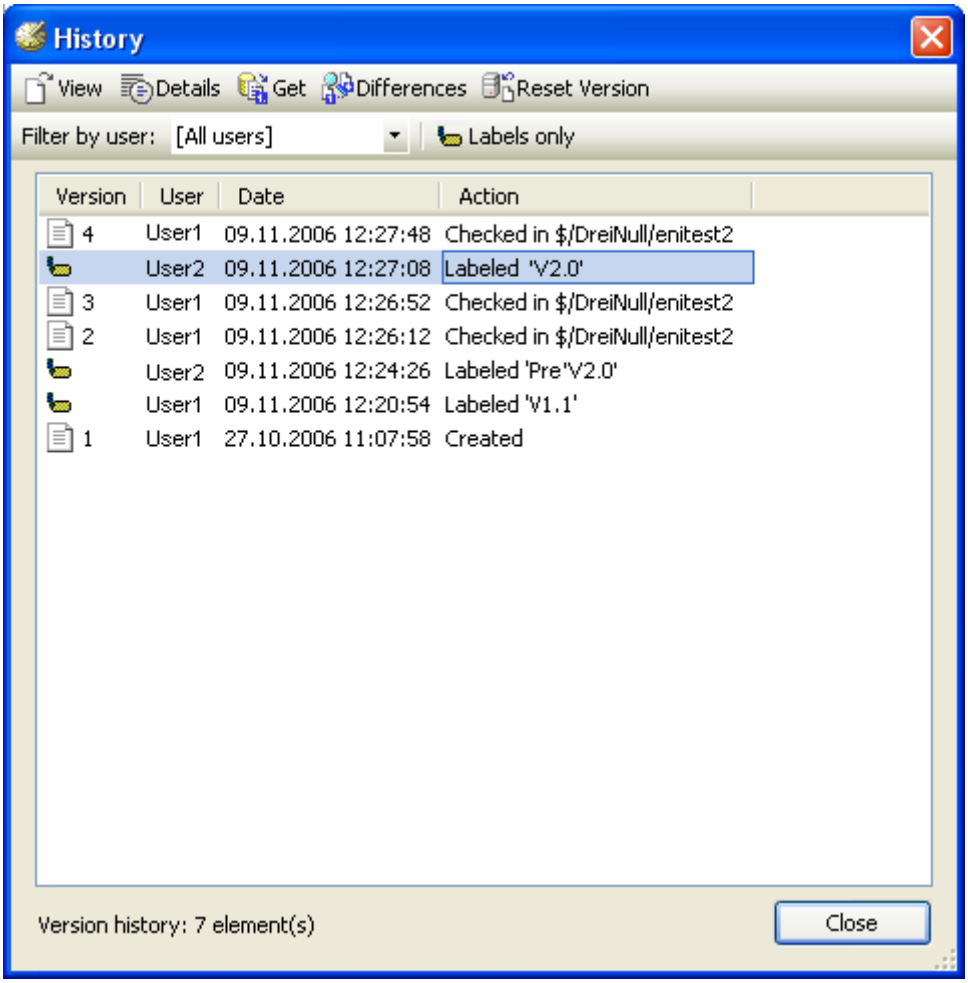


Abb. 266 'History' dialog, example

The list shows the following:

Version: Data base specific numbering of the versions of the object which have been checked in one after the other. Labeled versions get no version number but are marked by a label-icon ( ).

User: Name of the user, who has executed the check-in or labeling action.

Date: Date and time stamp of the action.

Action: Type of the action. Possible types: 'created' (the object has been checked in to the data base for the first time), 'checked in' (all check-in's of the object excluding the first one) and 'labeled with <label>' (a label has been assigned to this version of the object)

By a double-click on an entry the **Version Details dialog** will open. See also below: command "Details".

Possible filters:

Filter by user:By default the actions performed by all users are listed, but here you might set a filter on a particular user.

Labels only: Use this button to toggle between the display of just labeling actions and the display of all actions.

Commands:

View:	The currently selected object version will be displayed in the respective editor, however cannot be edited.
Details:	The Version Details dialog for the currently selected object version will open (like on a double-click on the entry). It shows Version number, User name, Date and type of Action and additionally the **Comment** which might have been added during an action. additionally show the which might have been added during the action. The **Previous** and **Next** arrow buttons in this dialog will jump to the Version Details dialog for the previous or next entry of the history list.
Get:	The selected object version will be loaded to the local project and overwrite the local version.
Differences:	Select two entries in the list and use this command to check whether they are different. Currently the differences will not be shown in detail, but only the information whether the object versions are different or not.
Reset Version:	The currently selected object version will be made the "latest" (newest). Example: There are four object versions 1,2,3,4. You select version 2 and use command 'Reset'. A new version "5" will be created, however is exactly the same as version 2. So, version 4, which was the "latest" up to this time and which is currently loaded in your project, now will be overwritten by version 2 resp. 5. If you use this Reset command, you will get asked whether you really want to overwrite the current version with an previous one. Not before you confirmed with Yes, the reset will be done actually.

12.28.12 Project History...

Symbol:

This command (category Source Control) opens the Project History dialog providing a list of database actions (for example Check-in, Label etc.) performed on the currently selected project. You will immediately be asked whether the object histories should also be included in the project. However you also later can switch on and off the object histories by command 'Display object histories', see below.

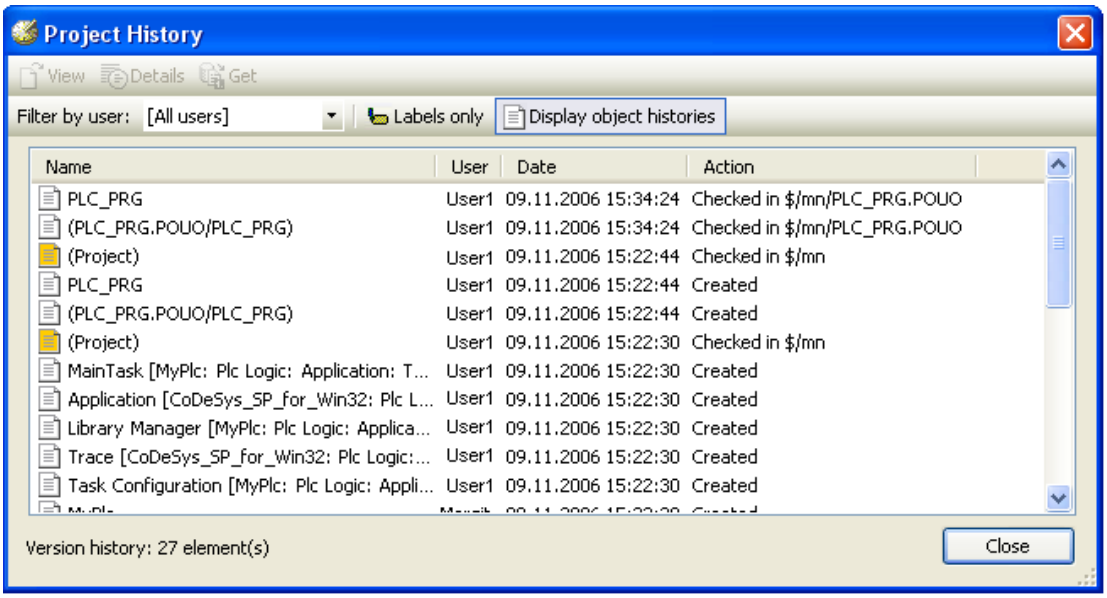


Abb. 267 'Project History' dialog, example

The actions are listed in chronological order. In contrast to the object history no version numbers are used. For each action the following information is given:

Name: Name of the Project resp. the Object,

User: Name of the user who performed the action

Date: Date and Time stamp of the action

Action : Action type. See the description of the History of a particular object for details.

The total number of objects is displayed below the list.

By a double-click on an entry the **Version Details dialog** will open. See also below: command "Details".

Possible filters:

Filter by user:By default the actions performed by all users are listed, but here you might set a filter on a particular user.

Labels only: Use this button to toggle between the display of just labeling actions and the display of all actions.

Display object histories: Use this button to toggle between the display of just actions on the project and the display of project actions plus the Histories for the particular objects. In the latter case the actions on the objects will be inserted in chronological order between the actions on the project. The object actions will be indicated by an icon, project actions by an icon.

Commands:

View:	This command is only available if an object action is currently selected. The object version will be displayed in the respective editor, however cannot be edited.
Details:	The 'Version Details' dialog for the currently selected action will open (like on a double-click on the entry). It shows Version number, User name, Date and type of Action and additionally the **Comment** which might have been added during an action. additionally show the which might have been added during the action. The **Previous** and **Next** arrow buttons in this dialog will jump to the Version Details dialog for the previous or next entry of the history list.
Get:	All objects of the currently marked project version will be called to the local project. Consider that this means, that local objects, which are not part of the project in this database version, will be removed from the project! Objects, which are contained in the database version but not in the local one, will be added to the local project. If the called database version defines other object bindings than the local one, the binding definitions of the database version will overwrite the local ones.

12.28.13 Label...

Symbol:

This command (category Source Control) opens the Label dialog, where you can define a label for the current version of all project objects.

By this label exactly this project version can be reloaded later from the database.

In the dialog insert a **Label** name (for example "Release Version 1.0") and optionally a **Comment**. When you confirm with OK, the dialog will close and the label action will appear in the project history as well as in the history for a single object. A labeled version does not get a version number, but is only marked with a label icon in the 'Version' column. If option 'Labels only' is activated in a History dialog, then only labeled versions will be listed.

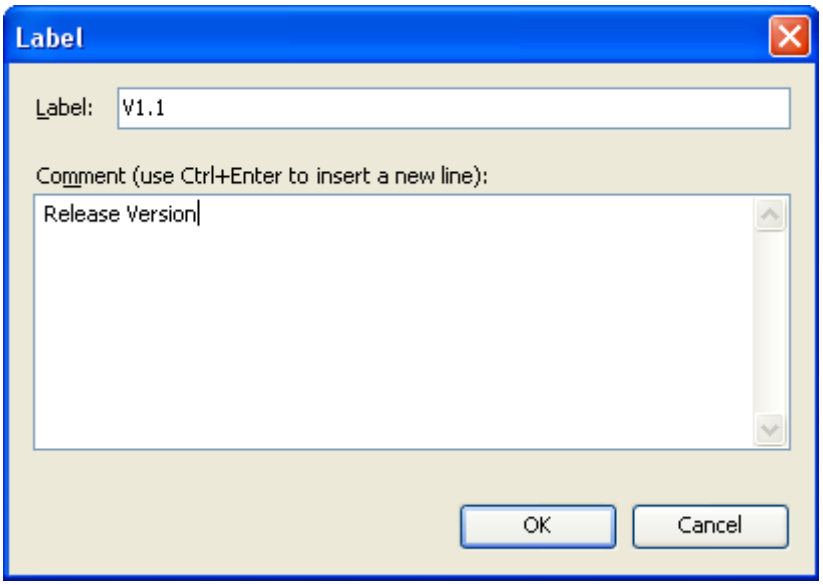


Abb. 268 'Label' dialog, example

12.28.14 Refresh Status

Symbol:

This command (category Source Control) checks the current source control status of all objects. The respective icons will be updated in the POUs and Devices trees and information and possibly errors or warnings resulting from this refresh procedure will be displayed in the Messages window.

12.28.15 Login...

This command (category Source Control) opens the dialog **Login to Source Control** for accessing the project database via the ENI Server. Notice that you must have a valid user account on the ENI Server.

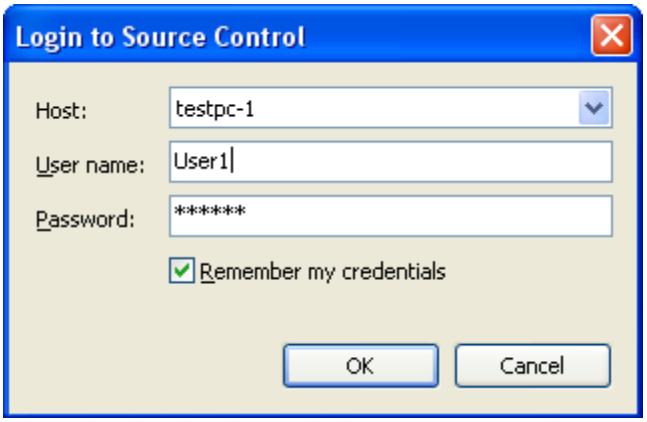


Abb. 269 'Login to Source Control' dialog

Host: Enter the name resp. TCP/IP address of the ENI Server. which provides the connection to the external project database.

User name, **Password**: Credentials for accessing your user account on the ENI Server.

Remember my credentials: If this option is activated, the given user name and password will be remembered until you use the Login dialog again. If the option is not activated, the credentials will only be remembered until »PLC Designer« gets terminated. Note that in contrast to »PLC Designer« V2.3 the authentication is only to be done once per ENI Server and not for each project resp. object category.

12.29 Text Editor

The command category 'TextEditor' provides editing supporting functions.

12.29.1 Text Editor commands

The 'Text Editor Basics' plug-in provides commands for editing any text in the project (command category 'Text Editor') . By default these commands are available in the 'Edit' menu. If necessary open the Customize dialog to view and modify the menu configuration.

The commands:

- Overwrite Mode
- Go To Line...
- Insert File As Text...
- Make Lowercase
- Make Uppercase
- Go To Matching Bracket
- Select To Matching Bracket

12.29.2 Overwrite Mode

Use this command ('category 'Text Editor') to toggle between overwrite mode (option activated) and insert mode (option deactivated. When editing in overwrite mode the existing characters will be overwritten, otherwise the new characters will be inserted.

12.29.3 Go To Line...

Use this command ('category 'Text Editor') to jump to a certain line within a text editor. A dialog ('Go To Line') will be opened where you can insert the desired line number. After closing the dialog with OK, the cursor will be set to the start of the corresponding line.

12.29.4 Insert File As Text...

This command ('category 'Text Editor') can be used to insert the content of a text file to the currently opened text editor. The standard dialog for browsing for a file ('Insert File') will be opened where you can search for the desired file, which must be in text format. The file contents will be inserted at the current cursor position.

12.29.5 Make Lowercase

This command ('category 'Text Editor') will set the currently marked text to lowercase.

12.29.6 Make Uppercase

This command ('category 'Text Editor') will set the currently marked text to uppercase.

12.29.7 Go To Matching Bracket

This command ('category 'Text Editor') will set the cursor at the next matching bracket. This is valid for brackets in program lines as well as for "bracket scopes". The display of bracket scopes can be activated by option 'Matching brackets' (Text Editor options, category Text Area).

12.29.8 Select To Matching Bracket

This command ('category 'Text Editor') will select the code lines up to the next matching bracket. This is valid for brackets in program lines as well as for "bracket scopes". The display of bracket scopes can be activated by option 'Matching brackets' (Text Editor options, category Text Area).

12.30 Textlist

This command category provides commands for creating, editing, exporting and importing text lists.

12.30.1 Text list commands

The commands for working with Textlists are provided with the 'Text lists for multi language support' plug-in.

Per default these commands are available when a Textlist object is edited or in the Visualization menu. For checking resp. customizing the menu structure use the Customize dialog.

The commands:

- Create Global Text List
- Add language
- Remove language
- Import/Export Textlists
- Update Visualization Text Ids
- Check Visualization Text Ids
- Remove Unused Text Ids

12.30.2 Add language

Symbol:

This command of category Text list commands is used to add a new language column to a textlist.

Open the textlist, perform the command and enter the language name in the **Choose language** dialog. After confirming with OK the language column will be added rightmost in the current list.

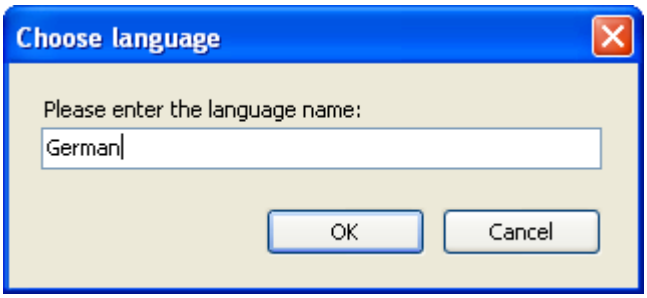


Abb. 270 Dialog Choose language

12.30.3 Remove language

Symbol:

This command of category Text list commands is used to remove that language column from a textlist. Open the textlist, set the cursor in the respective column and perform the command.

12.30.4 Create Global Text List

Symbol:

This command of category Text list commands can be used to explicitly create a global text list. The GlobalTextList object will be added to the POUs view. GlobalTextList is created automatically as soon as the first text is defined in the configuration of a visualization.

12.30.5 Import/Export Textlists

Symbol:

This command (category Textlist) provides the data exchange with other programs as for instance Excel. The data format in use is .csv ("Comma Separated Values"). The executing the command the following dialog appears:

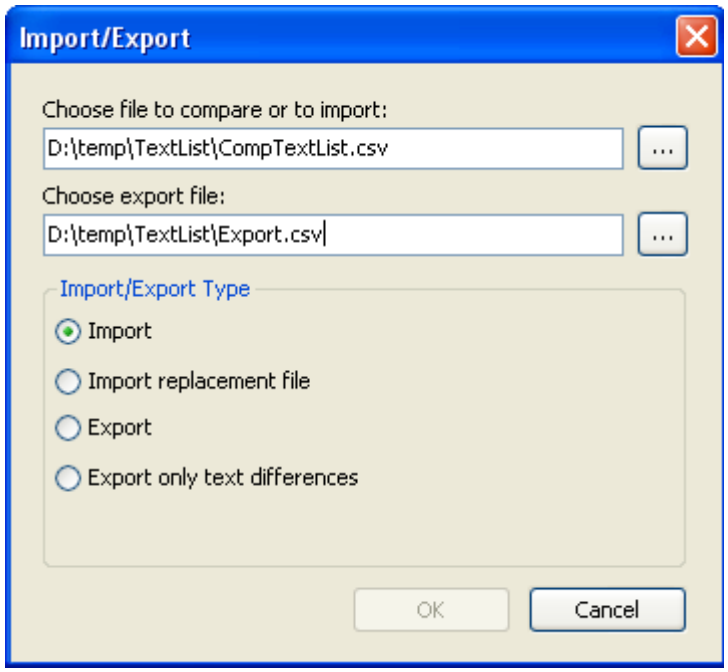


Abb. 271 Dialog - Import/Export Textlists

By entering the corresponding paths or by making use of the input assistant (...) the files to be imported, exported or compared may be specified. Which of the actions should be executed, may be defined by activating the corresponding item in the lower part of the dialog:

Import

Importing an external file its dataset is put in line with the dataset of the project. The dataset in the project is adjusted according to the following rules:

1. If the data content is identical, the data set will be left unchanged.
2. If a translation has been added to the external file, it will be added to the data set of the project as well.
3. If text within a translation has been modified, the modification will be overtaken in the project as well.
4. If translation texts are missing within the external file, the data set in the project will not be modified.
5. If a new line has been added to the external file, this new data record will be incorporated in the data set of the project file.
6. If the project contains an additional line, it will be preserved.
7. A modification within the column Default may be considered as insertion of new text. If however there are text positions containing several empty spaces instead of a single one, then this will not be handled as a modification!

Example - Import of .cvs-file

Data content of external file:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door		
TextList1	2	Light		

Data content of textlist of project before import:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel		
TextList1	0	Cancel	Abbrechen	Abortion
TextList1	1	Door	Tür	Door
TextList2	3	Seat	Sitz	Seat

During the import all differences are incorporated into the project. Thereby the two lists are adapted so that he following textlist will result in the project:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door	Tür	Door
TextList1	2	Light		
TextList2	3	Seat	Sitz	Seat

Import replacement file

While importing a textlist a modification within column „Default“ is considered as an insertion of a new line. The reason is, that the column „Default“ is serving as key for comparing the lines during import/export. If text shall be modified within the column "Default" (elimination of typing error or supplement to existing text), a replacement file becomes necessary.

Example - Import of a replacement file

Default old	Default new	Command
Cancel ?	Cancel	REPLACE
Do you want to register ?	Do you really want to register ?	REPLACE_AND_REMOVE
Do you really want to register ?!	Do you really want to register ?	REPLACE_AND_REMOVE

The replacement file will be executed top down. Thus, the change history might be accounted for.

The command defines what to do with a text line. The only commands available yet is REPLACE. It will have the following effect:

Normally, the text entered in column "Default" will be replaced by the new text. In the example „Cancel ?“ will be replaced by „Cancel“ and "Do you want to register ?" by "Do you really want to register ?". Simultaneously the texts of all visualization elements will be adjusted, i.e. the old text entries within the visualization elements will be replaced.

In the case that the new default text is already contained in the "Default" field of another row of the textlist, the row containing the entry to be replaced will be deleted completely. The visualization elements concerned receive the corresponding entries of the remaining row with the same default entry. In the example this will happen for the default entry „Do you really want to register ?!“ that should be replaced by „Do you really want to register ?“. Due to the change history there will already exist a row with this default entry when the related REPLACE command shall be carried out. To avoid multiple occurrences of the key, the row containing the old Default text „Do you really want to register ?!“ will be deleted completely from the textlist.

Export

Exporting textlists all modifications within the project are compared to an external file. A new export-file will be created according to the following rules:

1. If the data content is identical, the data set will be exported as it is.
2. If a translation has been added to the project file, it will be included.
3. If text within a translation has been modified, the modification in the project will be overtaken in the export file as well.
4. f translation texts are missing within the project file, the translations of the template will be used for the new data set.
5. If a new line has been added to the project, this new data record will be incorporated as new data set of the project file.
6. If the external file contains an additional line, it will be exported again.
7. A modification within the column Default may be considered as insertion of new text.

Example - Export of a .cvs-file

Data set of external file:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel		
TextList1	0	Cancel	Abbrechen	Abort
TextList1	1	Door	Tür	Door
TextList2	2	Seat	Sitz	Seat

Data content of textlists of project before export:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering whee
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door		
TextList1	3	Light		
TextList2				

During the export all differences are incorporated in the external file. Thereby the two lists are adapted so that the following external file will be created:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door	Tür	Door
TextList1	3	Light		
TextList2	2	Seat	Sitz	Seat

Export only text differences

If this option is activated, only the lines differing from their corresponding line in the different versions are included in the export file. Such a difference file is suited as input for translation purposes. As the file is intended to be kept as small as possible, missing items in the actual textlists will not be treated as differences.

Example - Export of text differences only

Data content of external file:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel		
TextList1	0	Cancel	Abbrechen	Abort
TextList1	1	Door	Tür	Door
TextList2	2	Seat	Sitz	Seat

Data content of textlists of project before export:

TextList	Id	Default	Deutsch	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering whee
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door		
TextList1	3	Light		
TextList2				

During the export all lines differing from the corresponding ones (line 2,3 and 5 of the actual list) are included in the export file.

Data content of external file after export:

TextList	Id	Default	Deutsch	English
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	3	Light		

Note: For locating the corresponding data sets the column Default is used for the GlobalTextList and the column Id for all other textlists. Therefore the column entitled Id is empty for all data sets of the GlobalTextList.

12.30.6 Update Visualization Text Ids

Symbol:

This command belongs to category Textlist. If a static text is modified within a visualization element, the visualization and evetually the GlobalTextList as well must have write permission (see. GlobalTextList). If modifications will be done, though the write permission is missing, it can be, that the text-ids do no longer fit to the texts of a visualization element.

By use of the command „Update Visualization-Text -Ids“ these errors can be corrected automatically. Therefore all affected visualizations and the GlobalTextList must have write permission.

12.30.7 Check Visualization Text Ids

Symbol:

This command belongs to category Textlist. If a static text is modified within a visualization element, the visualization and evetually the GlobalTextList as well must have write permission (see. GlobalTextList). If modifications will be done, though the write permission is missing, it can be, that the text-ids do no longer fit to the texts of a visualization element.

By use of the command „Check Visualization-Text-Ids“ such errors can be detected in the visualizations.

12.30.8 Remove Unused Text Ids

Symbol:

This command (category Textlist) serves to delete texts being no longer used in a visualization element from the GlobalTextList.

12.31 Trace

The command category "Trace" provides commands for working with the Trace Editor.

12.31.1 Trace commands

The 'TraceEditor' plug-in provides commands of category 'Trace' for working with the Trace Editor. By default these commands are available in the 'Trace' menu when the editor is active. If necessary open the 'Customize' dialog to view resp. modify the menu configuration (notice category 'View').

The following commands are available:

- Download Trace
- Start / Stop Trace
- Reset Trigger
- Cursor
- Mouse scrolling
- Mouse zooming
- Default appearance
- Compress
- Stretch
- Multi Channel
- Online List
- Upload Trace
- Load Trace...
- Save Trace...

12.31.2 Download Trace

Symbol:

This command (category Trace) will download the trace configuration in order to get the tracing activated on the device runtime system. This is necessary for the first tracing on an application and later always after any changes in the trace configuration or the application program.

Notice that an active trace is indicated by a green background color of the Trace object in the devices tree.

12.31.3 Start / Stop Trace

Symbols:

This command (category Trace) toggles start and stop tracing.

If tracing currently is stopped, the symbol is shown and performing the command means to restart the tracing, that is start to display the variables values as from the actual time.

If tracing currently is running, the symbol is shown and performing the command means to stop the tracing. This means the display will freeze the currently viewed section of the graph(s).

12.31.4 Reset Trigger

Symbol:

This command (category Trace) will reset the trace display after a trigger event has occurred or the trace has been stopped. The tracing (display) will continue with the actual values.

12.31.5 Cursor

Symbol:

As long as this option (category Trace) is activated (Default), the actual position of the mouse-cursor in a coordinate system of the trace window will be always indicated in the status bar. You can move the cursor by clicking in the black triangle of the cursor and scroll it with pressed left mouse key. The same can be done by using **left** resp. **right arrow key**. All keyboard shortcuts listed in Shortcuts for trace. The option will automatically be deactivated if one of the options Mouse scrolling or Mouse zooming is activated.

12.31.6 Mouse scrolling

Symbol:

This command (category Trace) activates the mouse-scrolling mode. If this mode is activated, the mouse-cursor will be displayed for a short moment as a double-arrow and you can scroll the time axis of the currently displayed trace by moving the mouse on the trace window. The same can be done with the **left/right arrow key** while pressing the **ALT key**. For vertical scrolling use the **up** and **down arrow key**. With additionally pressed **Ctrl key** the vertical and horizontal scrolling is done in greater distances.

If the Multi Channel option is enabled, you can scroll the time axis of all diagrams simultaneously with mouse or keypad (**left/right arrow key** while pressing the **ALT key**). For vertical scrolling of the selected diagram use the **up** and **down arrow key** . With additionally pressed **Ctrl key** the vertical and horizontal scrolling is done in greater distances.

If you scroll out of the view range currently available in the trace buffer, "out of data" will be displayed in the trace window.

To de-activate the mouse scrolling mode, activate the Cursor mode.

To get back to the default appearance use command Default appearance.

To zoom the display use command Mouse zooming.

All keyboard shortcuts listed in Shortcuts for trace.

12.31.7 Mouse zooming

Symbol:

This command (category Trace) activates the mouse-zooming mode. In this mode the mouse-cursor will get displayed as and you can draw a rectangle in the trace window in order to re-define the area of the trace curves to be displayed. This area will then be viewed in a way that it fills the trace window.

To de-activate the mouse zooming mode, activate the Cursor mode.

To get back to the default appearance use command Default appearance.

Notice also the possibility to **zoom** the display of the trace window by using a "**scroll mouse**" or by using a keyboard:

- Scrolling the mouse wheel zooms the coordinate system along the x- and y-axes. The same can be done with the numeric keypad **key +** and **key -**.
- Scrolling the mouse wheel while pressing the **Shift key** only zooms along the x-axis. The same can be done with the numpad **key +** and **key -** while pressing the **Shift key**.
- Scrolling the mouse wheel while pressing the **Ctrl key** only zooms along the y-axis. The same can be done with the numpad **key +** and **key -** while pressing the **Ctrl key**.

All keyboard shortcuts listed in Shortcuts for trace.

12.31.8 Default appearance

Symbol:

This command (category Trace) will restore the default settings of the appearance of the graphs, after the settings have been changed for example by zooming. The default settings are defined in the Configuration dialog.

12.31.9 Stretch

Symbol:

With this command (category Trace) you can stretch (zoom) the values of the Sampling Trace that are shown. With repeated stretches that follow one-after-another, the trace section displayed in the window will increasingly shrink in size.

This command is the counterpart to Compress.

12.31.10 Compress

Symbol:

With this command (category Trace) the values shown for the Sampling Trace are compressed; i.e., after this command you can view the progression of the trace variables within a larger time frame. A multiple execution of the command is possible.

This command is the counterpart to Stretch.

12.31.11 Multi Channel

With this command (category Trace) it is able to change the view of the trace diagram. Default view is a diagram with one x- and y-axis and all variables are visualized there. In multi channel view every variable is visualized in its own diagram but with identical x-Axis. Zooming and scrolling commands are affecting the x-axis of all diagrams simultaneously.

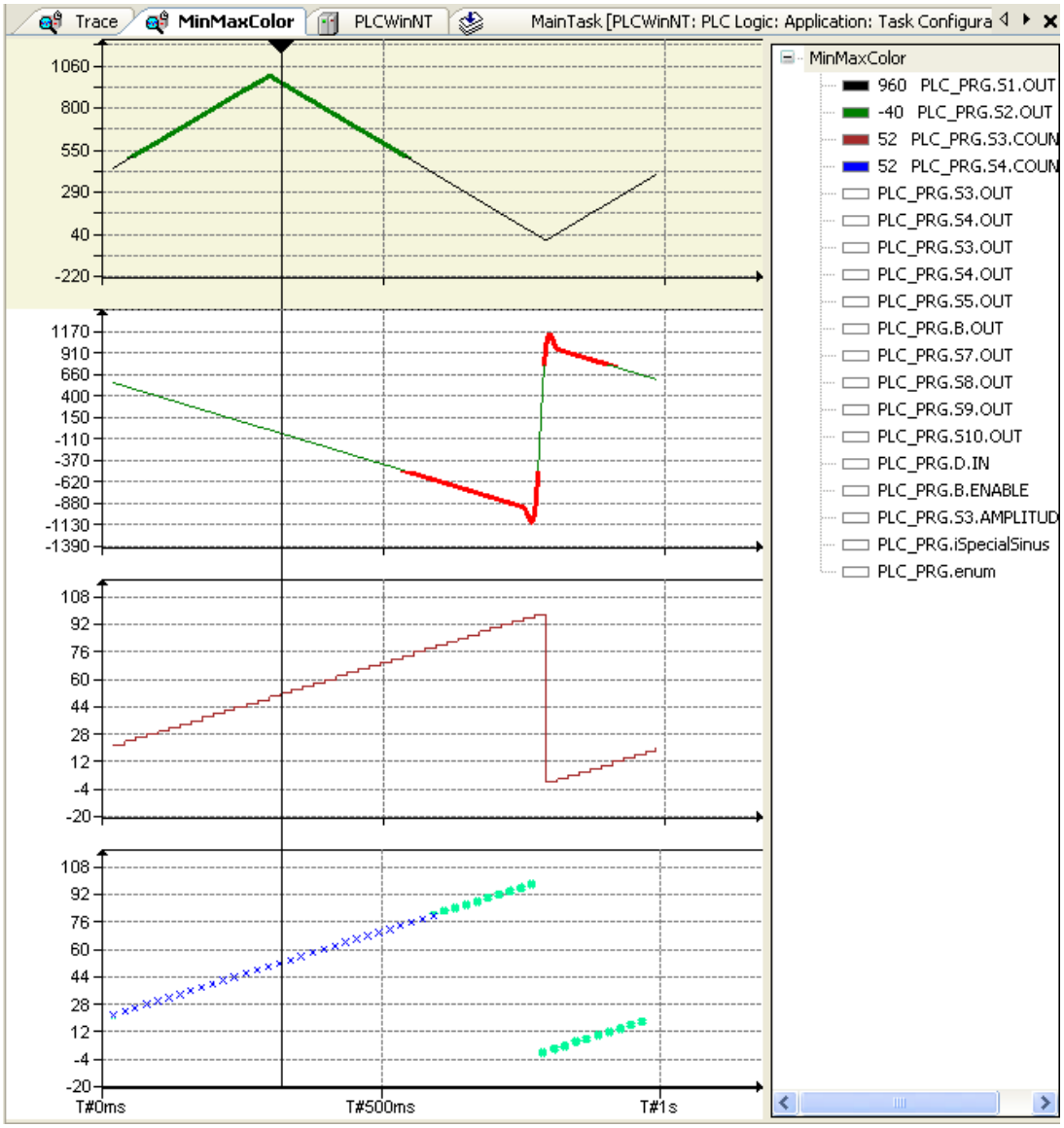


Abb. 272 Multi Cannel

All keyboard shortcuts listed in Shortcuts for trace.

12.31.12 Online List ...

This command (category Trace) opens a dialog, who is listening all traces currently existing on the device. If the trace symbol is placed under an application, the traces running under this application are listed. If the trace symbol is placed under the device, all traces of the applications below are listed and additionally the vendor-specific traces implemented in the device.

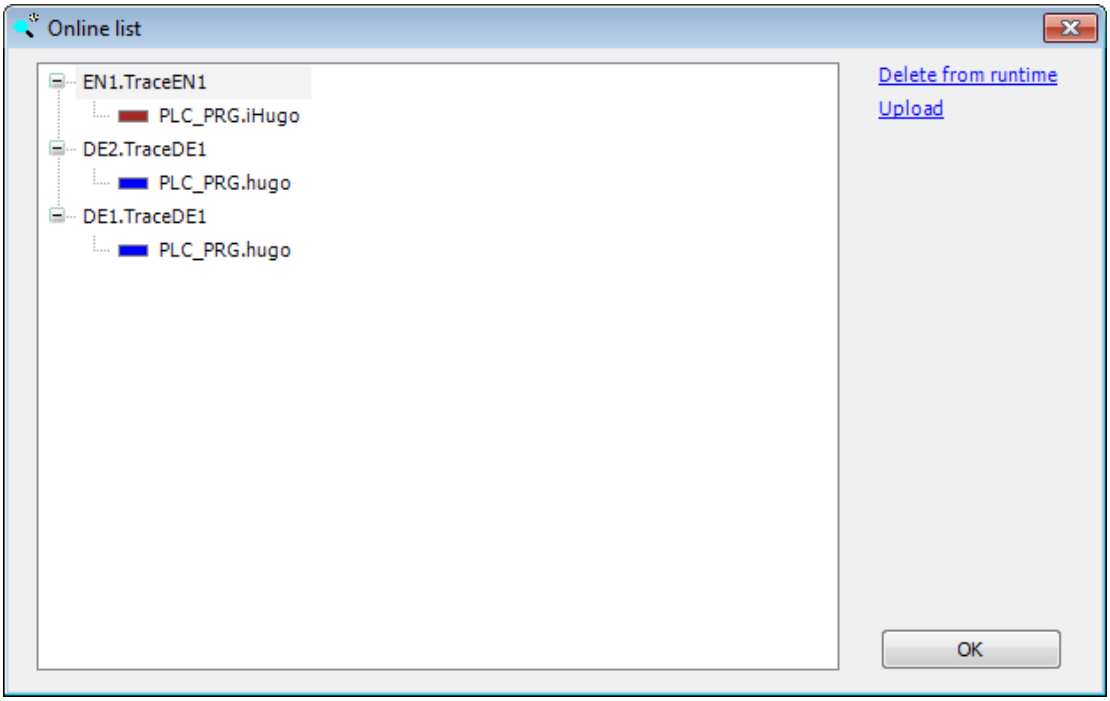


Abb. 273 'Online list' dialog

Delete from runtime: With this command the selected trace is removed.

Upload: With this command the selected trace is loaded up from the runtime system.

This command is only available, when in the runtime system exists the component 'Trace Manager'.

12.31.13 Upload Trace

This command (category Trace) will usually upload all trace records being below the object. If it is positioned below an application the trace records configured in »PLC Designer« programming system will visualized. If it is positioned below an device, all vendor dependent trace records who sampling parameters will visualized, too. The extended name with instance path provides non-ambiguous names (e.g. "Application.Trace.MyRecord").

This command is only available, when in the runtime system exists the component 'Trace Manager'.

12.31.14 Load trace...

This command (category Trace) is used to load a trace configuration which has been stored in a file, back to the editor. The file might have been created by command Save trace....

With the **Load Trace** dialog, which is a standard file-browsing dialog, you are able to look for a saved trace configuration. The default filter is set to "»PLC Designer« Trace files (*.trace)". Closing the dialog with **open** the chosen trace file will be loaded and displayed in the editor window.

12.31.15 Save Trace...

This command (category Trace) is used to save the current trace configuration to a file. The **Save Trace** dialog (standard dialog for saving a file) will be opened where you can define a file name and a location. The default file extension for »PLC Designer« trace files is *.trace.

A trace-file can be reloaded by command Load trace....

12.31.16 Shortcuts for trace

Note:The graphs of the trace diagram are changed in greater distances while pressing the **Ctrl key** additionally to the following keys.

The trace editor and the commands in the trace menu provides following function keys in default mode:

Keyboard shortcuts	Trace diagram action
Left/right arrow key while pressing the **ALT key**	Scroll the time axis of the trace horizontal
Up and **down arrow key** (without pressing the **ALT key**	Scroll the time axis of the trace vertical
Key + and **key -**	Zooms the coordinate system along the x- and y-axes
Key + and **key -** while pressing the **Shift key**	Zooms along the x-axis (horizontal)
Key + and **key -** while pressing the **Ctrl key**	Zooms along the y-axis (vertical)
Left/right arrow key	Scroll the cursor left resp. right

With activated multi channel option the using of the function keys has following effect:

Keyboard shortcuts	Trace diagram action
Left/right arrow key while pressing the **Alt key**	Scroll the time (horizontal) axis of all diagrams simultaneously
Up and **down arrow key** (with or without pressing the **Alt key**	Scroll only the selected diagram vertical
Key + and **key -** (numeric keypad or keyboard)	Zooms the coordinate system along the x- and y-axes of all diagrams simultaneously
Key + and **key -** while pressing the **Shift key** (numeric keypad or keyboard)	Zooms all diagrams along the x-axis (horizontal)
Key + and **key -** while pressing the **Ctrl key** (numeric keypad or keyboard)	Zooms all diagrams along the y-axis (vertical)
Left/right arrow key	Scroll the cursor left resp. right

12.32 Undo/Redo

The command category 'Undo/Redo' provides commands for restoring actions in the projects editors.

12.32.1 Undo/Redo commands - Edit Commands

The 'Edit Commands' plug-in provides for command category 'Undo/Redo' the following commands for restoring previous steps during editing an object in a project:

- Undo
- Redo

By default these commands are part of the 'Edit' menu. If necessary, open the Customize dialog, category Menu, to view or modify the current menu configuration.

12.32.2 Undo

Symbol:

Default Shortcut: <Ctrl>+<Z>

This command (category 'Undo/Redo') undoes the action which was most recently executed in the currently-open editor or view window.

Repeated use undoes all actions back to the time that the window was opened. This applies to all actions in the editors for POUs, data types, visualizations and global variables and in the Object Organizer.

With the Redo command you can restore an action which you have undone before.

12.32.3 Redo

Symbol:

Default Shortcut: <Ctrl>+<Y>

With this command (category 'Undo/Redo') in the currently-opened editor or view window an action can be restored, which has been undone (Undo) before.

You can carry out the Redo command as often as you have previously executed the Undo command.

12.33 User Management

The command category 'User Management' provides commands for setting up user accounts and user groups and for the configuration of the access rights within the project.

12.33.1 User Management commands

The 'User Management UI' plug-in provides commands for command category 'User Management'. These are used for:

- configuration of access rights on the project objects
- logging on or off to/from the project via a defined user account in order to get the access rights which are associated to this account

The configuration of user accounts and groups is done in the Project Settings subdialog 'User Management'.

By default the following commands are part of submenu 'User Management' in the 'Project' menu.

- Logon...
- Logoff
- Permissions....

If necessary, open the Customize dialog, category Menu, to view resp. modify the current menu configuration.

12.33.2 Logon

Symbol:

This command (command category User Management) opens the **Logon** dialog for logging on to a project or library via a defined user account.

Logging on with a certain user account means to log on with those object access rights which are granted to the group which the user belongs to. The configuration of user accounts and groups is done in the Project Settings subdialog 'User Management'.

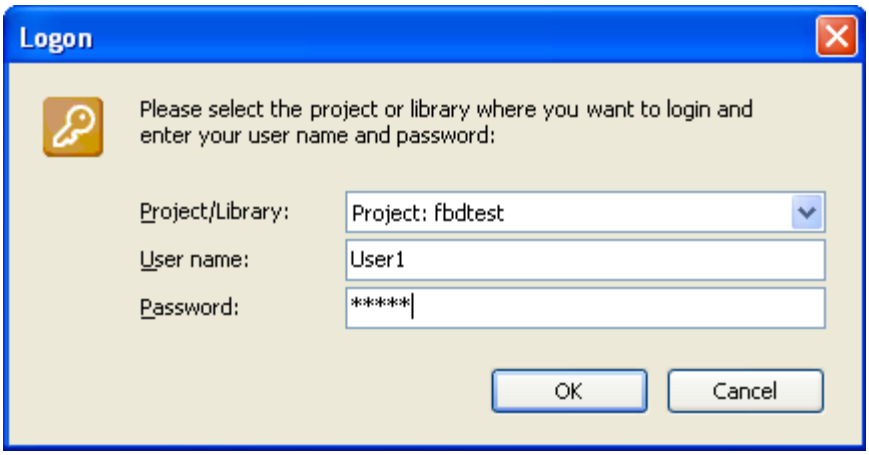


Abb. 274 Logon dialog

To log on select the project or an included library from the selection list in the **Project/Library** field. Enter **User name** and **Password** of a valid user account, noticeing that each project or library has an own user and access rights management. Log on with **OK**.

If already another user is logged on the project, this one will be logged out automatically by the new log-on action.

When you are logged on to a project or library and try to perform an action for which you have no right, automatically the following **Logon dialog** will be opened, giving the possibility to log on with another user account provided with the appropriate rights:

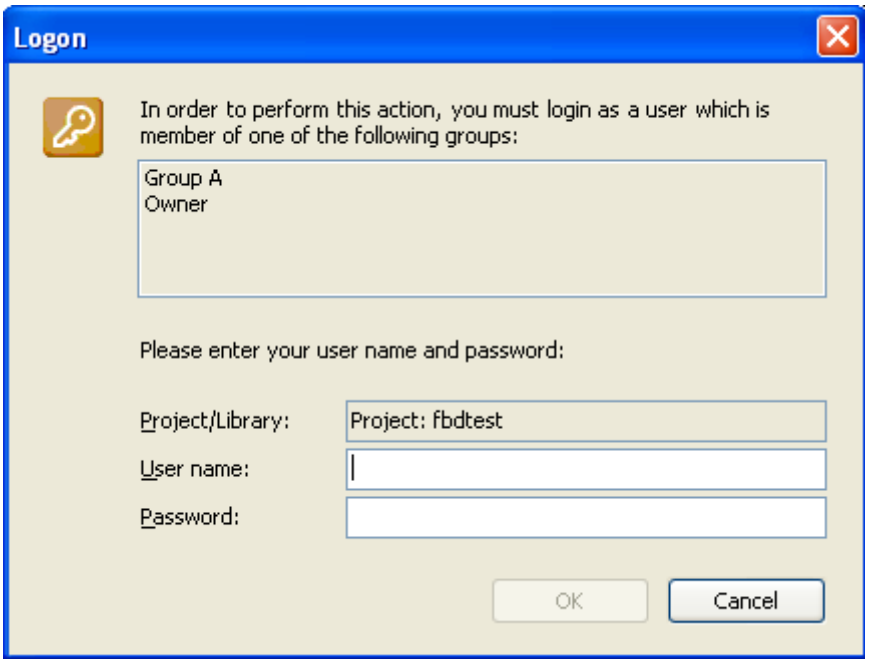


Abb. 275 Logon dialog on a non-permissible action

The upper part of the dialog shows - just for information - all groups which are provided with the necessary rights for the desired action. If you have an user account for one of these groups you now might log-on with the appropriate user name and password and finally perform the desired action.

The **status bar** always displays which user currently is logged on the project.

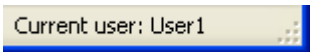


Abb. 276 Example: status line

12.33.3 Logoff

Symbol:

This command (command category User Management) logs off the currently logged-on user from the current project. No dialog or message appears.

If no user had been logged on to the currently opened project or to a referenced library an appropriate message will appear when trying to log off.

If the user currently is logged on to more than one project or referenced library (not necessarily with the same user account) the following **Logoff dialog** will appear when trying to log off:

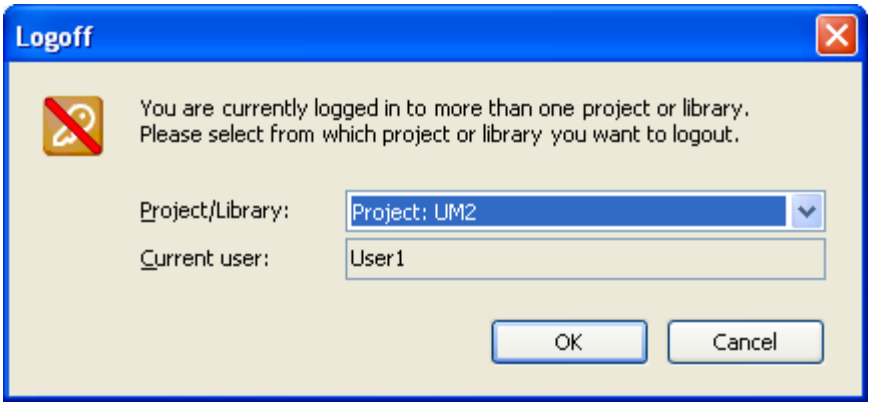


Abb. 277 Logoff dialog

From the **Project/Library** selection list choose those project/library for which you want to log off. The name of the **Current user** is displayed just for information.

The **status bar** always displays which user currently is logged on the project.

12.33.4 Permissions...

This command (command category User Management) opens the **Permissions dialog**, where the rights to work on objects or to perform commands in the current project can be configured. Notice that the rights concerning objects also (same effect) might be configured in the object Properties dialog.

Notice that any changes made in this dialog will be applied immediately.

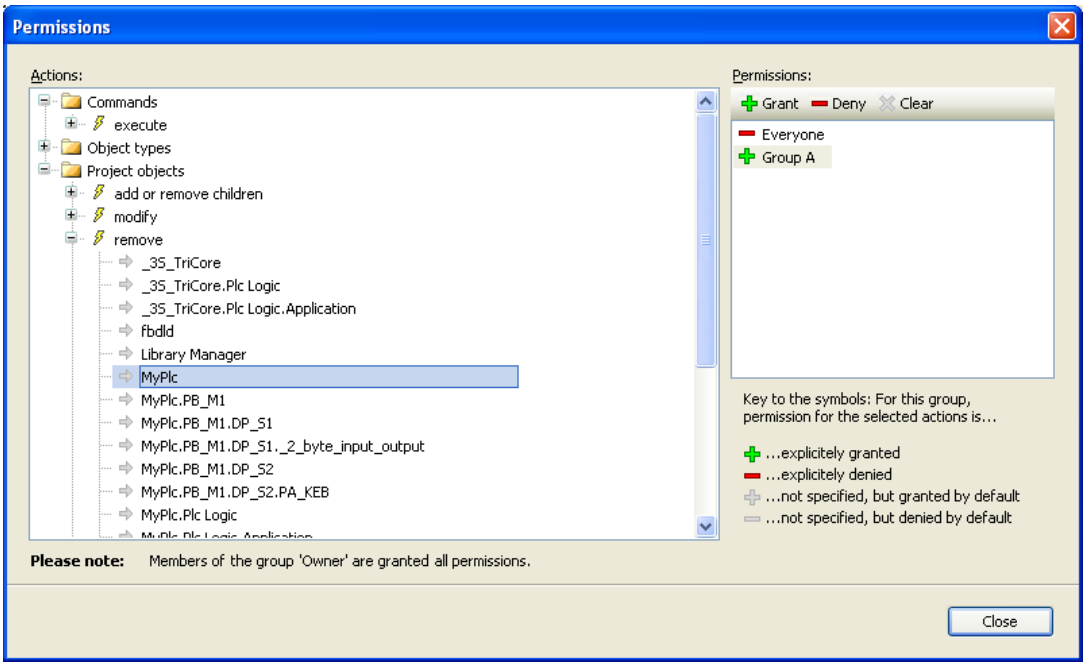


Abb. 278 Permissions dialog

The **Actions** window displays all possible rights, that is all actions which might be performed on any object of the current project. The tree is structured in the following way:

- Top-level see the names of some categories, which have been set up just for the purpose of optical structuring the rights management. They are grouping concering the execution of **Commands**, the configuration of **User accounts and Groups**, the creation of **Object Types**, the viewing, editing, removing and handling of child objects of **Project Objects**.
- Below each category node there are nodes for the particular **actions** which might be performed on the command, user account, group, object type or project object. These nodes also only have optical function.

Possible actions:

execute (execution of a menu command)

create (creating a new object in the current project)

add or remove children (adding or removing of "child" objects to an existing object)

modify (editing an object in an editor)

remove (deleting or cutting an object)

view (viewing an object in an editor)

- Below each action node find the possible "**targets**", that is project objects, of the respective action.

The **Permissions** window provides a list of all currently available user groups (except the "Owner" group) and a toolbar for configuring rights to a group.

Left to each group name one of the following icons indicates the currently assigned permission concerning the target which is currently selected in the Actions window:

- : The action(s) currently selected in the Actions window are **granted** for the group.
- : The action(s) currently selected in the Actions window are **denied** for the group.
- : The right to perform the action(s) which are currently selected in the Actions window, has not been granted explicitly, but is granted by **default**, for example because the corresponding right has been granted to the "father" object. (Example: The group has got the right for object "myplc", thus it by default it also has got it for object "myplc.pb_1".)
  Basically this is the default setting for all rights which not explicitly have been configured.
- : The right to perform the action(s) currently selected in the Actions window has not been designed explicitly, but is denied by **default**, for example in case because the corresponding right has been assigned to the "father" object.
- If currently multiple actions are selected in the Actions window, which do not have unique settings referring to the currently selected group, no icon will be displayed.

To configure the rights for a group select the desired action(s) in the Actions window and the desired group in the Permissions window. Then use the appropriate button in the toolbar of the Permissions window:

Grant: Explicit granting.

Deny: Explicit denying.

Clear: The currently granted right for the action(s) currently selected in the Actions window will be deleted, that is set back to the default.

12.34 Visual Commands

The command category 'Visual Commands' provides commands for editing a visualization, that is for handling the visualization elements.

12.34.1 Visual Commands

The commands listed below are provided with the 'Visual Editor' plug-in for command category 'Visual Commands'. They serve to edit a visualization object in the Visualization Editor.

By default the commands are part of the Visualization menu and usually also of the context menu. If necessary open the Customize dialog to view resp. modify the menu configuration. Notice category 'Visual Commands'.

The commands:

- Add Visualelement
- Frame selection
- Interface Editor
- Hotkeys Configuration
- Element List
- Background
- Selection:
- Select all
- Select none
- Group
- Ungroup

Alignment:

- Align bottom
- Align horizontal center
- Align left
- Align right
- Align top
- Align vertical center

Order:

- Bring one to front
- Bring to front
- Send one to back
- Send to back
- Activate keyboard usage

12.34.2 Add Visualelement

This command (category Visual Commands) by default is available in the Visualization menu and in the context menu. It is used to insert an element into the current visualization and thus corresponds to inserting via drag & drop from the visualization Toolbox.

The command opens a submenu where from a list of all currently available visualization elements one can be chosen. The selection matches that of the visualization Toolbox.

12.34.3 Select all

This command (category Visual Commands) is used to select all visualization elements of the current visualization at once. Notice also command Select none.

12.34.4 Select none

This command (category Visual Commands) is used to cancel at once any selection in the current visualization.

12.34.5 Group

Symbol:

This command (category Visual Commands) groups the currently selected visualization elements and displays the group as a single selected object. For multiple selection keep the <Shift> key pressed while clicking on the desired elements. Alternatively you might perform a mouse-click outside of an element in the editor window and - while keeping the mouse-button pressed - draw a rectangle around the desired elements.

For resolving the group use command Ungroup.

The following picture shows the grouping (from left to right) and ungrouping (from right to left) of two rectangle elements:

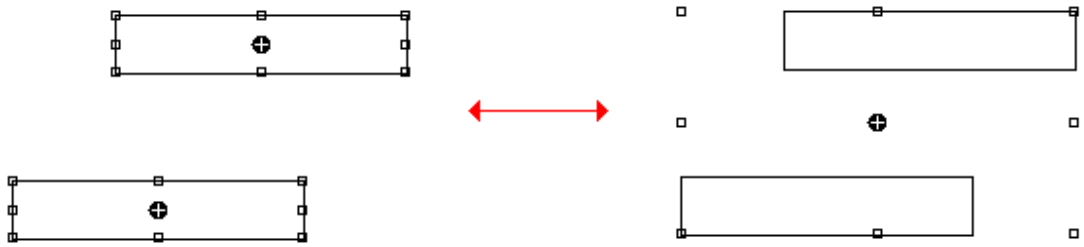


Abb. 279 Grouping and ungrouping

12.34.6 Ungroup

Symbol:

This command (category Visual Commands) resolves a selected group of elements. The particular elements will be displayed each selected. See also: Group.

12.34.7 Frame selection

This command (category Visual Commands) is used to configure the content of a **Frame** element.

A Frame element is used to define a subarea of a visualization which includes one or several other visualizations. In online mode it can be toggled between the display of these particular visualizations.

Basically the first visualization in the list of visualizations assigned to the frame will be displayed. But by an input on another - appropriately configured (Input properties, Switch Framevisualization) - visualization element the user might effect that one of the other assigned visualizations will be displayed in the frame. Thus switching between different displays within a visualization is possible, in contrast to »PLC Designer« V2.3, where one always had to jump completely between the particular visualizations.

The visualizations included in a frame are references, that is instances of the original visualizations, and placeholders defined in the original visualization objects can be replaced by locally applicable values. See in this context the following page: Frames,_References,_Interface (Placeholder)

The selection of the visualizations in a frame is done in the **Frame Configuration** dialog, which is opened by the 'Frame selection' command:

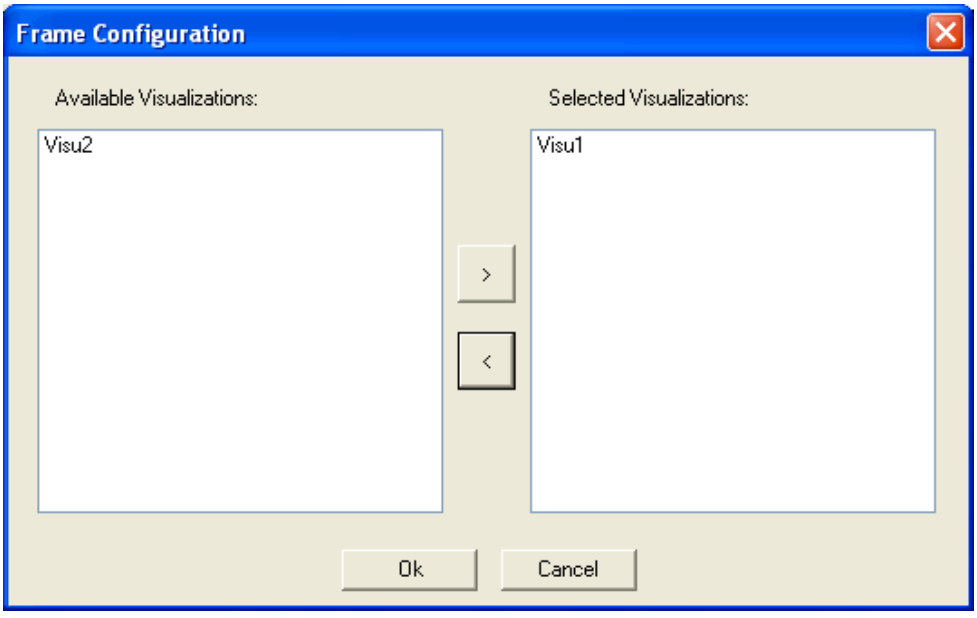


Abb. 280 Frame Configuration dialog

On the left side see the **Available Visualizations** of the project. Select those which should be included in the frame and via the arrow-buttons bring them to the **Selected Visualizations** list on the right side resp. remove them from there.

It is recommended to assign only those visualizations to a frame, which are managed in the global pool. Otherwise problems might occur, if at a later time any device or application objects will be renamed and due to this the path of the assigned visualizations will not be valid any longer.

The order of the selected visualizations from top to down determines the automatically generated implicit index numbers for the visualizations. The uppermost gets index "0", the following ones "1", "2" etc.. The index numbers are required for the configuration of the switch function (Switch Framevisualization) of another element. Initially the visualization with index "0" will be displayed.

Example:

Via a menu bar you want to determine which of several visualizations currently should be displayed in online mode. For this purpose you can create a menu bar in a visualization consisting of several elements, each calling a certain visualization. Within the same visualization add a frame element and in this frame include - via the Frame selection command - the visualizations to be toggled by the menu bar elements. So - in contrast to»PLC Designer« V2.3 - the menu bar just has to be included in one visualization to reach all visualizations to be toggled.

12.34.8 Interface Editor

Symbol:

Default Shortcut: <Alt> + <F6>

This command (category Visual Commands) opens the Interface Editor for defining "placeholder" variables in a visualization which is intended to get **referenced** in a frame element in another visualization. The command is available when the visualization is opened in the Visualization editor.

12.34.9 Hotkeys Configuration

Symbol:

This command (category Visual Commands) opens the editor for the hotkeys configuration for the current visualization. It will be displayed in a tabbed view in the upper part of the visualization editor.

12.34.10 Element List

Symbol:

This command (category Visual Commands) opens the element list editor of the current visualization. It will be displayed in a tabbed view in the upper part of the visualization editor.

12.34.11 Background

Symbol:

This command (category Visual Commands) allows to define a image filling the background of the complete visualization. The dialog 'Background' will be opened, where you can define an image file and / or an background color. For this purpose activate the respective option.

Bitmap: To define a background image, you must enter the path of an image file, which is available in an image pool, in the Bitmap field: Enter the name of the image pool and the ID string separated by a dot: <pool name>.<image ID> (for example "Images_1.drive_icon>", "Images_1.43").

Color: To define a background Color, choose one from the available color selection list.

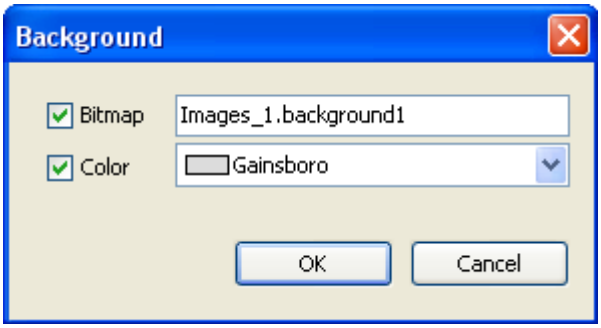


Abb. 281 Background dialog

12.34.12 Order

Bring one to front

Symbol:

This command (category Visual Commands) places the selected element one layer higher, that means nearer to the foreground of the visualization.

Bring to front

Symbol:

This command (category Visual Commands) places the selected element in the absolute foreground of the visualization.

Send one to back

Symbol:

This command (category Visual Commands) places the selected element in the absolute background of the visualization.

Send to back

Symbol:

This command (category Visual Commands) places the selected element one layer deeper, that means nearer to the background of the visualization.

12.34.13 Alignment

Align left

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the left line of that of those elements which is at the left-most position.

Align top

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the upper line of that of those elements which is at the upmost position.

Align right

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the right line of that of those elements which is at the right- most position.

Align bottom

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the bottom line of that of those elements which is at the most bottom position.

Align horizontal center

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the horizontal center of all selected elements.

Align vertical center

Symbol:

Using this command (category Visual Commands) the currently selected visualization elements will be aligned to the horizontal center of all selected elements

12.34.14 Activate keyboard usage

Symbol:

This command (category Visual Commands) is available in the menu bar for an integrated visualization. It activates resp. deactivates the keyboard usage in the online mode of the visualization.

When the keyboard operation is activated, any inputs on elements and the selection of elements can be done by using certain keys. In this case other commands given via key shortcuts will not be executed as long as the visualization editor is active and in online mode.

12.35 Views

Windows of the "view" type can be handled by the commands which belong to command category "view". Notice additionally the special command category 'Message View'.

12.35.1 Cross Reference View

View commands - Cross Reference View

The 'Cross Reference View' plug-in provides the command Crossreference View (command category 'View') which is used to open the view for the references of a variable within the project.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

Cross Reference View

This command (category View) opens a view window where you can get listed the cross references of a project variable, that is the locations where the variable is used within the project or just within the scope of the same POU. The view is titled **Cross Reference List**.

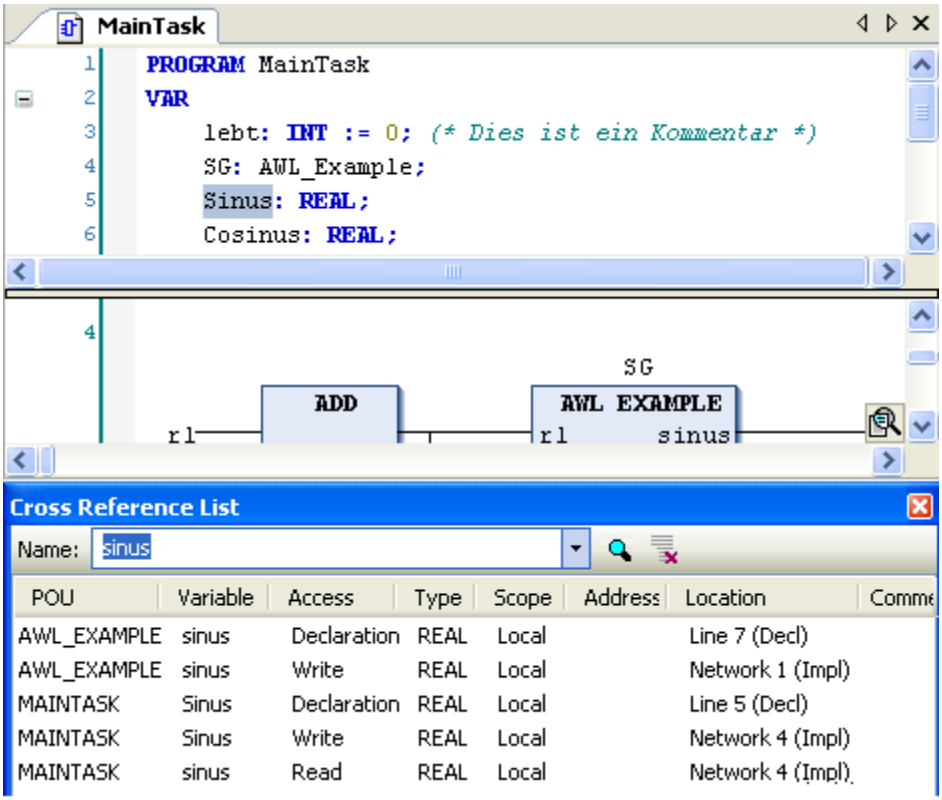


Abb. 282 Cross Reference List view, example

If **all cross references** within the project should be listed: Type in the identifier string manually or copy it via copy&paste from any editor window to the **Name** field and then confirm with <Enter> or use button .

If **only the cross references within the same POU** should be listed: Either select the identifier string within the editor window of that POU and draw it with the mouse into the Cross Reference List view, or activate option Automatically list selection in cross reference view and place the cursor within the string in the editor window resp. select that string. In this case the string will be automatically entered in the **Name** field and also the list of references will be displayed automatically.

If a valid identifier has been entered, the found locations will be listed in the form of a table showing the following information on the variable:

POU	Name of the POU where the variable is used.
Variable	Variable name resp. POU name of the reference.
Access	How the variable is used resp. accessed at this location: Declaration / Read / Write / Call
Type	Data type
Scope	Scope of the variable: Global / Local
Address	IEC address if defined
Location	Position of the variable reference within the editor (for example line, network)
Comment	Comment attached at the declaration of the variable.

The list can be **sorted alphabetically** along one of the columns. By a mouse-click on the column title field the sorting will be done in ascending or descending order (toggles with each further mouse-click).

A double-click in a line of the cross reference list the **opens the corresponding POU** and the respective location of the variable will be highlighted there. This corresponds to using button ("Show location") when a list entry is selected.

Button ("**Show previous location**", Shortcut: <Shift> + <F4>), will jump to the previous entry in the cross reference list.

Button ("**Show next location**", Shortcut: <F4>) will jump to the next entry in the cross reference list.

Button ("**Go to definition**", Shortcut: <F2>) will jump to the location, where the respective variable is declared. The corresponding declaration editor will be opened and the variable will be highlighted there.

Button corresponds to command Browse cross references and will effect that the current references **list is generated to the message window**. This might be useful if you want to keep the list available although the automatic display is activated and thus the list in the cross references view might change.

12.35.2 Breakpoints

View commands - Breakpoints

The 'Breakpoints' plug-in provides for command category 'View' the below listed commands for handling views and dialogs for the breakpoint and callstack functionality.

By default these commands are part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

The commands:

Breakpoints dialog

Call Stack

Breakpoints dialog

Symbol:

This command (category 'View') opens the **Breakpoints** dialog which provides an overview on all breakpoints currently set in the project. The current breakpoint parameters are displayed and can be modified. Also breakpoints can be added, removed, enabled or disabled.

The breakpoint parameters basically are those which have been set when creating a breakpoint by the 'Toggle Breakpoint" command (default condition parameters are assigned) or via the 'New Breakpoint' (possibility to define certain conditions).

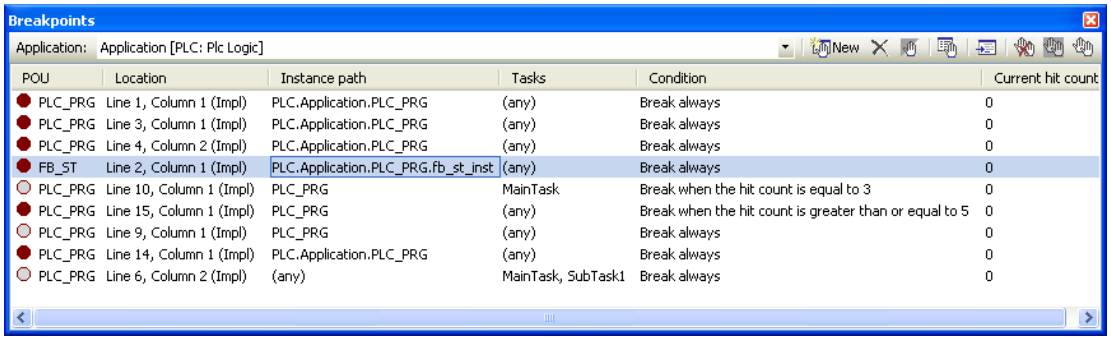


Abb. 283 Breakpoints dialog

Application	Name of currently active application	example: "App1 [PLC:Plc Logic]"
POU	Name of POU containing this breakpoint	example: "PLC_PRG"
Location	Breakpoint position within POU: line+column numbers (text editors) or network or element numbers (graphic editors); "(Impl)" in case of function blocks indicates that the breakpoint is in the implementation part of the function block	example: "Line 2, Column 1 (Impl)"

Instance path	Complete object path of the breakpoint position	example: "PLC.Application.PLC_PRG.FBinst1"
Tasks	Tasks during the run of which the breakpoints should be noticed: "(any)" in case of no restriction (default), resp. particular task name(s) .	example: "MainTask, SubTask1"
Condition	Definition of when (number of hits) the breakpoint should cause a break in processing; (default: 1).	example: "Break when the hit count is equal to 3"
Current hit count	Indicates how often the breakpoint has been run through (hit) up to now.	example: "3"

The following functions are available as **buttons** in the upper right part of the dialog for editing the current breakpoints parameters resp. for removing or adding breakpoints:

	New breakpoint	Opens the 'New breakpoint' dialog for defining a new breakpoint. See description of corresponding command 'New breakpoint'.
	Clear breakpoint	Removes the breakpoint; do not mix up with disabling !
	Enable/disable breakpoint	Toggles the breakpoint between 'enabled' ● and 'disabled' ○ status; in case of 'disabling' the breakpoint will not be removed from the list, but can be re-enabled.
	Properties	Dialog 'Breakpoint Properties' will be opened, where you can modify the breakpoint parameters. The dialog matches the 'New Breakpoint' dialog.
	Go to source position	The 'Select Online State' dialog will open from where you can get to the source position of the breakpoint.
	Clear all breakpoints	Removes all breakpoints. The list will be empty. Do not mix up with disabling!
	Enable all breakpoints	Enables (●)all currently disabled breakpoints .
	Disable all breakpoints	Disables (○) all currently enabled breakpoints. The breakpoints remain in the list and can be re-enabled.

Call Stack

Symbol:

This command (category 'View') opens the **Call Stack** window. When you are stepping through a program in online mode, always the currently reached step position will be indicated there with its complete call path.

The Call Stack window below the title bar always displays the name of the currently concerned **Application** and the name of the **Task** controlling the currently reached POU.

The call stack is displayed as a list of positions, each described by **POU** name, **Location** and - in case of instances - with the **Instance Path**. Depending on the editor, the location is described by the line and column numbers (text editor) or by the network or element numbers (graphic editors).

The first line in this list, indicated with a yellow arrow, describes the current step position. If this position is within a POU which is called by another POU, the position of the call will be described in the next line. If this POU again is called by another POU, the call position follows in the third line, and so on.

The call stack view is also available in offline mode resp. during normal online run. In this case The position which was last viewed during an online stepping session will be still displayed, but in greyed letters.

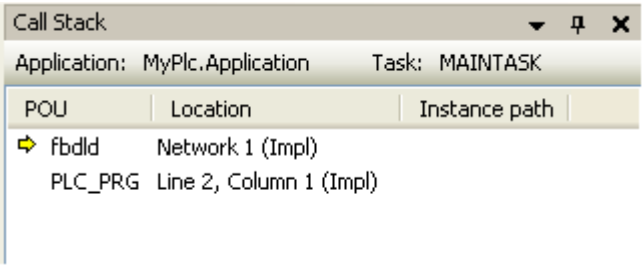


Abb. 284 Call Stack view, Current position "fbdld", called by "PLC_PRG"

12.35.3 Default Navigators

View commands - Default Navigators

The 'Default Navigators' plug-in provides the following commands for command category 'View' which will be visible as windows in the user interface:

- POUs
- Devices

By default each view in the upper right corner has a button for opening a menu with the following commands and options:

Open in editor: Opens the object in the appropriate editor.

Find object: Dialog 'Find Object' opens, where you can search the objects tree for a POU name. At **Find what:** enter the search string (may be a single character) and activate the desired options:

- Match whole name: The search will provide all object names which exactly match the sequence of characters of the search string.
- Match case: The search will provide all object names which match the search string exactly concerning the use of upper and lower letters.
- Match prefix: The search will provide all object names starting with the search string.
- Match substring: The search will provide all object names containing the search string.

The found objects will be displayed with **Name** and **Path** in the **Result** window.

Sort by type, **Sort by name**: The objects get sorted by type or name, and alphabetically.

Sort ascending, Sort descending: The above defined sorting is arranged in ascending or descending order.

POUs View

Symbol:

In the POUs view window all programming units of the current project (PLC program) are organized and can be instantiated for the use in an specific application.

Devices View

Symbol:

In the Devices view window all devices needed for the project are configured and the applications are defined appropriately.

For a description of the toolbar see the main help page on the Default Navigators commands.

12.35.4 PLC Designer Frame

View commands - PLC Designer Frame

The 'Frame' plug-in provides for command category 'View' commands (see below) which serve to define certain object properties resp. to switch on/off the full screen mode.

By default these commands are part of the 'View'. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

The commands:

- Full Screen
- Properties...
- Start Page

Properties...

Symbol:

This command (category 'View') opens dialog 'Properties <objectname>'. The properties of the currently selected object in the POUs or Devices view will be displayed on various tabs, the availability of which depends on the type of object.

The following dialogs are possible:

Common (information on object)

Provides information on the object:

Full name: Object name as used in the POUs or Devices view

Object type: Type of the object (for example POU, Application, Interface etc.)

Open with: Type of the editor which is used to edit the object

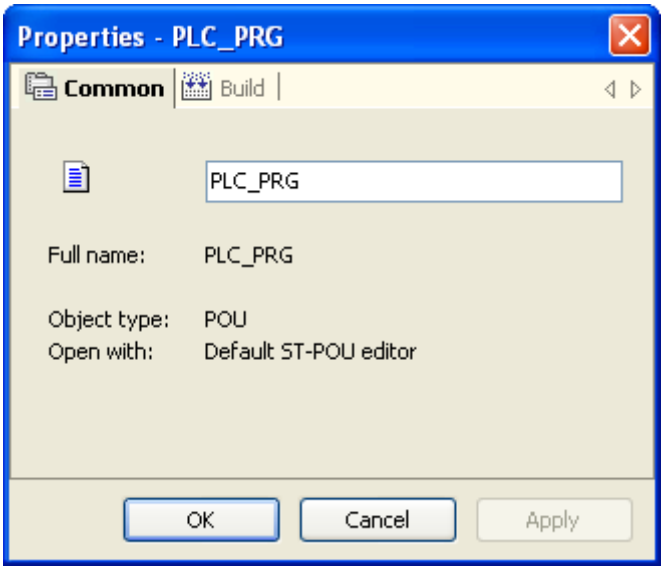


Abb. 285 Properties dialog, Category Common

Boot application

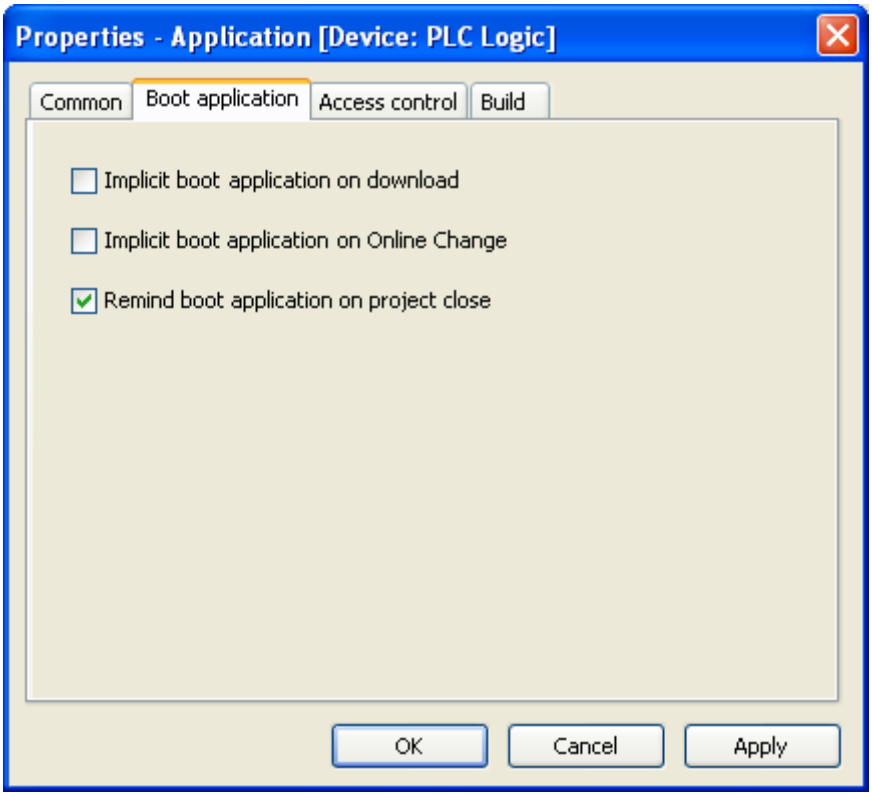


Abb. 286 Properties dialog, Category Boot application

It depends on the device whether these settings are available:

Implicit boot application on download: If activated, at a download of the project automatically a boot application will be created.

Implicit boot application on Online Change: If activated, at an online change automatically a boot application will be created.

Remind boot application on project close: If activated, when going to close the project you will be asked whether the boot application should be updated/created.

Bitmap

Here you can associate a bitmap to the current object resp. remove a currently associated bitmap. The bitmap will be used in the graphical display of the object in the Library Manager and in the toolbox of the FBD/LD/IL editor. For specifying the transparency of the bitmap you can activate option **Render pixels of this color transparently** and via a mouse-click on the color rectangle select the color to be made transparent.

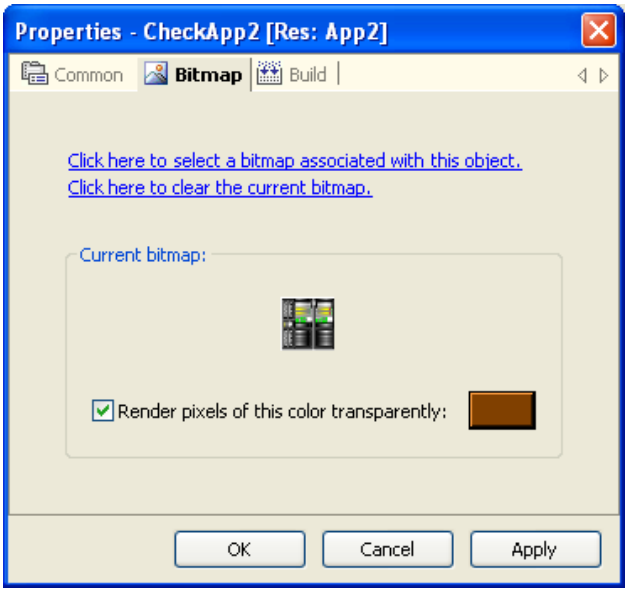


Abb. 287 Properties dialog, Category Bitmap

Dynamic Memory Settings (Application object)

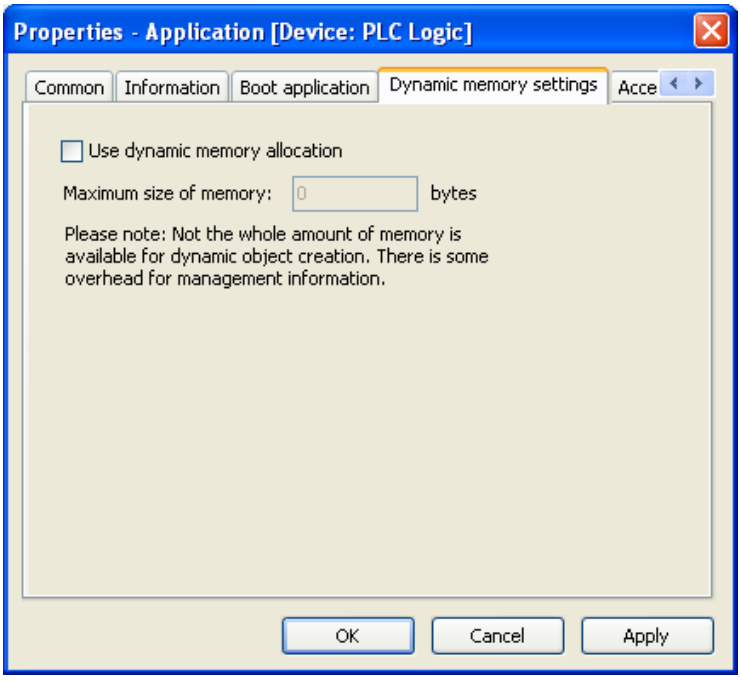


Abb. 288 Properties dialog, Dynamic Memory Settings

Use dynamic memory allocation: Activate this option if memory should be allocated dynamically for the application. In this case enter the desired **Maximum size of memory (bytes)**. Regard, that not the complete memory will be available for dynamic object creation, but that a part of it always will be used for management information.

The current settings will be transmitted to the PLC at every download or online change.

External file

Here you find the settings you have defined when adding an external file via the 'Add Object' dialog. You can modify these settings here. Not yet implemented: Additionally there is an option Update the embedded file which can be activated to force an immediate update of the external file. This option setting will not be saved after having closed the dialog with OK. To get the current settings effective without closing the properties dialog, use button **Apply**.

Visualization (Use visualization as Dialog or Numpad/Keypad, Display size, Noticeing background image)

For the currently selected visualization object the following properties are displayed and can be modified:

Use visualization as:

Visualization: Standard visualization object.

Dialog: The visualization represents a dialog and will be automatically available in other visualizations insofar as it can be specified there in the input configuration in category 'Open Dialog' of a visualization element.

Numpad/Keypad: The visualization represents a numpad or keypad and will be automatically available in other visualizations insofar as it can be specified there in the input configuration in category 'Write Variable' of a visualization element.

Visualization size: Current **Width** and **Height** of the visualization object (number of pixels)

Use automatic detected visualization size: The size of the visualization is calculated which exactly includes all currently defined elements and the background image. If the option **Include background image** however is deactivated, the background image - if exceeding the visualization area defined by the elements - will be cut.

Use specified visualization size: The visualization size is defined by the values specified above (Width, Height), regardless whether all visualization elements and the background image will fit into the defined area.

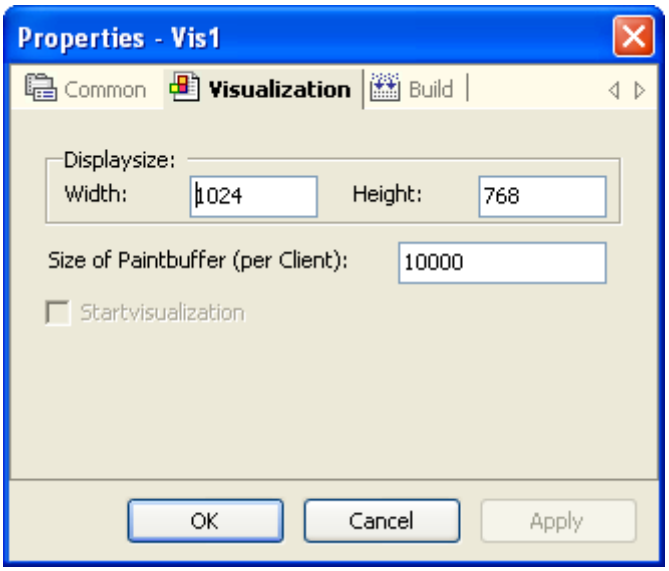


Abb. 289 Properties dialog, Category Visualization

Notice, that in the Visualization Manager you can specify, whether the visualization area as defined here, should be automatically fit to the screen size of the visualization client or not.

Access Control (User Group rights)

This dialog will only be available if the User Management plug-ins are part of the currently used profile. It allows to configure the access rights on the current object for the available user groups. This corresponds to the configuration via the Permissions dialog which is available in the User Management menu.

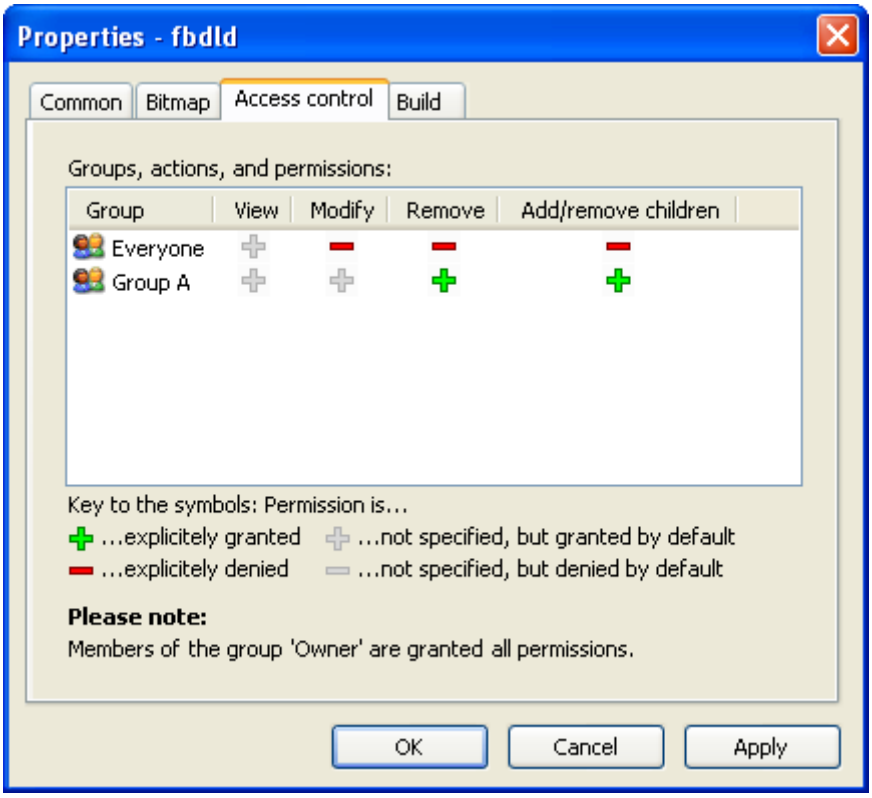


Abb. 290 Properties dialog, Category Access Control

To edit the right for a certain action and group select the respective field in the table, perform a mouse-click or use <spacebar> to open the selection list and from there choose the desired right.

For a description on possible actions, rights and the symbols please see the help page on the Permissions dialog.

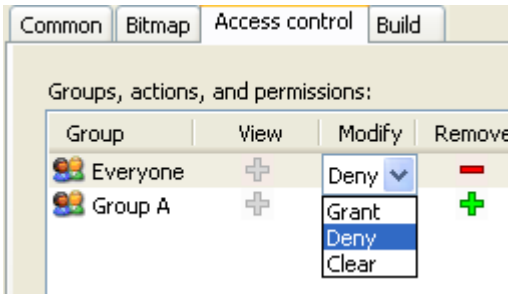


Abb. 291 Example: Selection list of rights for action 'Modify' for group 'Everyone'

Source Control (Change binding, Checkout status)

This dialog will only be available if the project is "under Source Control", that is if the project objects are bound to an external data base via the ENI.

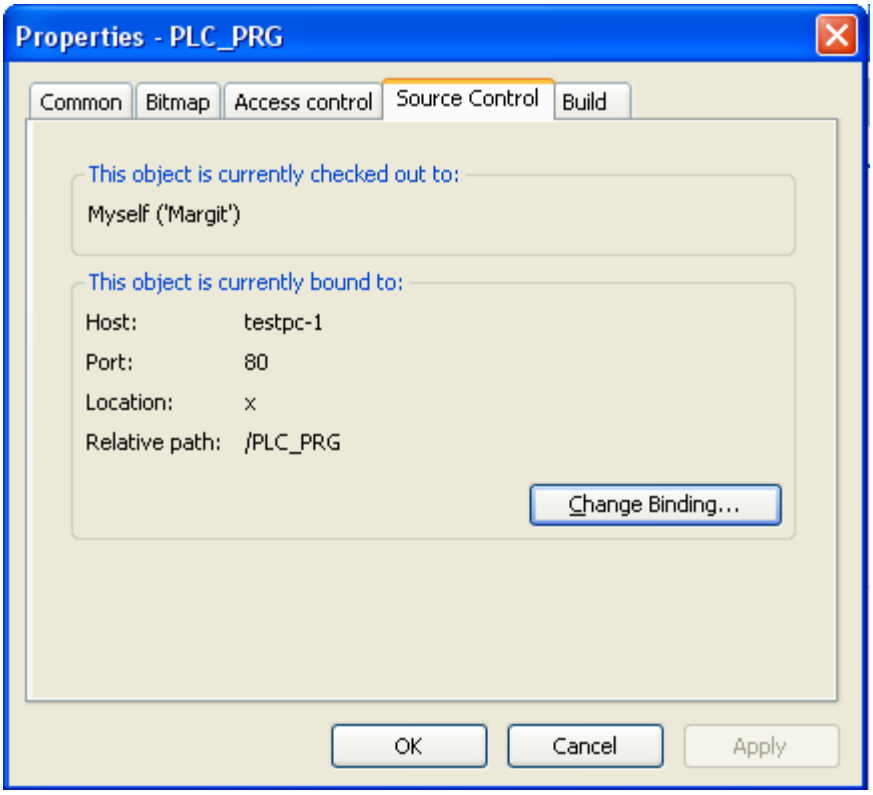


Abb. 292 Properties dialog, Category Source Control

This object is currently checked out to: Here you get information on whether the object is currently checked out and by which user.

The object is currently bound to: Here you get information on the data base location to which the object is bound:

Host: Name of the computer where the data base is located

Port: Port number for the connection to the data base system

Location: Data base folder containing the object

Relative path: Path of the object in the data base, relative to the above specified location

Change Binding: This button opens the 'Change Source Control Binding' dialog, where you can modify the data base location to which the object is bound:

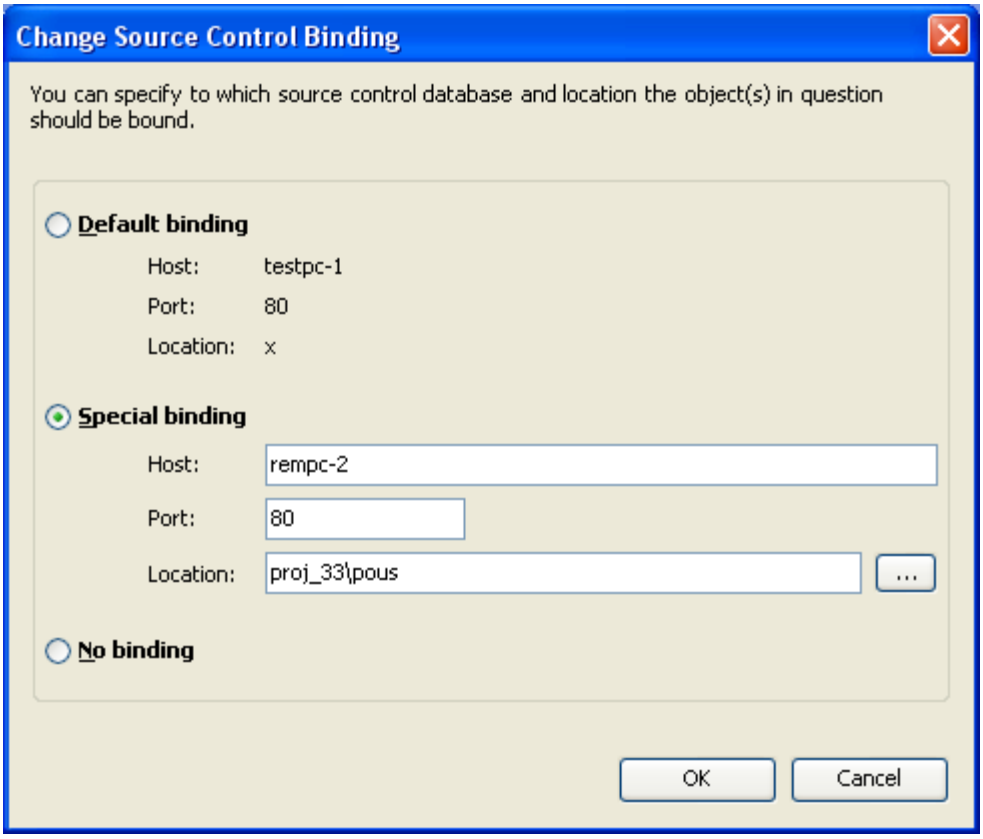


Abb. 293 Properties dialog, Change Source Control Binding

Default binding: This option is by default activated. The object is bound to the data base location which has been specified when setting the project under source control in the Change Source Control dialog. The respective Host name, Port and data base Location are shown.

Special binding: In order to bind the object to another than the default location (see above, Default binding), activate this option and enter the appropriate Host name, Port and Location. The ... button might be used to get a dialog for browsing for the desired location.

No binding: Activating this option means that the object is not stored in the data base. It is just stored locally with the project (corresponding to object category "Local" used in »PLC Designer«V2.3)

Build (Exclude from build, Enable system call, External implementation, Link Always, Compiler defines)

Concerning the compilation (build) the following options can be activated:

Exclude from build: The object will not be noticed during the next 'Build' run.

External implementation (Late link in the runtime system): No code is created for this object during a compilation of the project. The object will be linked when running the project on a target, if it is available there, for example via a library.

Enable system call: Background: In contrast to »PLC Designer«V2.3 now the ADR-Operator can be used with function names, program names, function block names and method names, thus replacing the INSTANCE_OF operator. See in this context 'Function pointers'. BUT there is no possibility to call a function pointer within»PLC Designer«! In order to enable a system call (runtime system) you must activate the current option for the function object.

Link Always: The object is marked for the compiler so that it is always included into the compile information. As a result, objects will always be compiled and downloaded to the PLC. This option goes relevant, when the object is located below an application or is referenced using libraries inserted below an application. The selectable variables for the symbol configuration use the compile information as basis too.

Compiler defines: Here you can enter "defines" (see {define} instruction) and conditions for the compilation of this object. The expression *expr* used in those pragmas can be entered here, several entries can be entered in a comma-separated list.

For example it might be useful to make dependent the compilation of an application on the value of a certain variable. (See an example In the following figure: "hallo" is defined and if variable test has value "2" the application will be compiled.)

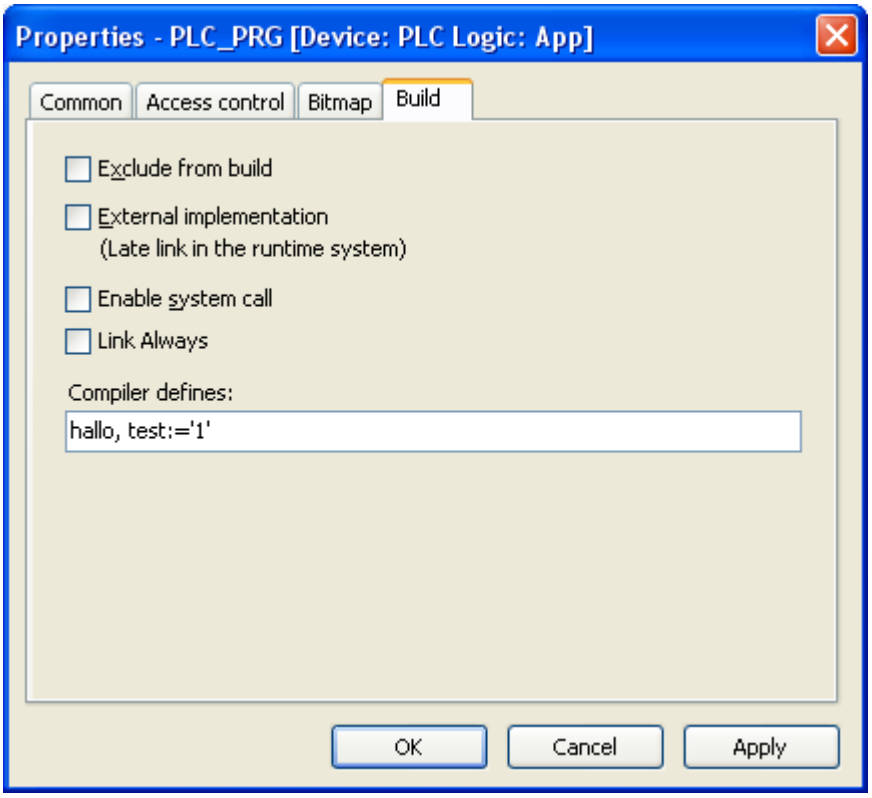


Abb. 294 Properties dialog, Category Build

Network Properties (for Global Variables List)

If the network functionality is supported by the current device, then the Network Properties of a GVL (Global Variable List) object can be viewed and edited in the Properties dialog. Specifying network properties for a GVL means to make available the included variables as network variables. A GVL is always to be defined by the "sender" of the network variables. The receiver must have a corresponding GNVL list. Regard the general description on how to use network variables.

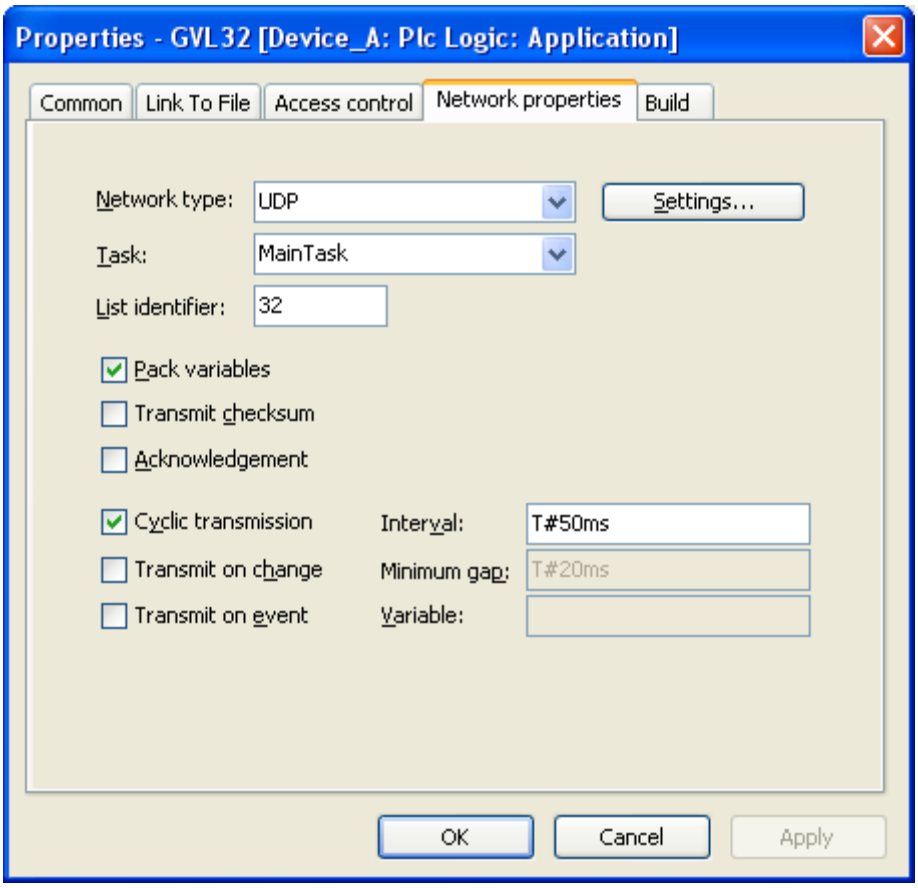


Abb. 295 Properties dialog, Category Network Properties

Network type: Choose the desired type from the target-dependent selection list. For example „CAN" as an abbreviation for a CAN network, or „UDP" for an UDP transmission system might be selectable.

Task: From the selection list choose the task of the current application which should control the sending of the variables. The variables always will be sent at the end of a task cycle.

List identifier: Number (ID) of the first list to be sent (default = 1). Further lists will be numbered in ascending order.

Settings: Protocol-specific settings; the permissible entries depend on the corresponding network library.

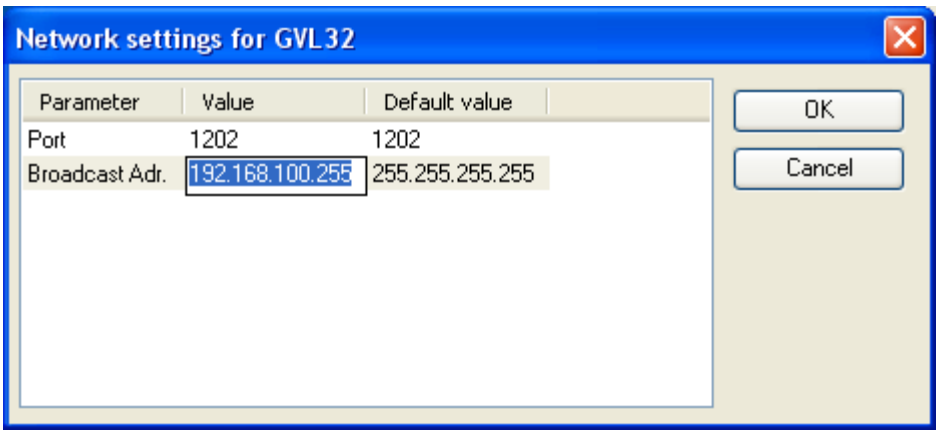


Abb. 296 Network Settings dialog

For "UDP" networks the following **Parameter**s are to be defined:

Port: Number of the port to be used for data exchange with the other network participants; the **Default value** is "1202"; the current value always can be modified in the **Value** field (select the field an press <space> to open the edit frame); Make sure that the other nodes in the network define the same port! If you have more than one UDP connection defined in the project then the port number will be automatically modified in all configuration sets according to the input you make here.

Broadcast address: The **Default value** is "255 . 255 . 255 . 255", which means, that the data exchange will be done with all participants in the network. The current value always can be modified in the **Value** field (select the field an press <space> to open the edit frame) where you can enter the address resp. the address range of a sub-network (for example enter "197 . 200 . 100 . 255", if you want to communicate with all nodes which have IP-addresses "197 . 200 . 100 . x").

Notice for Win32 systems, that the Broadcast addresses must match the subnet mask of the TCP/IP configuration of the PC !

The following options can be activated or deactivated for configuring the transmission behavior of the variables:

Pack variables: For the transfer the variables are bundled in packets (telegrams) whose size depends on the network. If the option is deactivated, one packet per variable will be set up.

Transmit checksum: A checksum will be added to each packet of variables. The checksum will be checked by the receiver to make sure that the variable definitions of the sender and the receiver are identical. A packet with a non-matching checksum will not be accepted.

Cyclic transmission: Variables are sent within the **Interval** specified after Interval. (time notation for example T#70ms).

Transmit on change: Variables will be sent only if their values have changed; the **Minimum gap** can define a minimum time lapse between transfers.

Transmit on event: The variables will be sent as soon as the specified **Variable** gets TRUE.

Notice that the network variables will be automatically sent at each **bootup**. This means that the actual variable values will be transmitted even if none of the other defined transmission triggers (change or event) would force this at this moment.

Network Settings (for Global Network Variables List)

If the network functionality is supported by the current device, then the Network Settings for a GNVL (Global Network Variable List) object can be viewed and edited in the Properties dialog. Basically these are the settings which have been defined when adding the NVL object via the 'Add Object' dialog. (Regard the general description on how to use network variables).

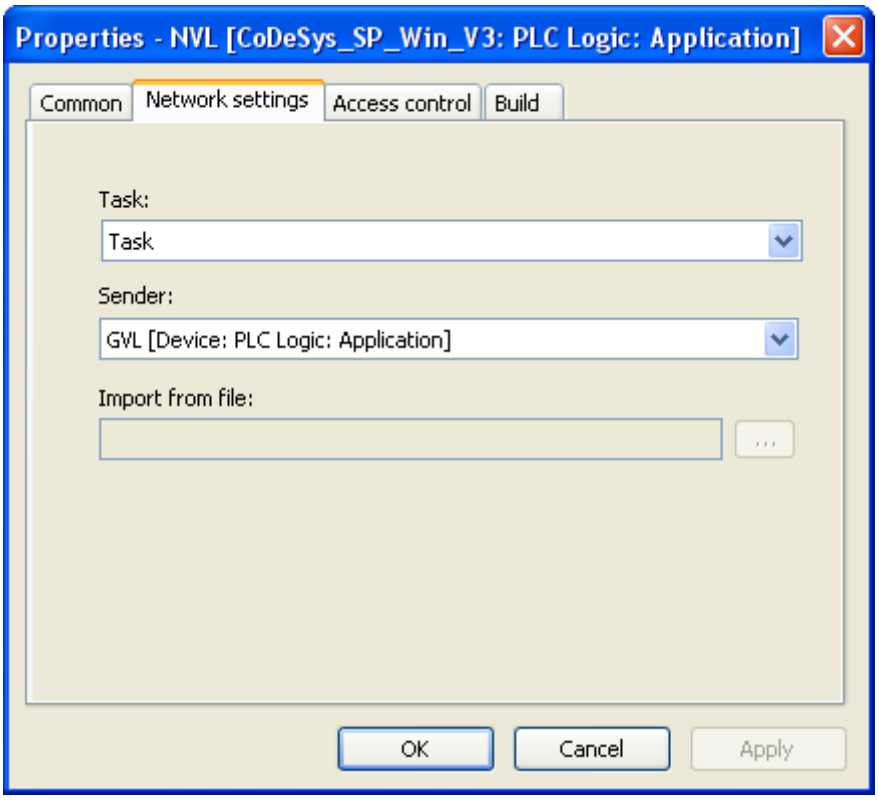


Abb. 297 Properties dialog, Category Network Settings

Sender: Here you see either the name of the Global Variables List of the sending device, which is referenced in the current NVL object, or "Import from file". If the name is specified, the squared brackets include the name of the device and the application. In case the Sender GVL is imported from a *.gvl file, which previously has been exported from the respective Global Network Variables List, the complete file path is shown here, for example: "D:\projects\pr9519\project_xy.gvl"

Task: Name of the task of the current device, which controls the data exchange of the network variables.

Import from file: In case the Sender GVL is specified via an *.gvl export file, which previously has been generated from the respective Global Network Variables List. the file path is shown here

Link to File (for Global Variables Lists export and import)

Global variables lists can be defined with the help of an external file in text format. Such a file can be generated by using the export functionality provided in the Properties dialog of the respective variables list: If option **Export before compile** is activated, automatically at each project compilation (e.g. by F11) a file with extension "**gvl**" will be created and be stored at the path specified in the **Filename** field. If option **Import before compile** is activated, an existing list export file can be read at each project compilation. This allows to import a GVL created from another project, for example in order to set up network variables communication.

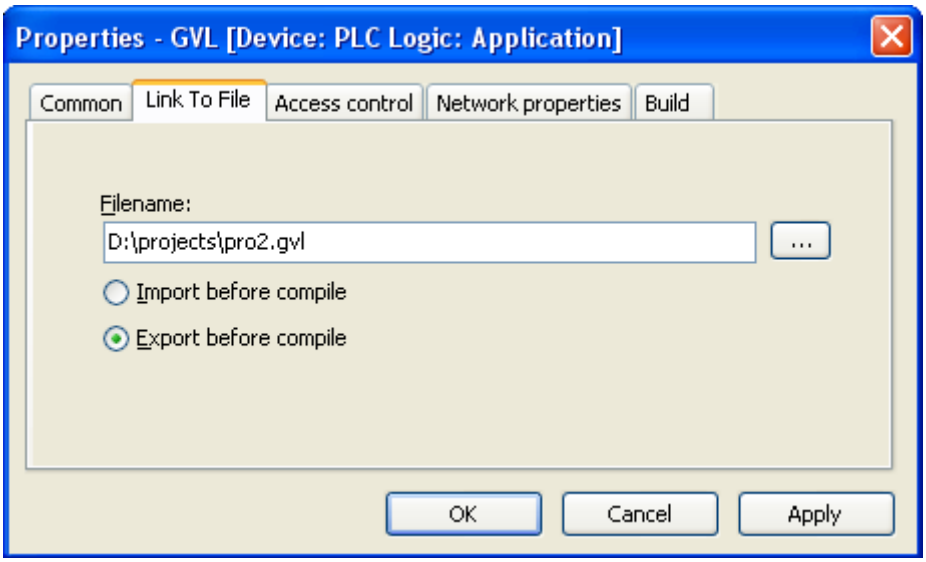


Abb. 298 Properties dialog, Category Network Settings

SFC settings (Flags)

This dialog allows settings for the current SFC object concerning compilation and flag handling. The items handled in the tabs 'Flags' and 'Build' correspond to those handled in the 'SFC' options dialog, where the default settings for SFC objects are defined.

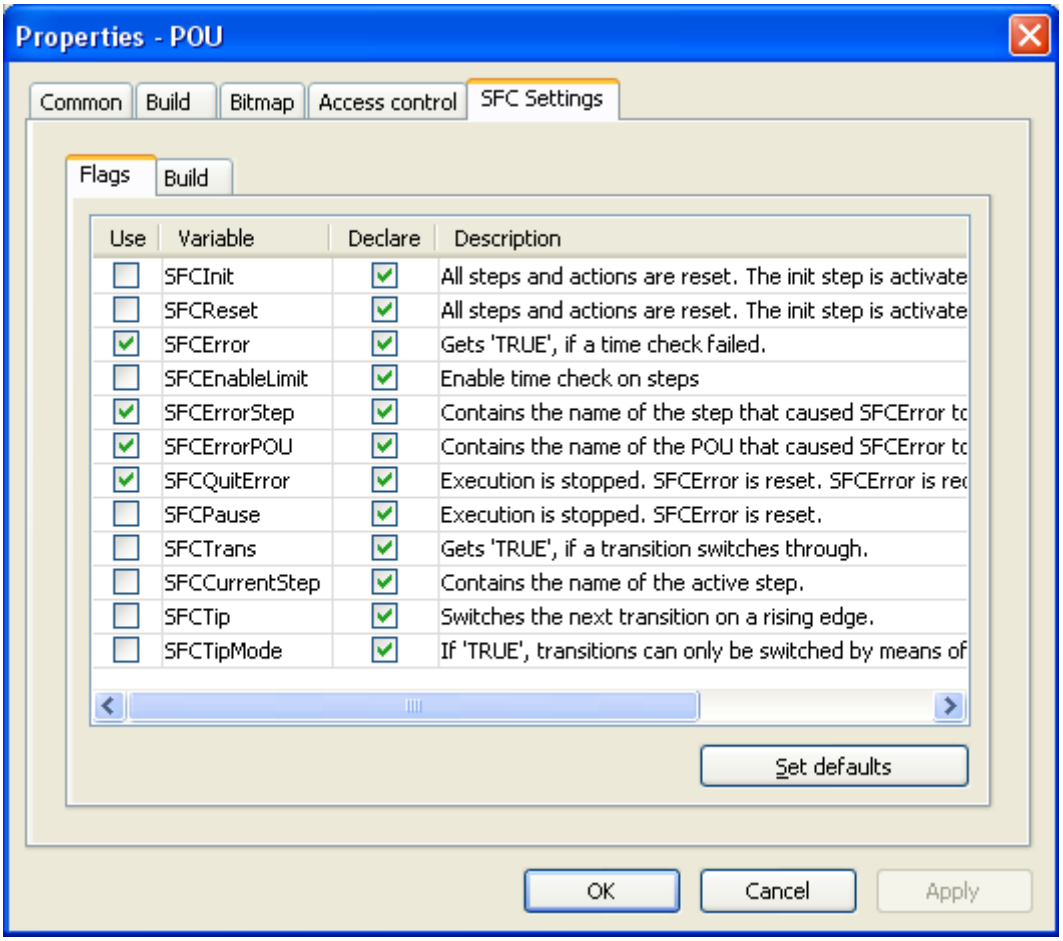


Abb. 299 Properties dialog, Category SFC Settings, Flags

The **Set defaults** button will exactly apply those defaults, which are currently defined in the SFC options dialog, to the current object.

CAM

The **CAM** tab serves to specify the global settings of the CAM, that is its dimensions, its period and continuity requirements.

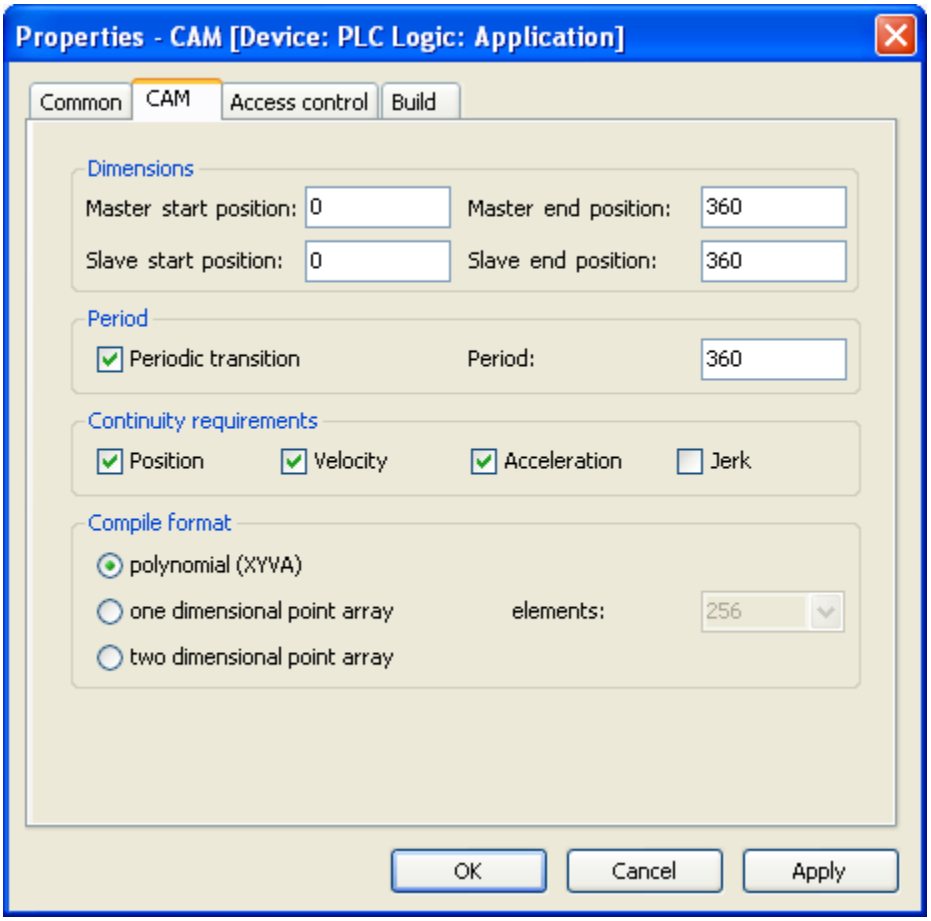


Abb. 300 Properties editor: CAM

Dimensions:

Master start / end position: Start and end position of the master define the set of master values and thus the scale of the horizontal axis of the CAM. The default settings are thought of in arc degrees and thus adjusted to 0 and 360.

Slave start / end position: The associated slave positions are determined by the CAM-defining mapping, but the cutout displayed by the graphs, that is the scale of the vertical axis, may be specified by the minimal and maximal value of the slave display.

Period:

Periodic transition: If this option is activated, the CAM- associated mapping is intended to be cyclic. That is, at the end position of the master the associated slave position is a multiple of its value at the start position of the master, whereas the value of slave velocity and acceleration equal their values at the start point.

Period: The slave period or feed rate is measured in units of the slave scale and can be assigned by typing in the white rectangle following "Period:"

Continuity requirements:

The activation/deactivation of the check boxes (**Position**, **Velocity**, **Acceleration**, **Jerk**) belonging to the continuity properties of the mapping has no impact on its editing. However an activation will entail an examination of the editing, reporting eventual violations of the continuity requirements in the message window (category "CAM"). Anyway, in case of the position mapping there is no way to edit jumps. The default setting will require continuity of velocity and acceleration as well. They may be omitted (for example in the special case of a mapping consisting of linear segments only), what might lead to kinks of the position graph. By default, the jerk, that is the fourth derivative, is not tested for jumps.

Compile format:

During compilation structure variables of type MC_CAM_REF are created. The description of the CAM segments is according to one of the following options:

polynomial (XYVA): polynomial description on particular points consisting of master position, slave position, slave velocity and slave acceleration.

one dimensional point array: 1-dimensional table of slave positions

two dimensional point array: 2-dimensional table of related master/slave positions

CNC Editor

CNC object settings, that are only relevant for the selected CNC object, can be set here. A further description can be found in Properties of a CNC object.

CNC Settings

The application-wide CNC settings are stored in the CNC settings object. A further description can be found in CNC Settings.

Full Screen

Symbol:

Default Shortcut: <Ctrl> + <Shift> + <F12>

This option (category 'View'), if activated, effects that the »PLC Designer« frame window will be displayed in full-screen mode. To toggle back to the previous mode deactivate the menu entry or press <Ctrl>+< Shift>+< F12>.

Start Page

Symbol:

This command (category 'File') by default is available in the View menu. It opens a view providing a selection of commands for quick starting with a new or recent project, version information, and a viewer for the »PLC Designer« home page.

In the Load and Save Options you can configure, that the start page automatically appears when the programming system gets started.

12.35.5 MessageView

View commands - MessageView

The 'Message View' plug-in provides for command category 'View' the command 'Messages' for viewing the messages window. By default the command is part of the 'View' menu. If necessary open the 'Customize' dialog to view resp. modify the menu configuration (notice category 'View').

Messages

Symbol:

This command (category View) opens the Messages window.

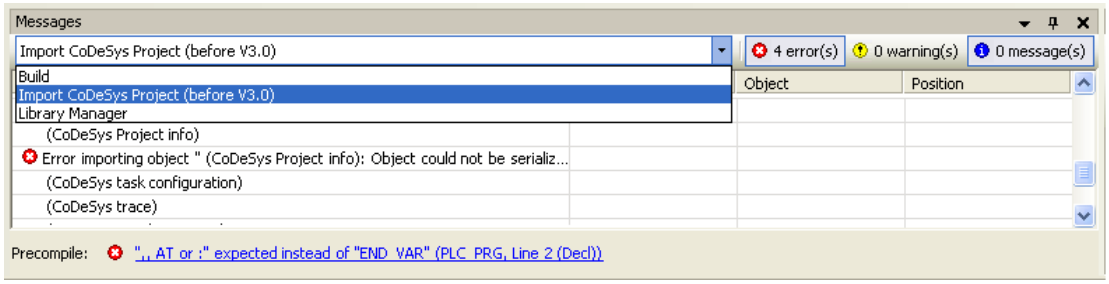


Abb. 301 Messages window, example

Messages might describe errors, warnings or just information.

Further on messages are categorized after the concerned component or functionality. For example messages on syntactical checks of the project are generated in categories 'Precompile', messages on the compilation of the project in category 'Build' (for example compile errors, code size). There also might be messages on the import of a project, on the Library Manager etc.

You can select the desired **message category** in the selection list below "Messages" (see in the figure shown above) with exception of the **Precompile** messages, which always are shown in the field below the messages table.

The messages belonging to the chosen category will be listed in the Messages table with the following information: **Description** (message text), **Project** (project name), **Object** (name of concerned object within the project), **Position** (for example line number, network number etc. within the object).

If you want to fade out or in a certain type of messages in the table, use the buttons in the upper right corner: **error(s)**, **warning(s)**, **message(s)**. These buttons in each case show the number of available messages and by a mouse-click on a button you can toggle the display of the respective message type.

You can **navigate** between the messages currently shown in the table resp. jump from a message to the position in the concerned object by using the **commands** 'Next message', 'Previous message' and 'Go To Source Position' . By a double-click on a message you also get to the source position. In order to get to the "source position" of a Precompile message perform a mouse-click on the blue and underlined message text.

12.35.6 SFC Implementation Object

View commands - SFC Implementation Object

The 'SFC Implementation Object' plug-in provides the command SFC Element Properties (command category 'View') which is used to open the Properties dialog for the currently selected element in a SFC.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

Element Properties

Symbol:

This command (category View) opens the **Properties** view for the currently selected SFC element. The properties, like step or transition name, comment, step time attributes and associated actions are displayed in a structured table. They can be edited by a mouse-click in the values field resp. in case of the 'Init Step' property by a click on the checkbox to activate or deactivate the option.

12.35.7 Source Control

View commands - Source Control

The 'Source Control' plug-in provides the command 'Pending Checkins' for command category 'View'. The command opens a dialog for handling the objects currently checked out from the project data base.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

Pending Checkins

This command of category 'View' is available in projects which are under Source Control. By default it is part of the View menu and opens the Pending Checkins dialog where you can view and handle the objects currently checked-out from the project database.

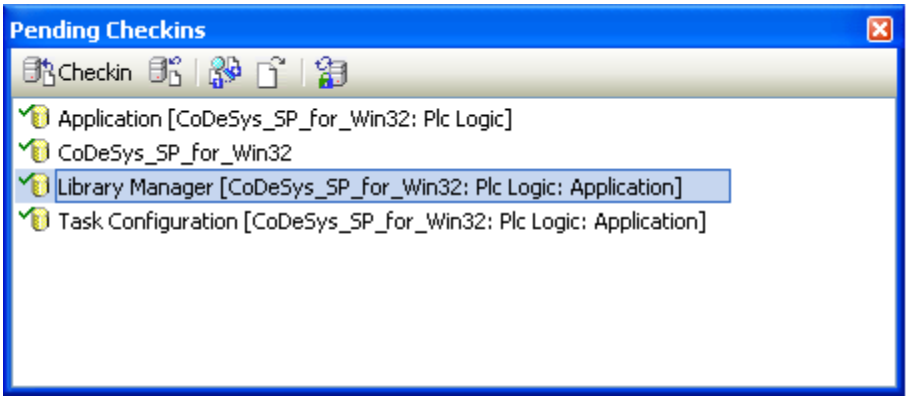


Abb. 302 Pending Checkins dialog

The dialog displays a list of the currently checked out objects. The toolbar provides the commands shown below, which per default also are available in the Source Control menu. With exception of the refresh command they always work on the object which is currently selected in the list. In case of multiple selection just the checking and undo checkout commands are available:

Checkin

Undo Checkout

Show Differences

Open Editor

Refresh Status

12.35.8 ToolBox

View commands - ToolBox View

The 'ToolBox View' plug-in provides the command ToolBox (command category 'View') which is used to open the toolbox for the currently used editor.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

ToolBox

This command (category View) opens the toolbox for the currently used editor in a window. Typically toolboxes are available with graphic language editors or the visualization editor and provide graphic programming elements which can be inserted into the editor via drag&drop.

12.35.9 Visualization Element

View commands - Visualization Element

The 'Visualization Element' plug-in provides the command Element Properties (command category 'View') which is used to open the Properties dialog for a visualization element.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

Visual Element Properties

Symbol:

This command (category View) opens the **Properties** view for the currently selected visualization element. The properties are displayed there in a structured table, the particular configuration parameters being grouped in folders. Those folders can be opened or closed via a mouse-click on the preceded plus- resp. minus-sign.

It depends on the visualization element, which parameters are currently available for configuration.

Note: This command might also be available in other graphic editors, also for the purpose of opening the properties view for the currently selected element (example: SFC Editor).

12.35.10 Watch

View commands - WatchList View

The 'Watchlist View' plug-in provides the command 'Watch' (command category 'View') which provides a submenu for opening the available watch lists.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

Watch

This command (category View) opens a submenu with commands **Watch 1**, **Watch 2**, **Watch 3**, **Watch 4**and **Watch all Forces**. These are used to open the respective watch list in a view window. Watch all Forces is a special watch view for the currently forced values.

By default this command is part of the 'View' menu. If necessary open the Customize dialog to view resp. modify the menu configuration (notice category 'View').

12.36 Window

Windows of the "window" type can be handled by the commands which belong to command category "window".

12.36.1 Window commands -»PLC Designer« Frame

The frame provides commands for command category 'Window' which can be used to handle the user interface windows.

By default the commands are part of the 'Window' menu. If necessary, open the Customize dialog, category Menu, to view resp. modify the current menu configuration.

The commands:

- Dock
- Float
- Auto hide
- Next Editor
- Previous Editor
- Next Pane
- Previous Pane
- Close Editor
- Close All Editors
- New Horizontal Tab Group
- New Vertical Tab Group
- Reset Window Layout
- Window <n>
- Windows...

12.36.2 Dock

This command (category 'Window') can be used to dock a window which previously has been undocked by a Float command and thus currently is placed as a floating window on the screen.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.3 Float

This command (category 'Window') can be used to un-dock a window which is currently docked, that is fix part of the»PLC Designer« user interface frame. The window will become "floating" and can be positioned anywhere on the screen. To dock a floating window, use the Dock command.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.4 Auto hide

This command (category 'Window') can be used to "hide" a window. The window will be represented by a tab at the border of the»PLC Designer« user interface and only be visible if you click with the cursor in this tab.

The command corresponds to the use of the **docking button** in the upper right corner of a window.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.5 Next Editor

Default Shortcut: <Ctrl> + <F6>

If several editor windows are currently opened, this command (category 'Window') can be used to change the focus from the current window to the nest one. This is the window represented with the tab to the right of the currently active window.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.6 Previous Editor

Default Shortcut: <Shift> + <Ctrl> + <F6>

If several editor windows are currently opened, this command (category 'Window') can be used to change the focus from the current window to the previous one. This is the window represented with the tab to the left of the currently active window.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.7 Next Pane

Default Shortcut: <F6>

This command (category 'Window') can be used in a window with two or several panes to get to the next pane.

Example: If in an object is opened in a ST-Editor window and the cursor is currently placed in the declaration part, with 'Next Pane' the focus will change to the implementation part.

See also: Previous Pane

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.8 Previous Pane

Default Shortcut: <Ctrl> + <F6>

This command (category 'Window') can be used in a window with two or several panes to get to the previous pane.

Example: If in an object is opened in a ST-Editor window and the cursor is currently placed in the implementation part, with 'Previous Pane' the focus will change to the declaration part.

See the description on how to customize the user interface for general information on arranging windows and views.

See also: Next Pane

12.36.9 Close Editor

Default Shortcut: <Ctrl> + <F4>

This command (category 'Window') closes the currently active editor window.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.10 Close All Editors

Default Symbol:

This command (category 'Window') closes all currently opened editor windows.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.11 New Horizontal Tab Group

Symbol:

This command (category 'Window') is available, if several editor windows are arranged in tabs.

It will place the currently active tab window in a new, separate tab group **below** the existing one. ("Active" window means that window, where the cursor was last placed.)

If you open a further object, that is a further editor window, this will be added to that tab group which contains the currently active window.

See also: New Vertical Tab Group

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.12 New Vertical Tab Group

Symbol:

This command (category 'Window') is available, if several editor windows are arranged in tabs.

It will place the currently active tab window in a new, separate tab group to the **right** of the existing one. ("Active" window means that window, where the cursor was last placed.)

If you open a further object, that is a further editor window, this will be added to that tab group which contains the currently active window.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.13 Reset Window Layout

This command (category 'Window') serves to reset all currently open views to their default docking positions. A confirmation prompt will be displayed before the reset gets executed.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.14 Window <n>

For each currently opened editor window the Window menu contains a command (category 'Window') named "<object name>", by which the window can be made active, that is the focus can be placed there. Behind the object name "(Offline)" will be added for offline views, "(Impl)" resp. "<instance path>" will be added in case of function block views.

See the description on how to customize the user interface for general information on arranging windows and views.

12.36.15 Windows...

This command (category 'Window') opens the 'Windows' dialog where you get listed all currently defined editor windows, that is windows used for editing any objects.

To activate a window, that is to set the focus to that window, select the respective entry and use button **Activate**.

To close one or several windows, select those entries and use button **Close window(s)**. Multiple selection is possible.

Tile horizontally: Currently not implemented.

Tile vertically: Currently not implemented.

Close will close the dialog.

See the description on how to customize the user interface for general information on arranging windows and views.

13 General Editors

13.1 CAM Editor

Der CAM Editor steht nur für »PLC Designer« SoftMotion Steuerungen zur Verfügung. Diese setzen die Installation des zusätzlichen Softwarepakets »PLC Designer« SoftMotion voraus.

13.1.1 A brief survey

The SoftMotion CAM disk is integrated in the »PLC Designer« programming interface. Within the CAM Editor CAM disks (or switches) can be implemented via graphical or tabular editing. When generating code for the related application, global data structures (CAM Data) are created, which can be accessed by the IEC program. For this purpose the library SM3_Basic is included automatically while adding the SoftMotion drive.

(Hint: Have a look on the exemplary programs and general hints on Using CAMs)

See in the following on this page:

- Definition of a SoftMotion CAM disk
- Creation of CAM disk
- Edit a CAM
- Compilation of the CAM

13.1.2 Definition of a SoftMotion CAM disk

A CAM disk prescribes a functional dependency of one parameter (Slave) on another (Master). The dependence can be expressed via a continuous mapping from the range of master values to slave values. More precisely, after subdividing the master-axis in appropriate intervals the graph of this mapping can be represented either by a straight line or by a quintic polynomial (degree 5) on each of these intervals.

Graph of a CAM disc: the master values are assigned to the horizontal axis, the slave values to the vertical one.

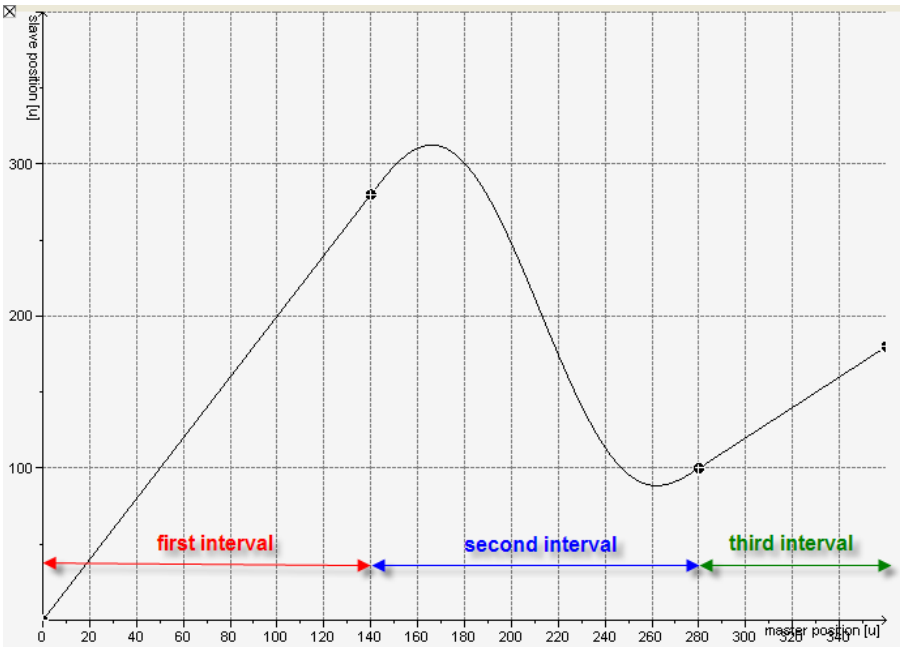


Abb. 303 Example of CAM disk graph

In the example, the master values range between 0 and 360. This range is divided into three intervals:

- the first one on the left: [0, 140]
- the second one in the middle: [140, 280]
- the third one on the right: [280, 360]

On the first and third interval the mapping is linear and therefore represented by a straight line. Hence, its first derivative (that is its gradient)

is constant, its higher derivatives are equal to zero.

On the second interval the mapping is a polynomial of degree 5. Hence, its first derivative is a polynomial of degree 4, its second derivative

(that is its curvature) a polynomial of degree 3, its third derivative a polynomial of degree 2 (and so on...).

If you think of the mapping as motion of the slave in dependence of the position of the master, its first derivative accords to the velocity of the slave, its second derivative to the slave acceleration.

Keeping this physical interpretation in mind it is evident that the mapping has to be continuous, that is its graph shows no jumps. In particular, this must hold for each point, where two intervals collide. Furthermore, in general also the first and second derivative are required to be continuous. (In fact, these three continuity demands claimed for the start point of an interval as well as for the endpoint uniquely define the quintic polynomial inserted between two lines.)
Additionally, if the CAM disk is defined as periodic transition, the values of the mapping and of the first and second derivative at the end position of the slave must fit the corresponding values at its start position (the value of the mapping at the end position being also allowed to be a multiple of the value at the start position).

Moreover, tappets, that is binary switches, can be added to the CAM disk at arbitrary positions. Especially, **Digital CAM tables** may be created, that is CAM tables with tappets only, by setting the slave position equal to zero over the whole range of master positions.

13.1.3 Creation of CAM disk

A CAM disk is an object that can be added to a project or application. Therefore select the project (either in the Device or in the POUs window) or the application and activate the entry **Add object...** of the context menu, whereon the "Add object"-dialog will open.

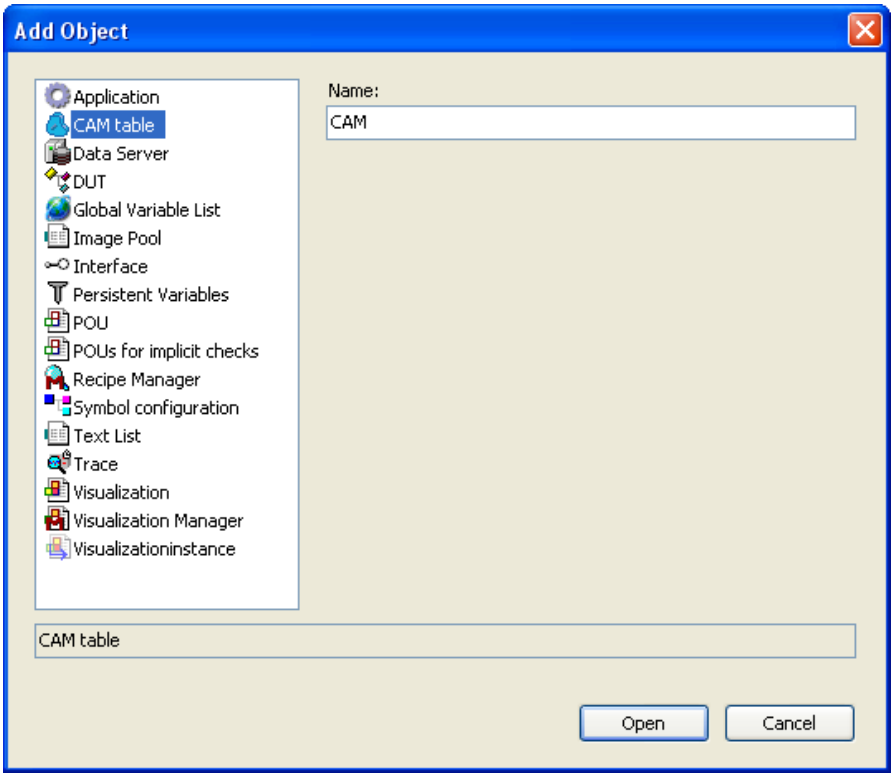


Abb. 304 Add object dialog

Therein pick the entry CAM table, enter a name ("CAM" by default) and confirm with **Open**. The CAM disk is inserted in the device tree (or POUs tree) and the CAM-Editor opens displaying the default settings of a newly created CAM disc.

13.1.4 Edit a CAM

The CAM editor consists of four parts, whereof the two former are opened automatically when creating a CAM:

- a **toolbox** providing tools for selecting and inserting points.
- a **main editor** entitled with the name of the CAM followed by its path in the device tree (for example CAM [PLCWinNT:Plc Logic:Application]). This window contains 4 tabs displaying the settings of the CAM either graphically or by means of tables.
- an element specific properties editor that opens after a click on the entry **Element properties** of menu item **View**.
- an object properties editor, containing general information on the CAM. It is opened either via the entry **Properties...** within the context menu of the CAM object in the device window or via the command **Properties...** ( ) within the category **View**.

The toolbox

Depending on the tab actually selected in the main editor the toolbox contains the following CAM editor tools:

- the **Select tool** is always available. Activate it by a click on Select, whereon a mouse click within the main editor will select points or mapping segments of the graphical descriptions of the CAM and entries in its tabular descriptions. Selected points or mapping segments will be displayed bold within the graphical representation, whereas selected table entries will be indicated by a thin blue frame in addition to a light blue inking of the related table row.
- the **Add point tool** is available only for the tab entitled "CAM". Activate this tool by a click on Add point, whereon a mouse click within the main editor will insert new points in the position, velocity or acceleration graph of the mapping. Remark that the graph of the jerk is not editable.
- the **Add tappet tool** is available only for the tab entitled "Tappets". Activate this tool by a click on Add tappet, whereon a mouse click within the main editor will insert a new tappet in a track.

See in the following description of the main editor some more detailed information on use cases of the tools:

Editing within the main editor

The main editor contains **four different tabs** displaying the mapping and its derivatives or the added tappets either **graphically** or with help of a **table**. Modifications in the graphic editor will be taken over to the table editor and vice versa. The edit mode depends on which tab is selected:

- the tab entitled "CAM": **graphic editor** for the definition of a CAM; the tab is separated into four sub-windows displaying (in top-down order) the graph of the slave position in black, the velocity in blue, the acceleration in green and the jerk in yellow.

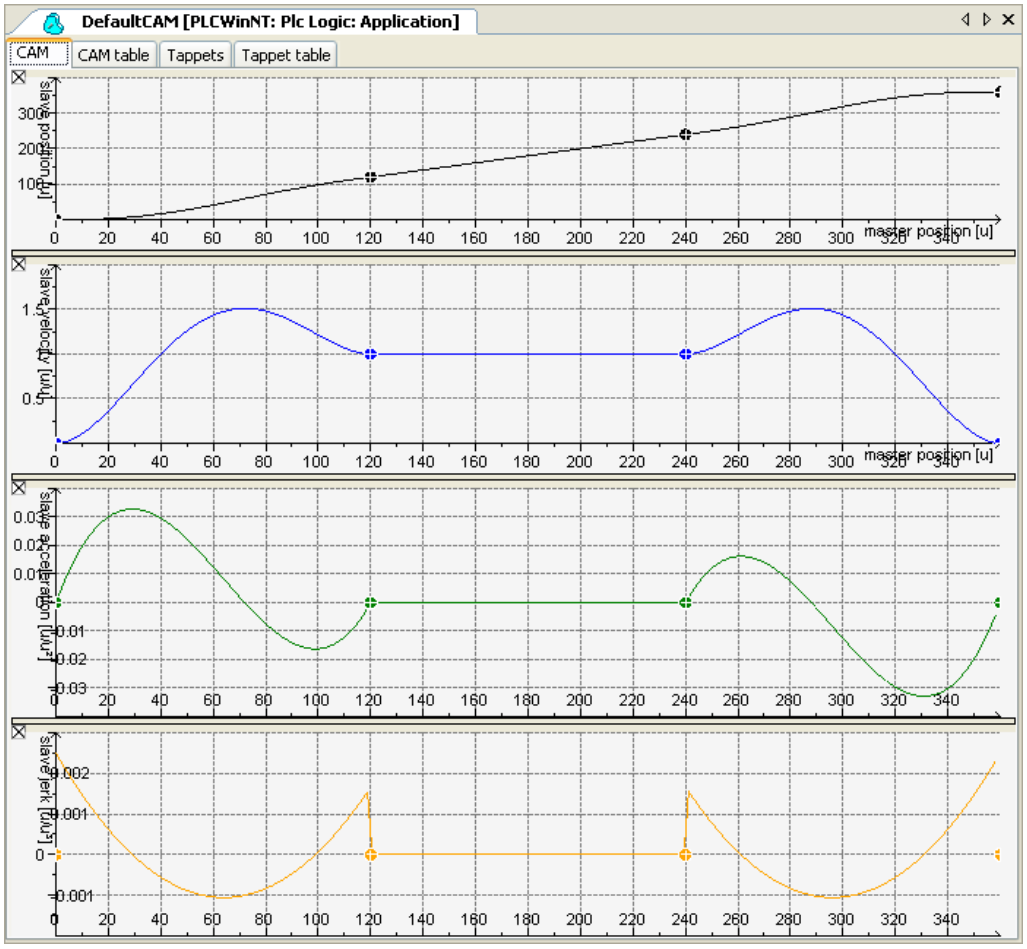


Abb. 305 Main editor: CAM

To adjust the height of the sub-windows move the horizontal divider by keeping the left mouse button pressed. You may also close a sub-window by a click on ⊠in its upper left corner. Afterwards, instead of the sub-window only an attributed string (**Position**, **Velocity**, **Acceleration** or **Jerk**) will be displayed. Reopen the sub-window by a click on this string.
In each sub-window the horizontal axis of the coordinate system displays the range of **master values** ([0,360] by default). In the position diagram the vertical axis is scaled according to the value range prescribed in the Properties editor, whereas it is scaled automatically according to the related value range of velocity, acceleration or jerk in each of the corresponding diagrams.

Except for the jerk all diagrams are editable. As velocity, acceleration and jerk are derived by derivation, a modification of position, velocity or acceleration will affect each of the three other displayed graphs.

The **default mapping** of a CAM created is equipped with four points defining the three intervals [0,120], [120,240] and [240,360]. The assigned slave positions equal the master positions, the assigned values of velocity, acceleration and jerk equal zero except for the second and third point which have an assigned velocity of 1. On each of these intervals the corresponding mapping piece is of type Poly5 (that is a quintic polynomial).

You can change the default mapping in the following ways:
Select the Add point tool of the toolbox to **insert a further point**. You can indicate its appropriate position in the coordinate system by a click on the left mouse button within any of the upper three diagrams. The interval containing the master value of the new point will be separated in two parts, whereon two newly created mapping segments (of type Poly5) will be adjusted to their start and end values.

Use the Select tool of the toolbox to **modify position**, **velocity** or **acceleration** assigned to a point within the corresponding diagram while keeping the left mouse button pressed. Alternatively you can make use of the element specific properties editor.

After its selection you may also **delete a point** (as long as it is not associated to the start or end position of the master) by pressing the delete key .

- the tab entitled "CAM table": **table editor** for the definition of a CAM:The first five columns of the displayed table contain attributes related to points (that is the horizontal component X and vertical position **Y** of its position, its assigned velocity **V**, acceleration **A** and jerk **J**). The following five attributes belong to mapping segments (that is the **Segment Type**, the extreme values (**min**..., **max**...) of the slave **position** and the maximal value of **velocity** and **acceleration** within the associated interval). The first row of the table is reserved for the **start position** of the master (and the related slave values), the last one for its **end position**. An odd number of enclosed rows (one at least!) determine either mapping pieces or points in alternating order.

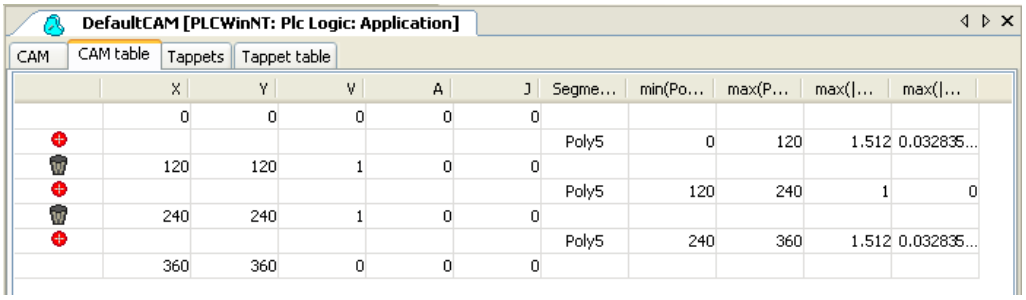


Abb. 306 Main Editor: CAM table

To **add a point** within an already existing interval click on the symbol preceding the row containing the attributes of the associated mapping segment. The two new rows will be inserted beyond, the first containing the entries related to the mid point of the interval, the second one containing the attributes of a further mapping segment. Noticeless of the former segment type associated to the whole interval the segment type will be set to Poly5 on each of its halves by default.

To **modify a table entry** you select it by a click. Notice that the maximal values contained in the last 4 columns result from the settings of the related mapping segment and thus cannot be edited. After a selection has been done you may enter the desired value by typing or chose the segment type from the selection provided upon a further click into the segment type entry. Be aware of the continuity requirements! In case of violation your modification will rest without effect or entail automatic modifications of other table entries.

To **delete a point** (as long as it is not associated to the start or end position of the master) click on the symbol that precedes the table row belonging to this point. That row as well as the succeeding one will be removed from the table. The interval at the left of the point removed will be merged with the interval on the right. The associated mapping segment is of type Poly5 no matter what type the former two single segments were of.

- the tab entitled "Tappets": **graphic editor** for the definition of tappets tracks. A tappet track in this context defines one or several master-position-depending tappets (that is tappet events) for the same tappet switch. On top a horizontal axis displays the range of master positions. Beyond follow the tracks headed by their identification number that can be augmented or diminished by use of the arrows at their right hand-side (by default you find two tracks, the first one already equipped with two tappet actions).

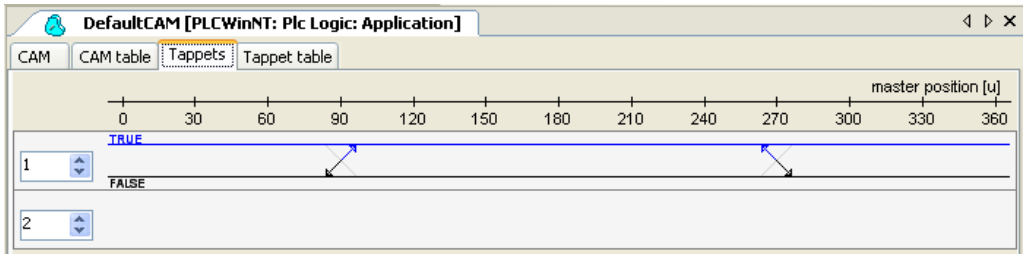


Abb. 307 Main Editor: Tappets

Select the **Add tappet** tool of the toolbox to add a new tappet action. You can indicate its appropriate position by a click on the left mouse button within the desired track. The new tappet will be inserted and instead of the Add tappet tool and now the Select tool will automatically be activated. After selecting a tappet by a click in its center you may **move the tappet** while keeping the mouse button pressed or **delete the tappet event** by pressing the delete key.

You can assign different events ("switch on", "switch off", "invert", "none") to the pass of a tappet in either positive (ascending master values) or negative direction. Have a look at the table beyond for all possible combinations. All attributes of tappets of one track, that is of the same identification number, refer to the same **tappet bit**, that is a variable of boolean type. By default, a cam has the attributes "switch on" on positive pass and "switch off" on negative pass , so that the boolean variable will be set to TRUE on positive pass and set to FALSE on negative pass. The event "invert" will negate the value of the boolean variable, whereas "none" has no impact at all. You may **change the attributes of the cam** each by a click on the related end of the cross to turn it on or off.

Table of possible combinations for tappet attributes:

Tappet symbol	positive pass	negative pass
	none	none
	switch on	none
	switch off	none
	none	switch on
	none	switch off
	switch on	switch off
	switch on	switch off
	switch off	switch on
	switch off	switch off
	invert	none
	none	invert
	switch on	invert
	invert	switch on
	invert	switch off
	switch off	invert
	invert	invert

- the tab entitled "Tappet table": **table editor** for the definition of tappets tracks. Below a line showing the Track ID, preceded by the symbol, as much lines are displayed as tappets are defined for this track. These lines are preceded with the symbol and each contains the tappet events definitions. So the following table columns are available: Track ID (identification number of the tappet track); X: the associated master position; positive pass / negative pass: events defined for passing the master position in positive or negative direction.

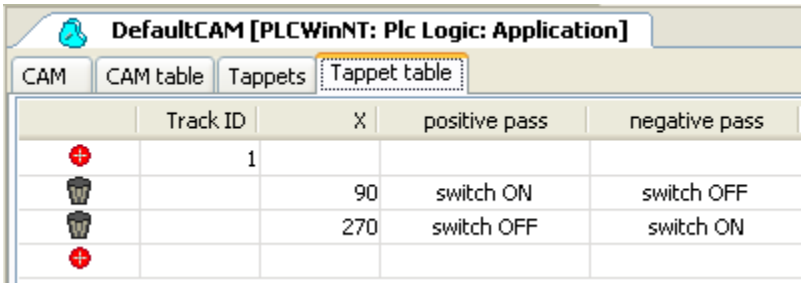


Abb. 308 Main Editor: Tappet table

Attributes of tappets of one track, that is of the same identification number, refer to the same tappet bit, that is a variable of boolean type. By default, a cam has the attributes "switch on" on positive pass and "switch off" on negative pass, so that the boolean variable will be set to TRUE on positive pass and set to FALSE on negative pass. The event "invert" will negate the value of the boolean variable, whereas "none" has no impact at all.
To add a tappet within an already existing track click on the symbol preceding the track ID. A new row will be inserted beyond with master position 0 and default pass attributes.

To modify a table entry you select it by a click. Afterwards you may either type a new Track ID or master position or chose the attribute from the selection provided upon a further click into the positive/negative pass field.
To delete a tappet click on the symbol that precedes the table row belonging to this point.

To add a new track click on the symbol of the last table row. The two new rows will be inserted, the first containing a track number not yet allocated, the second already providing a default tappet with master position 0 and default pass attributes. A track itself is deleted as soon as all of its tappets have been removed.

Editing within the element properties editor

The **properties of an element** currently selected in a graphical editor for a CAM or tappet, can be displayed and edited in the "Properties" window, which by default is available next to the toolbox window.

Click here for a detailed description.

Editing within the CAM object properties editor

The basic settings for the CAM object currently selected in the device tree can be defined in the **object properties** dialog.

Click here for a detailed description.

13.1.5 Compilation of the CAM

During the compilation structure variables of type **MC_CAM_REF** are created. They contain a description of each mapping piece describing the CAM. Data structures of this type are used as input of the MC_CamIn module. The structure is part of the library **SM3 Basic.library**.

13.1.6 Using CAMs - General Hints

The library SM3_Basic.library provides appropriate modules for the handling of CAMs. The library can be inserted in the project by use of **Add library...** in the library manager. Otherwise it will be inserted automatically with adding a SoftMotion drive to the device tree.

The following paragraphs give deeper insight in the meaning of particular parameters (periodicity, offset etc.) and the possibility to attach a CAM to another:

- Effects of module parameters
- Switching between CAMs

Effects of module parameters

CAM Properties (Properties dialog of the CAM):

- **Periodic transition, Period**

  A periodic CAM can be processed repeatedly without exhibiting "jumps" at the transition from one period to its next one. As far as the check box in front of "Periodic transition" is activated ☑, the slave position associated to the end position of the master is a multiple of the slave position value associated to the start position of the master, whereas the value of slave velocity and acceleration equal their values at the start point. The period or feed rate is measured in units of the slave scale.

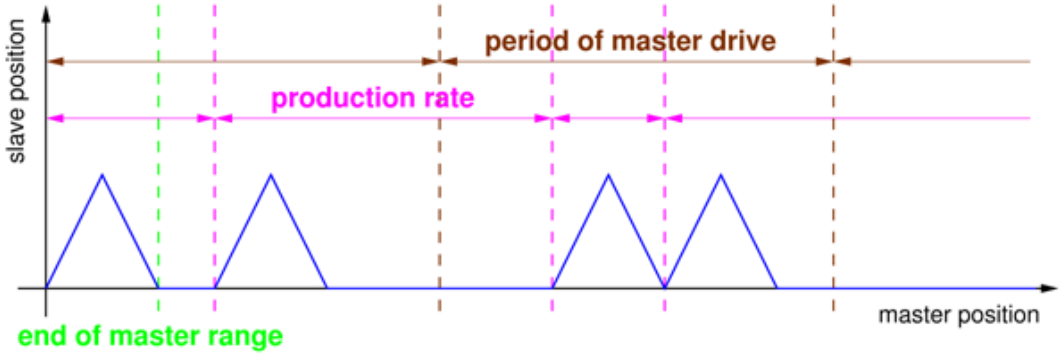
  Even if "Periodic transition" is de-activated the related CAM may be processed consecutively. In this case it's left to the user to make sure that the transitions are consistent to the desired extent.

Module MC_CAMTableSelect:

- **Periodic**

  This parameter determines whether a CAM will be processed again after the end position of the master has been reached. This will be the case of the parameter set to TRUE.

  Have a look to the following picture for a clear differentiation between the occurring periods: As example you may think of a band-conveyor transporting identical objects. The master values range from 0 to the length of the objects. Within this range an object is approached and left behind again by a tool (for example for stamping a mark) whose position is driven by the slave (blue graph).



  Of course, the master range of a CAM (that is the range of defined master values) is not equatable with the period of the master drive (that is one turn-over of the band in the example). Therefore, the assignment 'SlavePosition = CAM(MasterPosition)', that is the definition of the slave position in dependence of the master position via the CAM, is valid only in the first cycle of the CAM.

  A new cycle of the CAM driving the tool has to be started with the arrival of a new object. As there is a distance between the objects on the band (and this distance even may differ), the production rate, that is the time slice between successive starts of the CAM, is not identical to the master range of a CAM.

  In case of the parameter set to FALSE, as soon as the end position of the master is reached, the output variable EndOfProfile of module CamIn will be set to TRUE and the slave will be fixed up to its actual position. Remark that the CAM action is not terminated by quitting the defined master range. When the master reenters the valid range, the slave will still be controlled according to the CAM.

- **MasterAbsolute**

  If this input is TRUE, the CAM will be started at the current master position. This point may also be situated in the middle section of the CAM. However, if the point is outside of the range specified by the CAM, an error message will be generated.

  If this input is FALSE, the CAM will be shifted to the current position, that is afterwards the zero-point of the CAM master will be located at the current master position. A CAM, whose master range does not include "0", cannot be used within this mode, an error message ("...master leaving specified range...") will be generated.

- **SlaveAbsolute**

  The parameter CamTableSelect.SlaveAbsolute influences the StartMode of the slave (primarily described by the parameter CamIn.StartMode). The interaction of both parameters results into a StartMode displayed in the table beyond:

CamIn.StartMode CamTableSelect.SlaveAbsolute	absolute	relative	ramp_in	ramp_in_pos	ramp_in_neg
TRUE	absolute	relative	ramp_in absolute	ramp_in_pos absolute	ramp_in_neg absolute
FALSE	relative	relative	ramp_in relative	ramp_in_pos relative	ramp_in_neg relative

For a detailed description of the modes see MC_Camin.StartMode.

Module MC_CamIn

- **MasterOffset, MasterScaling**

  These inputs transform the master position according to the following formula and use the transformed position X for the evaluation of the CAM:

  X = MasterScaling*MasterPosition + MasterOffset

  Thus, if the value of MasterScaling is greater than 1, the CAM will be processed at a higher speed rate, for a value less than 1 the speed rate will be lowered.

- **SlaveOffset, SlaveScaling**

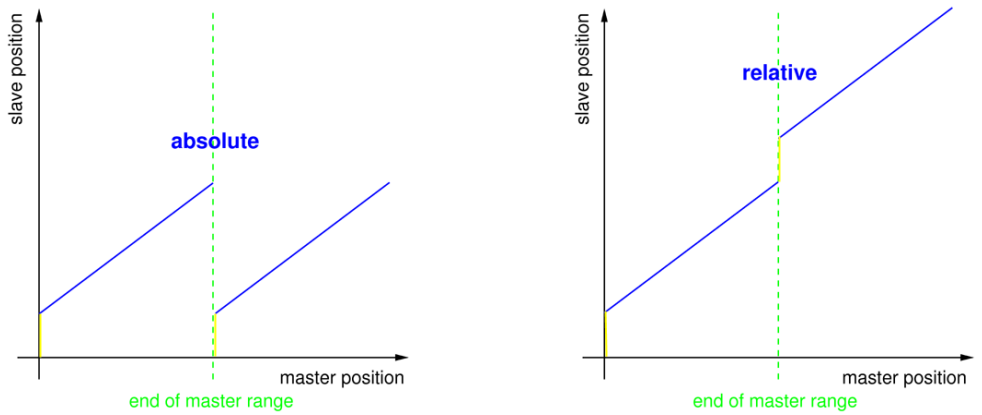
  These inputs shift resp. stretch the CAM in slave direction. The CAM will be stretched first, afterwards it will be shifted according to the following formula:

  Y = SlaveScaling*CAM(X) + SlaveOffset

  A SlaveScaling >1 effects a stretching of the CAM, thus the range of the slave will increase; accordingly a SlaveScaling<1 effects a compression.

- **StartMode: absolute / relative / ramp_in / ramp_in_pos / ramp_in_neg**

  The picture beyond demonstrates the effect of the option 'absolute' compared to 'relative' for an exemplary CAM. The CAM is defined on a master range that is limited by the vertical axis at left and the dashed green plumb line at right. As we are concerned with a periodic CAM, it will be processed again at master positions exceeding the range.



absolute: At the start of the new CAM cycle the CAM is evaluated independently from the current slave position. This results in a jump, if the slave position related to the start master position differs from the one related to the end master position.

relative: The new CAM will be shifted according to the current slave position; that is the position the slave holds at the end of the former CAM cycle will be added as 'slave offset' to the values evaluated by the CAM. Nevertheless this results in a jump if the slave position related to the start master position differs from 0 in the CAM definition.

ramp_in: Starting the CAM potentially occurring jumps will be prevented by a compensating movement (according to the limitations VelocityDiff, Acceleration, Deceleration). Thereby, as far as the slave is rotatory, the option **ramp_in_pos** will compensate in positive direction only, whereas **ramp_in_neg** will compensate in negative direction. For linear slaves the direction of the compensation will result automatically, that is **ramp_in_pos** and **ramp_in_neg** are interpreted like **ramp_in**.

Switching between CAMs

Basically a switch between two CAMs is possible at any time. However, there are some items to be thought about:

Within the CAM editor the position of the slave has been defined as output of the CAM function evaluated at a master position that is contained in the master range, what can be expressed by the simple formula:

SlavePosition = CAM(MasterPosition)

As the actual period of the master drive generally is differing from the master range in the definition of the CAM, the master position has to be scaled to the function domain to serve as adequate input of the CAM function:

SlavePosition = CAM(MasterScale*MasterPosition + MasterOffset)

In analogical manner the output of the function (that is the virtual slave position) has to been scaled, if the start of a CAM in absolute mode gave rise to a jump:

SlavePosition = SlaveScale*CAM(MasterPosition) + SlaveOffset

In the worst case both scalings have to be adopted, so that the SlavePosition in fact results in the more complex formula

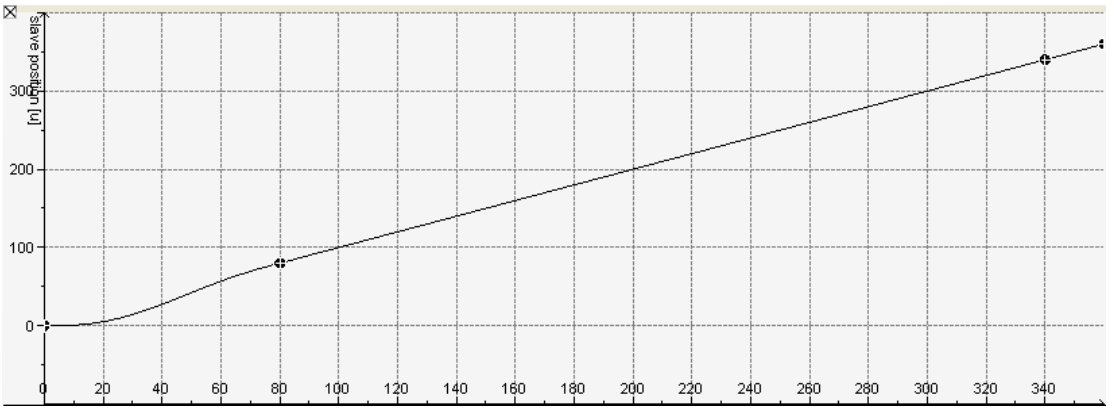
Slaveposition = SlaveScale*CAM(MasterScale*Masterposition + MasterOffset) + SlaveOffset

The appropriate scales and offsets may change at the end of each CAM cycle.

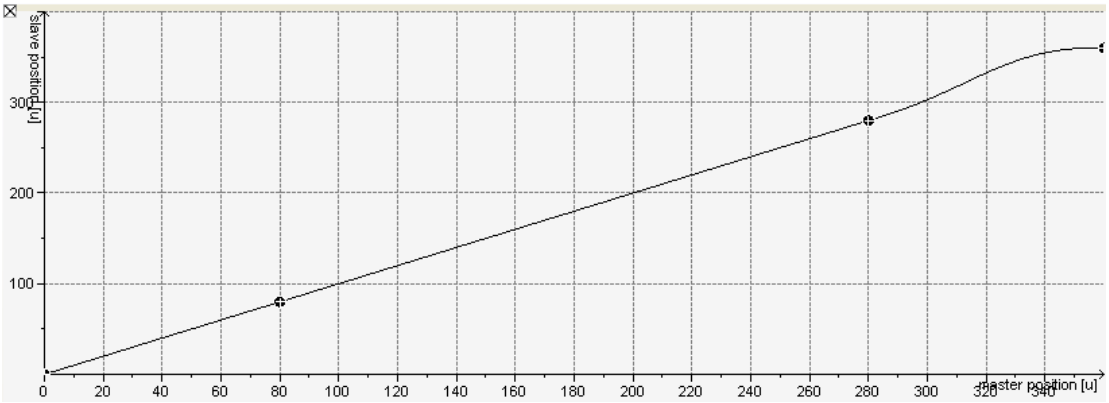
Unfortunately, a restart of CAM module MC_CamIn will delete its memory and therefore also the scaling and offset values. Hence, the CAM function as defined will be used yielding different slave values in general. For this reason it is recommended to restart MC_CamIn-FB only in case that a different CAM shall be processed.

Have a look at the following example of a switch from CAM1 to CAM2:

CAM1 is made up of a quintic polynom followed by two lines.



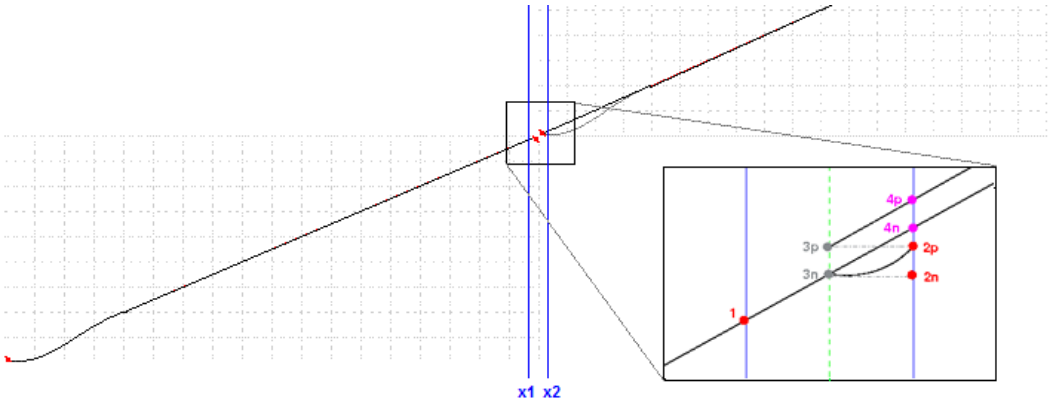
CAM2 is made up of two lines followed by a quintic polynomial.



A switch between the two CAMs should respect the following issues:

1. Velocity and acceleration at the end point of the first CAM should match the corresponding values at the start point of the second CAM in order to avoid jumps. In the example case this requirement is fulfilled: the end point of CAM1 and the start point of CAM2 both have velocity 1 and acceleration 0.

2. The second CAM may be started relative, if its slave position assigned to the start equals 0. However, the previous CAM has to be run in non-periodic mode. Otherwise, if the first CAM was periodic, having processed its first cycle the assignment of the slave position is not unique anymore:



The box at the right side displays the transmission from CAM1 to CAM2 in detail. The blue lines indicate evaluations of the CAM function at master position x1 and x2. Let us consider the **periodic** case at first:

MasterAbsolute := TRUE; SlaveAbsolute := FALSE;	
CAM(x1, CAM1, PERIODIC:=TRUE);	This call evaluates the CAM at master position x1, which is smaller than the end position of the range of CAM1. So, CAM1 is evaluated in the usual way resulting in Point 1 for the slave position.
CAM(x2, CAM1, PERIODIC:=TRUE);	When the module is called for the next time, the master position x2 has exceeded the end position of CAM1 (indicated by the green dashed line and equal to the abscissa of point 3p). Therefore, the EndOfProfile Flag is raised. As CAM1 is started in periodic mode, it will be restarted at end of range of CAM1what will return point 2p as result of the call.
CAM(EXECUTE:=FALSE);	Switch to the new CAM.
CAM(x2, CAM2, PERIODIC:=TRUE);	Second evaluation at master position x2. This time the new cam CAM2 gets evaluated. As CAM2 is started in relative mode the actual position of the slave (2p) is added as offset to the CAM2 mapping. So, the start point of its graph is shifted to point 3p. Its evaluation at master position x2 results in point 4p, thus in an undesirable jump.

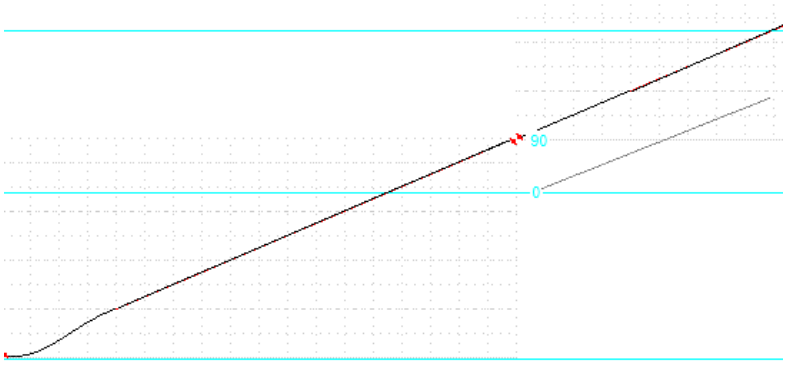
Choose the **non-periodic** mode for avoiding jumps:

MasterAbsolute := TRUE; SlaveAbsolute := FALSE;	
CAM(x1, CAM1, PERIODIC:=FALSE);	This call evaluates the CAM at master position x1, which is smaller than the end position of the range of CAM1. So, CAM1 is evaluated in the usual way resulting in Point 1 for the slave position.
CAM(x2, CAM1, PERIODIC:=FALSE);	When the module is called for the next time, the master position x2 has exceeded the end position of CAM1 (indicated by the green dashed line and equal to the abscissa of point 3n). Therefore, the EndOfProfile Flag is raised. As CAM1 is started in non-periodic mode, the slave position related to x2 (2n) is identical to the slave position related to the end of the master range of CAM1, that is to point 3n.
CAM(EXECUTE:=FALSE);	Switch to a new CAM.
CAM(x2, CAM2, PERIODIC:=FALSE);	Second evaluation at master position x2. This time the new cam CAM2 gets evaluated. As CAM2 is started in relative mode the actual position of the slave (2n) is added as offset to the CAM2 mapping. So, the start point of its graph is shifted to point 3n. Its evaluation at master position x2 results in point 4n situated on the same line as point 1 and 3n.

3. Wanting to start the CAMs in absolute mode pay attention to an adequate start position of the slave. If the master range equals the period of the slave there should be no complications for switching and it does not matter whether the CAM gets started in periodic mode or not.

   In the example described above CAM2 can be started in absolute mode, if both periods (of master and slave position) equal 360° in accordance to the range of master values of CAM2.

   Otherwise, if the period of the slave is 270° for example (indicated by light blue lines), the option absolute is not admissible without further arrangements:



   In this case the slave is located at 90° at the transition of CAM1 to CAM2. The start of CAM2 in absolute mode results in a jump to 0° on the grey line. However you can handle this situation properly by setting the slave offset to the adequate value of 90°.

13.1.7 CAM Data Structures

In course of the compilation of the project the edited CAM data is internally converted into a list of global variables ("CAM Data").

Each CAM is represented by a structure variable of type MC_CAM_REF that can be accessed by the IEC-program as well as by the CAM preprocessing functions and function blocks. In order to get an compilation void of detected errors, the appropriate structure definitions must be available in the IEC program by adding the library SM3_Basic.library.

Furthermore the data structure _SMC_CAM_LIST (ARRAY OF POINTER TO MC_CAM_REF) referring to the particular CAMS via pointers and a data structure pointing on all available CAMs are created.

Of course the corresponding data structures can also be created resp. filled in by the IEC program during run time. For this reason a detailed description of all data structures that are not for internal use only, will follow.

Short Survey on data structures

In the course of the compilation of the project the edited CAM data is internally converted into a list of global variables
Each CAM is represented by a structure variable of type **MC_CAM_REF** that can be accessed by the IEC-program as well as by the CAM preprocessing functions and function blocks.

The appropriate data types are provided by the library **SM3_Basic.library**, which is included in the library manger automatically with adding a SoftMotion drive to the device tree.

Of course it is possible, that the variable describing a CAM is created or assigned by the IEC program during run time. For this reason a detailed description of all data structures that are not for internal use only, will follow:

MC_CAM_REF

ARRAY OF SMC_CAMXYVA

SMC_CAMTable_<type of variables>_<number of elements>_1

SMC_CAMTable_<variable-type>_<number of elements>_2

To allow easy acces to the structure variables, these are summarized and listed in the global variable **g_CAMManager** with the property **Count** and the method **GetProgram(int n)**.

Example:

Access to the data object of the structure **MC_CAM_REF:**

```
PROGRAM CAMManageRef
VAR
   pCAM_Ref: POINTER TO MC_CAM_REF;
   n: INT;
   i: INT;
END_VAR

n := g_CAMManager.Count;
FOR i:=0 TO n-1 DO
pCAM_Ref := g_CAMManager.GetCAM(i);
 (* pCAM_Ref verarbeiten *)
END_FOR
```

MC_CAM_REF

This data structure represents a generic CAM and includes the following components:

wCAMStructID: WORD	has a fixed value and is used to check whether the data structure given to the module as an input is a MC_CAM_REF structure
xStart, xEnd : LREAL	start- and end position of the master defining the range of master values available
byType : BYTE	describes the CAM type, that is the way in which the CAM is represented. The following settings are valid: 1: equidistant, 1-dimensional table of slave positions 2 :non-equidistant, 2-dimensional table of related master/slave positions 3: polynomial description on particular points consisting of master position, slave position, slave velocity and slave acceleration (XYVA)
byVarType : BYTE (in case of byType=1 or byType=2)	defines the type of variables that enter in the curve table: 0: INT 1: UINT 2: DINT 3: UDINT 4: REAL 5: LREAL
nElements : INT	number of elements, that is, depending on CAM type, the number of slave positions, master/slave positions or XYVA points

byInterpolationQuality : BYTE (in case of byType=1 or byType=2)	interpolation degree, is 1 in case of linear (default) or 3 in case of cubic interpolation
pce: POINTER TO BYTE	pointer to actual data element depending on type: 1: SMC_CAMTable_<VarType>_<nElements>_1 2: SMC_CAMTable_<VarType>_<nElements>_2 3: ARRAY OF SMC_CAMXYVA
nTappets: INT	number of switch actions
pt: POINTER TO SMC_CAMTappet	pointer to ARRAY OF SMC_CAMTappet
strCAMName:STRING	name of the CAM

ARRAY OF SMC_CAMXYVA

A XYVA-CAM is represented by an array of variable of type SMC_CAMXYVA describing each point of the CAM by means of

dX (master position)

dY (slave position)

dV (first derivative dY/dX; slave velocity)

dA (second derivative d^2Y/dX^2; slave acceleration).

At a minimum the start and end point of the CAM must be enclosed in the array.

SMC_CAMtable_<variable_type>_<number_of_elements>_1

This data structure represents an equidistant curve table. It has the following components: Table: ARRAY [0..<number of elements>-1] OF ARRAY[0..1] OF <type of variables>	array of slave positions; the slave positions related to the start and end position of the master must be enclosed
fEditorMasterMin: <type of variables>	storing of master values in Table is scaled to the range [fEditorMasterMin, fEditorMasterMax] in SoftMotion units
fEditorMasterMax: <type of variables>	
fEditoSlaverMin: <type of variables>	storing of slave values in Table is scaled to the range [fEditorSlaveMin, fEditorSLaverMax] in SoftMotion units
fEditorSlaveMax: <type of variables>	
fTableMasterMin: <type of variables>	storing of master values in Table is scaled to the range [fTableMasterMin, fTableMasterMax] in Table units
fTableMasterMax: <type of variables>	
fTableSlaveMin: <type of variables>	storing of slave values in Table is scaled to the range [fTableSlaveMin, fTableSlaveMax] in Table units
fTableSlaveMax: <type of variables>	

SMC_CAMtable_<variable_type>_<number_of_elements>_2

This data structure represents a non equidistant curve table. It has the following components:

Table: ARRAY [0..<number of elements>-1] OF ARRAY[0..1] OF <type of variables>	array of master/slave position pairs; start and end position must be enclosed; index 0 refers to the master, index 1 to the slave
fEditorMasterMin: <type of variables>	storing of master values in Table is scaled to the range [fEditorMasterMin, fEditorMasterMax] in SoftMotion units
fEditorMasterMax: <type of variables>	
fEditoSlaverMin: <type of variables>	storing of slave values in Table is scaled to the range [fEditorSlaveMin, fEditorSLaverMax] in SoftMotion units
fEditorSlaveMax: <type of variables>	
fTableMasterMin: <type of variables>	storing of master values in Table is scaled to the range [fTableMasterMin, fTableMasterMax] in Table units
fTableMasterMax: <type of variables>	
fTableSlaveMin: <type of variables>	storing of slave values in Table is scaled to the range [fTableSlaveMin, fTableSlaveMax] in Table units
fTableSlaveMax: <type of variables>	

Example of a manually created CAM

The example shows how a CAM is created within an IEC program, that is without making use of the editor. The following lines have to be entered in the declaration part:

```
CAM: MC_CAM_REF:=(
byType:=2, (* non-equidistant )
byVarType:=2, (* UINT *)
nElements:=128,
xStart:=0,
xEnd:=360);
Table: SMC_CAMTable_UINT_128_1:=(
fEditorMasterMin := 0, fEditorMasterMax := 360,
fTableMasterMin := 0, fTableMasterMax := 65536,
fEditorSlaveMin := 0, fEditorSlaveMax := 360,
fTableSlaveMin := 0, fTableSlaveMax := 65536);
```

The following lines have to be entered in the implementation:

```
(* Create CAM (of linear type for example); unique *)
FOR i:=0 TO 127 DO
Table.Table[i][0]:=Table.Table[i][1]:=REAL_TO_UINT(i / 127.0 * 65536);
END_FOR
(* connect pointers; must be done in each cycle !!! *)
CAM.pce := ADR(Table);
```

The CAM created can be used as input for module MC_CamTableSelect. The output of this module in turn can be used for MC_CamIn.

13.1.8 CAM: Element Properties

The properties of elements used in the CAM Editor can be viewed and modified in the Properties editor window.

To open this editor pick the entry **Element Properties**, by default available in the **View** menu. An window entitled "Properties" will appear. If currently no element (point, mapping segment, tappet,...) is selected, the workspace of the editor is blank and in the "Description" field in the lower part nothing is displayed.

Else, if in one of the graphical tabs of the CAM Editor (entitled "CAM" or "Tappets") an element has been selected by a click, the workspace embodies a table showing all properties related to the selected element.

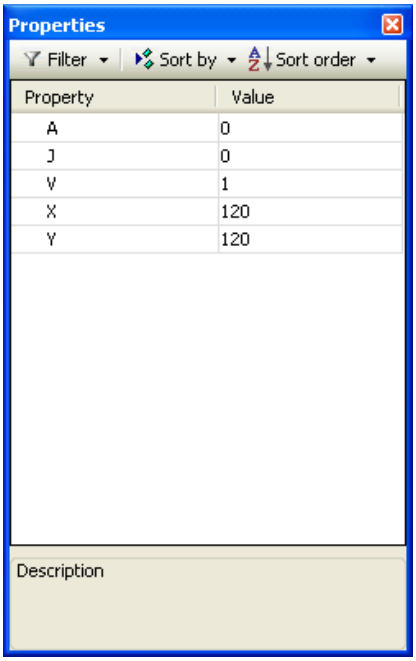


Abb. 309 Example of editor displaying properties of a selected point

The properties editor is equipped with a small tool bar heading the work space. We can omit its first item: So far there are no filter options, so that all properties related to the selected element are displayed. With means of the second item you may chose whether the properties shall be sorted by type or alphabetically. Select the desired feature by an click.

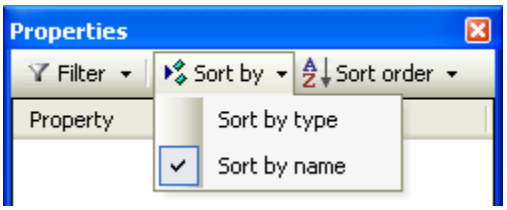


Abb. 310 Toolbar: Sort by...

The third item determines whether the sorting shall be presented in ascending or descending order. Select the desired order by a click.

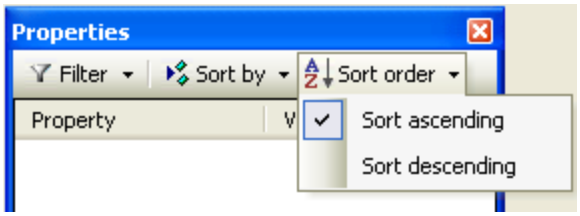


Abb. 311 Toolbar: Sort order

A table entry is **selected** by a click on it. The selection will be indicated by a thin blue frame in addition to a light blue inking of the related table row. At the same time an explanation of the related property will be displayed within the sub-field "Description".

Depending on the related property the **value** of the selected table entry may be **modified** by typing or by making an appropriate choice from the selection list provided upon a further click into the table cell. The modification will be adopted after the deselection of the table cell, that is after a click on another table entry or on blank workspace.

13.2 CFC Editor

The CFC Editor, provided by the 'CFC Editor' plug-in, is available for programming objects in the programming language **Continuous Function Chart** (CFC), which is an extension to the IEC 61131-3 programming languages. You choose the language when adding a new POU object via the 'Add Object' command in your project.

The CFC Editor is a graphical editor.

The editor will be available in the lower part of the window which opens when opening a CFC POU object and which also includes the Declaration Editor in its upper part.

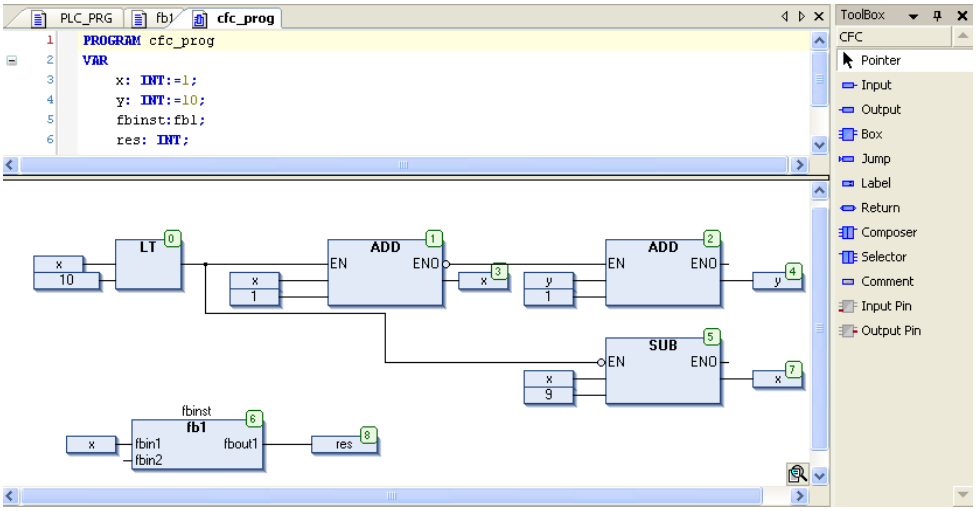


Abb. 312 CFC Editor

The CFC Editor in contrast to the network editors allows free positioning of the elements, which for example allows direct insertion of feedback paths. The sequence of processing is determined by a list which contains all currently inserted elements and can be modified.

The following elements are available in a toolbox for inserting: box (operators, functions, function blocks and programs), input, output, comment, label, jump, composer, selector.

The input and output lines of the elements can be connected by dragging a connection with the mouse. The course of the connecting line will be created automatically and noticeing the shortest possible way. The connecting lines are automatically adjusted as soon as the elements are moved. See also: Insert and arrange elements.

You may change the dimension of the editor window by zooming: Use the button in the lower right corner of the window and choose between the listed zoom factors. Alternatively you may select the entry ... to open a dialog where you can enter any arbitrary factor by typing.

The commands for working in the CFC Editor can be called from the context menu or from the "CFC" menu which is available as soon as the CFC Editor is active.

13.2.1 CFC - Continuous Function Chart - Language

The Continuous Function Chart in extension to the IEC 61131-3 standard is a graphical programming language basing on the Function Block Diagram language. However in contrast to that no networks are used but free positioning of graphic elements, which for example allows feedback loops.

For creating CFC programming objects in »PLC Designer« see: CFC Editor.

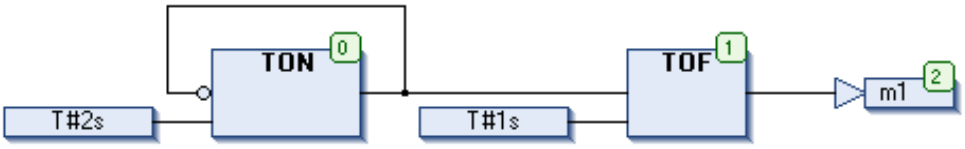


Abb. 313 Example of a CFC network

13.2.2 Cursor positions in CFC

A possible cursor position in a CFC program is indicated by default by a grey shadow when moving with the cursor over the elements.

As soon as you click on one of these shadowed areas, still before you are leaving the mouse-button, the area will change color to red. As soon as you leave the pressed mouse-button, this will become the current cursor position, the respective element or text being selected and displayed red-colored.

There are three categories of cursor positions. See the possible positions indicated by a grey shaded area as shown in the following pictures:

1. If the cursor is positioned on a **text**, this will be displayed blue-shaded and can be edited. The … button is available for opening the input assistant. Primarily after having inserted an element "???" is displayed and must be replaced by a valid identifier.

Abb. 314 Possible cursor positions and two examples of selected texts

2. If the cursor is positioned on the **body of an element** (box, input, output, jump, label, return, comment), this will be displayed red-colored and can be moved by moving the mouse.

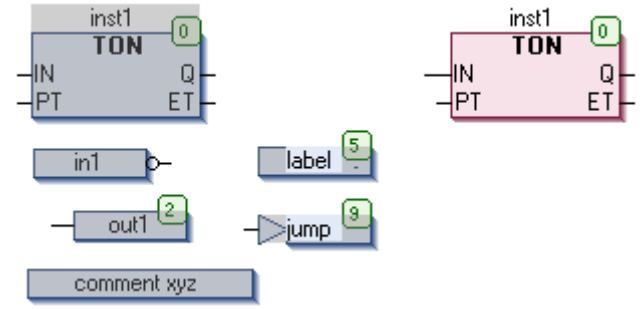


Abb. 315 Possible cursor positions and example of a selected body

If the cursor is positioned on an **input** or **output** connection of an element, this will get displayed red-colored and can be negated or set/reset.

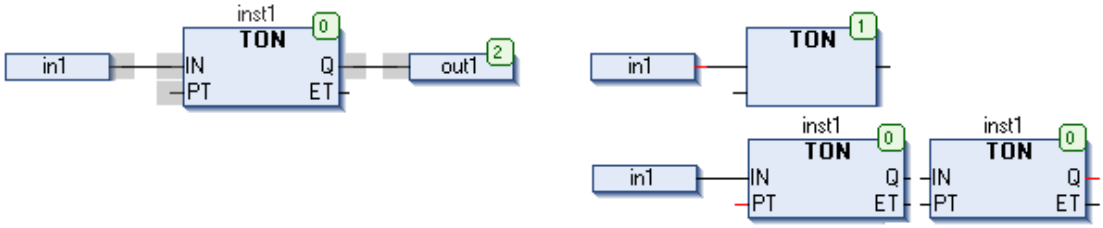


Abb. 316 Possible cursor positions and examples of selected output and input positions

13.2.3 CFC Elements / Toolbox

The graphical elements available for programming in the CFC editor window are provided by a toolbox which comes with the 'CFC Editor' plug-in. The toolbox can be opened in a view window by command 'Toolbox' which by default is in the 'View' menu.

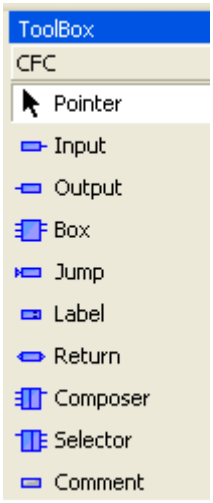
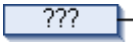
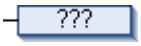
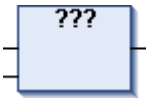
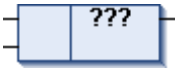


Abb. 317 CFC toolbox, default

The desired element can be selected in the toolbox and inserted in the editor window via drag&drop.

Besides the programming elements there is an entry Pointer, by default at the top of the toolbox list. As long as this entry is selected, the cursor has the shape of an arrow and you can select elements in the editor window for positioning and editing them.

The elements:

	Input		The text offered by "???" can be selected and replaced by a variable or constant. The input assistance serves to select a valid identifier.
	Output		
	Box		A box can be used to represent operators, functions, function blocks and programs. The text offered "???" can be selected and replaced by an operator, function, function block or program name. The input assistance serves to select one of the available objects. In case a function block is inserted, another "???" will be displayed above the box and have to be replaced by the name of the function block instance. If an existing box is replaced by another one (by modifying the entered name) and the new one has a different minimum or maximum number of input or output pins, the pins will be adapted correspondingly. If pins are to be removed, the lowest one will be removed first.
	Jump		The jump element is used to indicate at which position the execution of the program should continue. This position is defined by a label (see below). So replace the text offered by "???" by the label name.
	Label		A label marks the position to which the program can jump (see above 'Jump') In online mode a RETURN label for marking the end of POU is automatically inserted.
	Return		Note that in Online mode a RETURN element is automatically inserted in the first column and after the last element in the editor. In stepping it is automatically jumped to before execution leaves the POU.
	Composer		A composer is used to handle an input of a box which is of type of a structure. The composer will display the structure components and thus make them accessible in the CFC for the programmer. For this purpose name the composer like the concerned structure (by replacing "???" by the name) and connect it to the box instead of using an "input" element .

Example

A CFC program *cfc_prog* handles an instance of function block *fubblo1*, which has an input variable *struvar* of type of a structure. By using the composer element the structure components can be accessed:

Structure *stru1* definition:

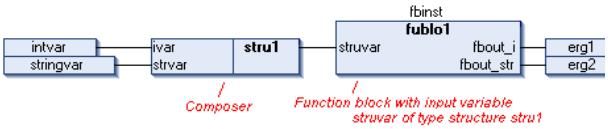
```
TYPE stru1 :
STRUCT
 ivar:INT;
 strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Function block *fublo1*, declaration and implementation:

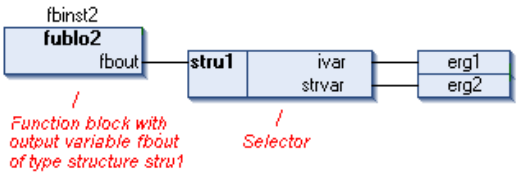
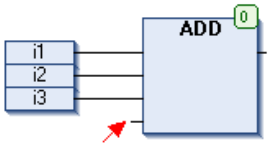
```
FUNCTION_BLOCK fublo1
VAR_INPUT
 struvar:STRU1;
END_VAR
VAR_OUTPUT
 fbout_i:INT;
 fbout_str:STRING;
END_VAR
VAR
 fbvar:STRING:='world';
END_VAR

fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

Program *cfc_prog*, declaration and implementation:

```
PROGRAM cfc_prog
VAR
 intvar: INT;
 stringvar: STRING;
 fbinst: fublo1;
 erg1: INT;
 erg2: STRING;
END_VAR
```



	Selector		A selector in contrast to the composer is used to handle an output of a box which is of type of a structure. The selector will display the structure components and thus make them accessible in the CFC for the programmer. For this purpose name the selector like the concerned structure (by replacing "???" by the name) and connect it to the box instead of using an "output" element . Example A CFC program *cfc_prog* handles an instance of function block *fubblo2*, which has an output variable *fbout* of type of a structure *stru1*. By using the selector element the structure components can be accessed: Structure *stru1* definition: TYPE stru1 : STRUCT ivar:INT; strvar:STRING:='hallo'; END_STRUCT END_TYPE Function block *fublo1*, declaration and implementation: FUNCTION_BLOCK fublo2 VAR_INPUT CONSTANT fbin1:INT; fbin2:DWORD:=24354333; fbin3:STRING:='hallo'; END_VAR VAR_INPUT fbin : INT; END_VAR VAR_OUTPUT fbout : stru1; fbout2:DWORD; END_VAR VAR fbvar:INT; fbvar2:STRING; END_VAR

			Program *cfc_prog*, declaration and implementation: VAR intvar: INT; stringvar: STRING; fbinst: fublo1; erg1: INT; erg2: STRING; fbinst2: fublo2; END_VAR 
	Comment	<Enter your comment here...>	Use this element to add any comments to the chart. Select the placeholder text and replace it with any desired text. You obtain a new line within the comment with <Ctrl> + <Enter>.
	Input Pin		Depending on the box type an additional input might be added. For this purpose select the box element in the CFC network and draw the Input Pin element on the box.
	Output Pin		Depending on the box type an additional output might be added. For this purpose select the box element in the CFC network and draw the Output Pin element on the box.

13.2.4 Insert and arrange elements

The elements available for programming in the CFC Editor are provided in a toolbox which by default is available in a window as soon as the CFC editor is opened.

The CFC Editor options define general settings for the working within the editor.

Inserting

To insert an element select it in the toolbox by a mouse-click, keep the mouse-button pressed and draw the element to the desired position in the editor window. During drawing the cursor will be displayed as an arrow plus an rectangle and a plus-sign. When you leave the mouse-button the element will be inserted.

Selecting

To select an inserted element for further actions like editing or rearranging notice the possible cursor positions for element bodies, in- and outputs and text By a mouse-click on a elements body the element gets selected and will be displayed by default red-shaded now. By additionally keeping pressed the <Shift>-key you can click on and thereby select further elements. You also can press the left mouse-button and draw a dotted rectangle around all elements which should be selected. As soon as you leave the button the selection will be indicated. By command Select all, which by default is available in the context menu, all elements are selected at once.
By using the arrow keys the selection mark can be shifted to the next possible cursor position. The sequence depends on the execution order or the elements, which is indicated by element numbers, see below.

When an input pin is selected and <Ctrl>+<Leftarrow> is pressed, the corresponding output will be selected. When an output pin is selected and <Ctrl>+<Leftarrow> is pressed, the corresponding output(s) will be selected.

Replacing boxes

To replace an existing box element, replace the currently inserted identifier by that of the desired new element. Notice that the number of input and output pins will be adapted if necessary due to the definition of the POUs and thus some existing assignments might be removed.

Moving

To move an element, select the element by a mouse-click on the element body (see possible cursor positions) and drag it, while keeping the mouse-button pressed, to the desired position. Then leave the mouse-button to place the element. You also can use the Cut and Paste commands for this purpose.

Connecting

The connections between the inputs and outputs of elements can be drawn with the mouse. The shortest possible connection will be created taking into account the other elements and connections. If the course of connection lines is painted light-grey-colored, this might indicate that elements are positioned covering each other.

Copying

To copy an element, select it and use the Copy and Paste commands.

Editing

After inserting an element, by default the text part is represented by "???". To replace this by the desired text (POU name, Label name, Instance name, comment etc.) select the text by a mouse-click to get an edit field. Also button ... will be available then to open the Input Assistant.

Deleting

A selected element can be deleted by command Delete, which is available in the context menu, or by the <Del> key.

Execution order, element numbers

The sequence in which the elements in a CFC network are executed in online mode is indicated by numbers in the upper right corner of the box, output, jump, return and label elements. The processing starts at the element with the lowest number, which is "0". The execution order can be modified by commands which are by default available in submenu "Execution Order" in the CFC menu.

When adding an element, the number will automatically be given according to the topological sequence (from left to right and from above to below). The new element receives the number of its topological successor if the sequence has already been changed, and all higher numbers are increased by one.

Notice that the number of an element remains constant when it is moved.

Consider that the sequence influences the result and must be changed in certain cases.

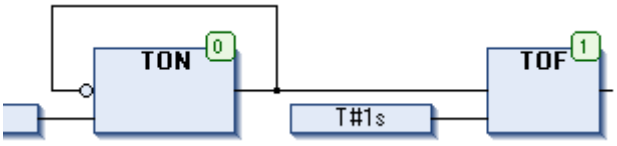


Abb. 318 Example of element numbers

Changing size of the working sheet

In order to get more space around an existing CFC chart in the editor window, the working area ("working sheet") size can be changed. This might be done by selecting and dragging all elements with the mouse or by the Cut and Paste commands (see above, Moving).

Alternatively a special dimensions settings dialog can be used, which might save time in the case of big charts. See Edit Working Sheet for a description.

13.2.5 CFC Editor in online mode

In online mode the CFC-Editor provides views for monitoring and for writing and forcing the variables and expressions on the controller. Debugging functionality (breakpoints, stepping etc.) is available. See below.

- For information on how to open objects in online mode see User Interface in online mode.
- Notice that the editor window of an CFC object also includes the Declaration Editor in the upper part. For information on the Declaration Editor in online mode see Declaration Editor in online mode.

Monitoring

The actual values are displayed in small monitoring windows behind each variable (**Inline Monitoring**).

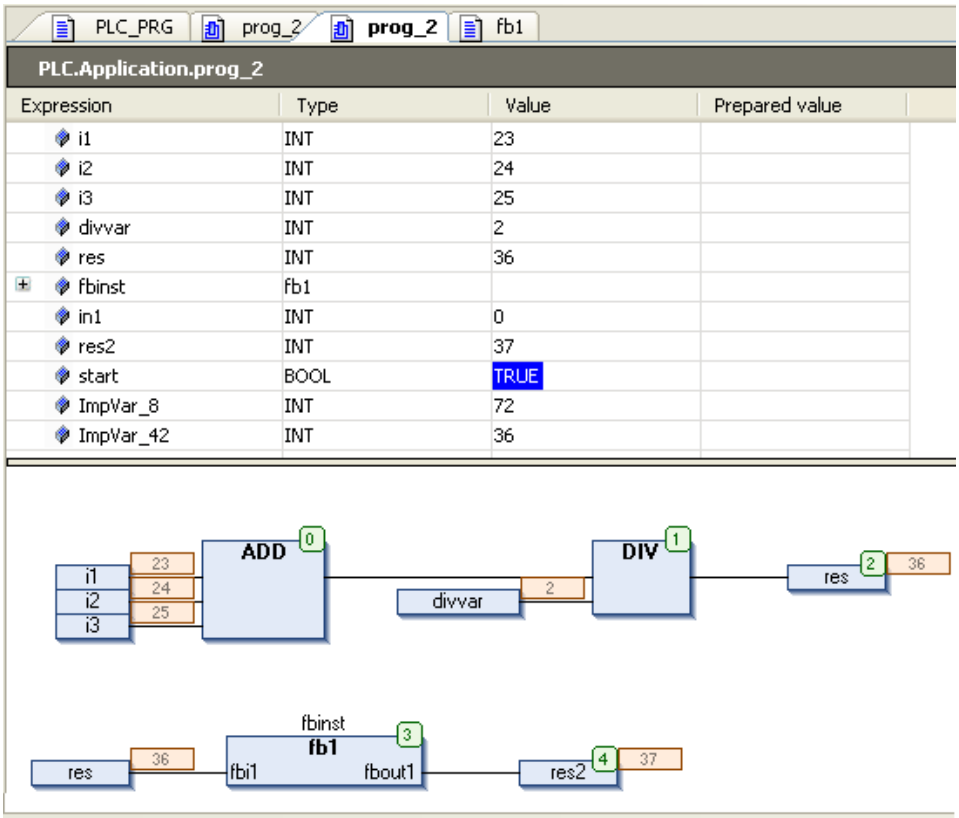


Abb. 319 Example: Online view of a program object PLC_PRG

Notice that for the online view of a function block POU: In the implementation part no values will be viewed in the monitoring windows but "<Value of the expression>" and the inline monitoring fields in the implementation part will show three question marks each.

Breakpoint positions in CFC Editor:

The possible breakpoint positions basically are those positions in a POU at which values of variables can change or at which the program flow branches out resp. another POU is called. See possible positions in the following picture.

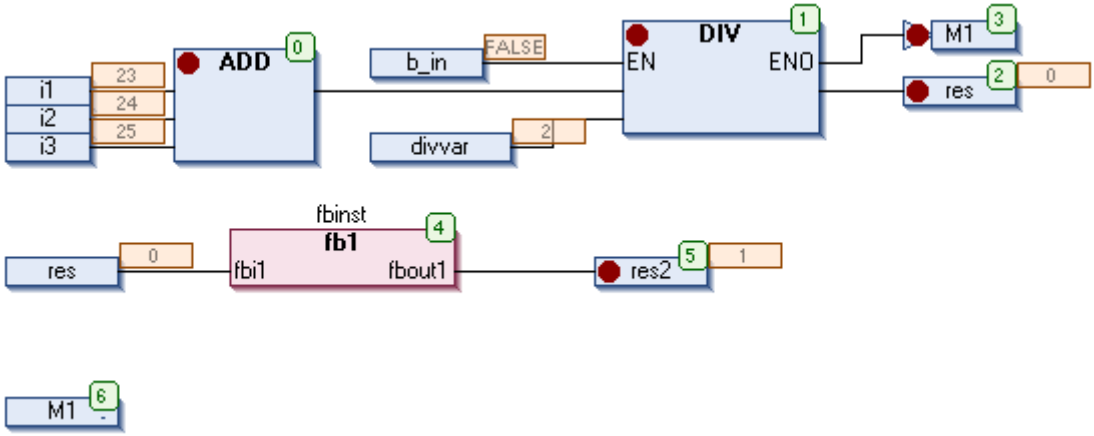


Abb. 320 Breakpoint positions in CFC Editor

Notice for breakpoints in methods: A breakpoint will be set automatically in all methods which might be called. So, if an interface-managed method is called, breakpoints will be set in all methods of function blocks implementing that interface and also in all derivative function blocks subscribing the method. If a method is called via a pointer on a function block, breakpoints will be set in the method of the function block and in all derivative function blocks which are subscribing the method.

13.3 CNC Editor

The SoftMotion Din66025 CNC editor is integrated in the »PLC Designer« programming interface. Within the CNC Editor complex multi-axis movements can be implemented via graphical and textual or tabular editing. The properties of the path elements are viewed in a window as well as the toolbox allows a simple graphical programing of the CNC path.

See in the following on this page:

- Overview
- Creation of a CNC program
- Edit CNC program
- Compilation of the CNC
- Open an existing project

Note: By using the **F1** key, the online help page of the active dialog or editor opens directly.

Overview

The CNC editor in »PLC Designer« allows to program multi-dimensional motions graphically and textually in parallel following the CNC language DIN66025. Additionally a tabular editor can be used in place of the text editor.

Have a closer look to the programming examples provided with the setup.

With Din66025 basically up to 9-dimensional motions can be realized, whereby up to 6 dimensions are not interpolated linearly. In three dimensions lines, arcs, elliptic arcs, parables and splines can be programmed; there are 8 further so called additional axes, five of which are interpolated linearly, the remaining three are interpolated using a spline of degree 3. For each programmed path »PLC Designer« automatically creates a global data structure (CNC Data), which can be used by the IEC program.

This can be done in different ways:

1. The CNC program is stored as an array of G-codes and can be moved, scaled or rotated with program transformation and will be decoded during runtime of the PLC program with the aid of a decoder module. Thus for the particular path objects GEOINFO structure objects will be available. These objects can be edited with the aid of path preprocessing modules like tool radius correction provided in SM3_CNC library, afterwards interpolated, transformed and returned to the drive interface for the communication with the hardware.
2. The CNC program is written as a list of GEOINFO structure objects named SMC_OUTQUEUE to a data structure and thus can directly be fed to the interpolator. In comparison to 1. by this method you can avoid calling the decoder and the path preprocessing modules. But in exchange you do not have the possibility to change the program during runtime. In particular variables cannot be used in G-code in this mode.
3. The CNC program is written as described in 1. to the file system of the controller and is read and executed at runtime step by step. This method particularly is suitable for big programs, which cannot be kept completely in the memory.

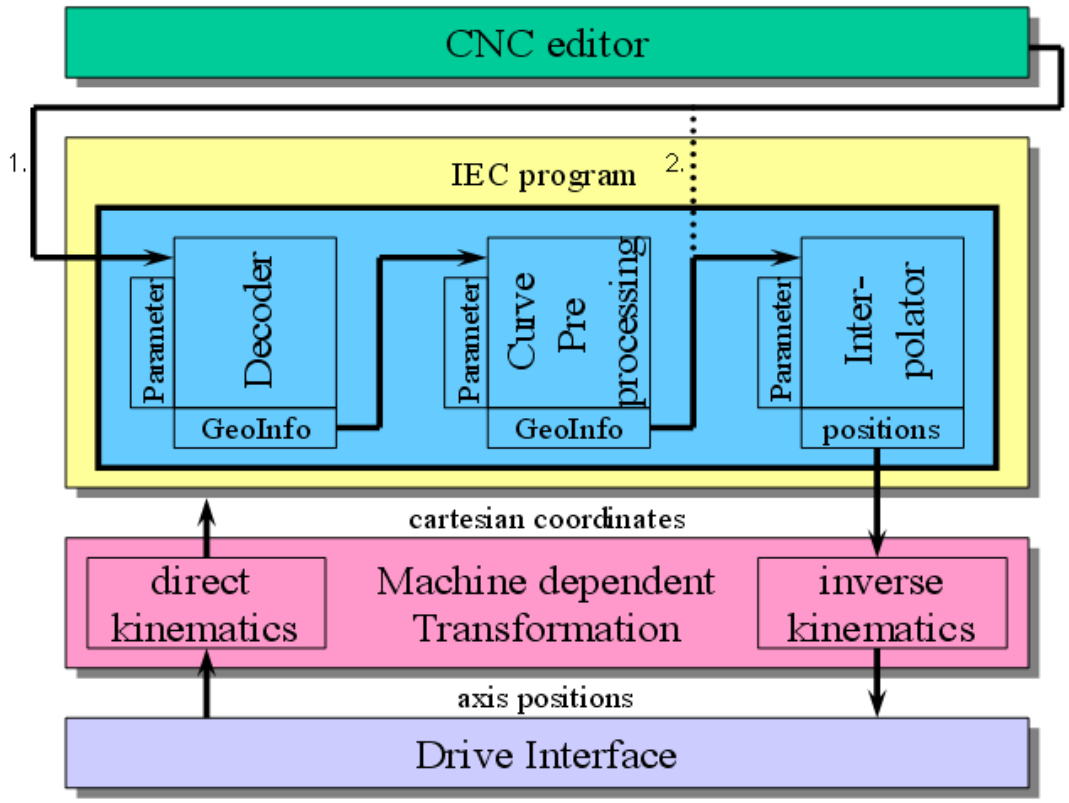


Abb. 321 The SoftMotion software parts of the CNC editor

Creation of a CNC program

A CNC program is an object that has to be added to an application. Therefore select an application and activate the entry **Add object...** of the context menu, whereon the "Add object"-dialog will open.

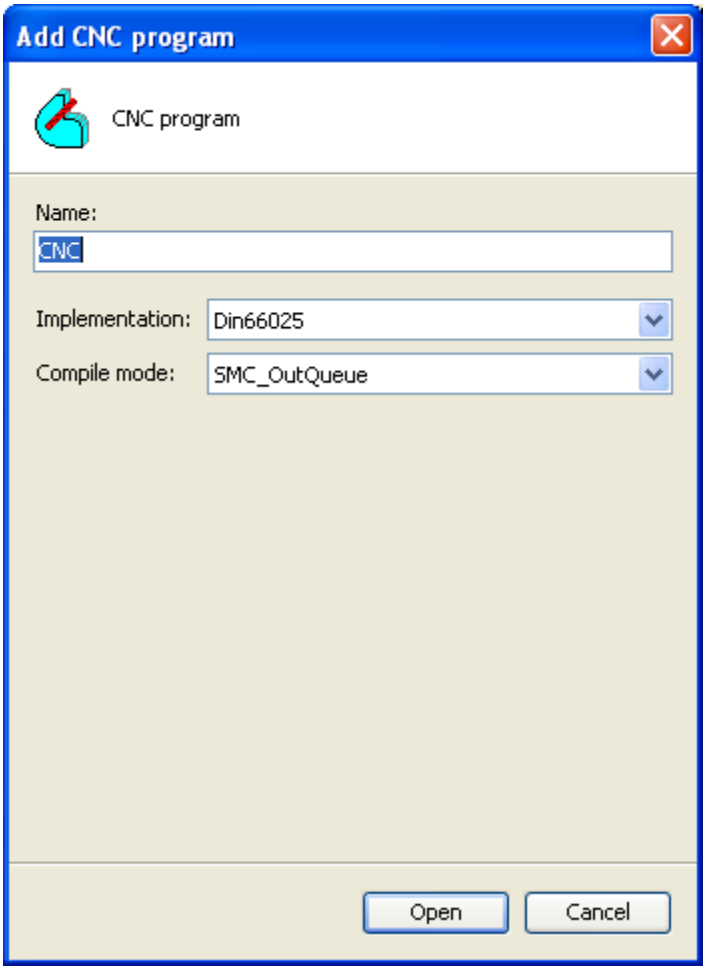


Abb. 322 Add CNC program dialog

Therein enter a name, an implementation and an compile mode ("CNC", "Din66025" and "SMC_OutQueue" by default) and confirm with **Open**. The CNC object is inserted in the device tree and the CNC editor opens displaying the empty view of the new CNC object.

Edit a CNC object

The CNC editor consists of four parts, whereof the first three are opened automatically when creating a CNC object: the main editor, the toolbox, the path element properties window and the CNC object properties window.

Main editor

The main editor is entitled with the name of the CNC program followed by its path in the device tree (for example CNC [Device:PLC Logic:Application]). This window is separated in an upper part where the DIN66025 program is listed in the text editor or the CNC path is in table view visualized. In the bottom part of the window a graphical view of the motion is shown. Selecting a path element in the path view editor implicates a selected sentence resp. row in the upper editor. The sentence in the text editor is highlighted in a **light blue** inking. The row in the table editor is highlighted in a **light gray** inking. Modifications in the graphic editor will be taken over to the text or table editor and vice versa.

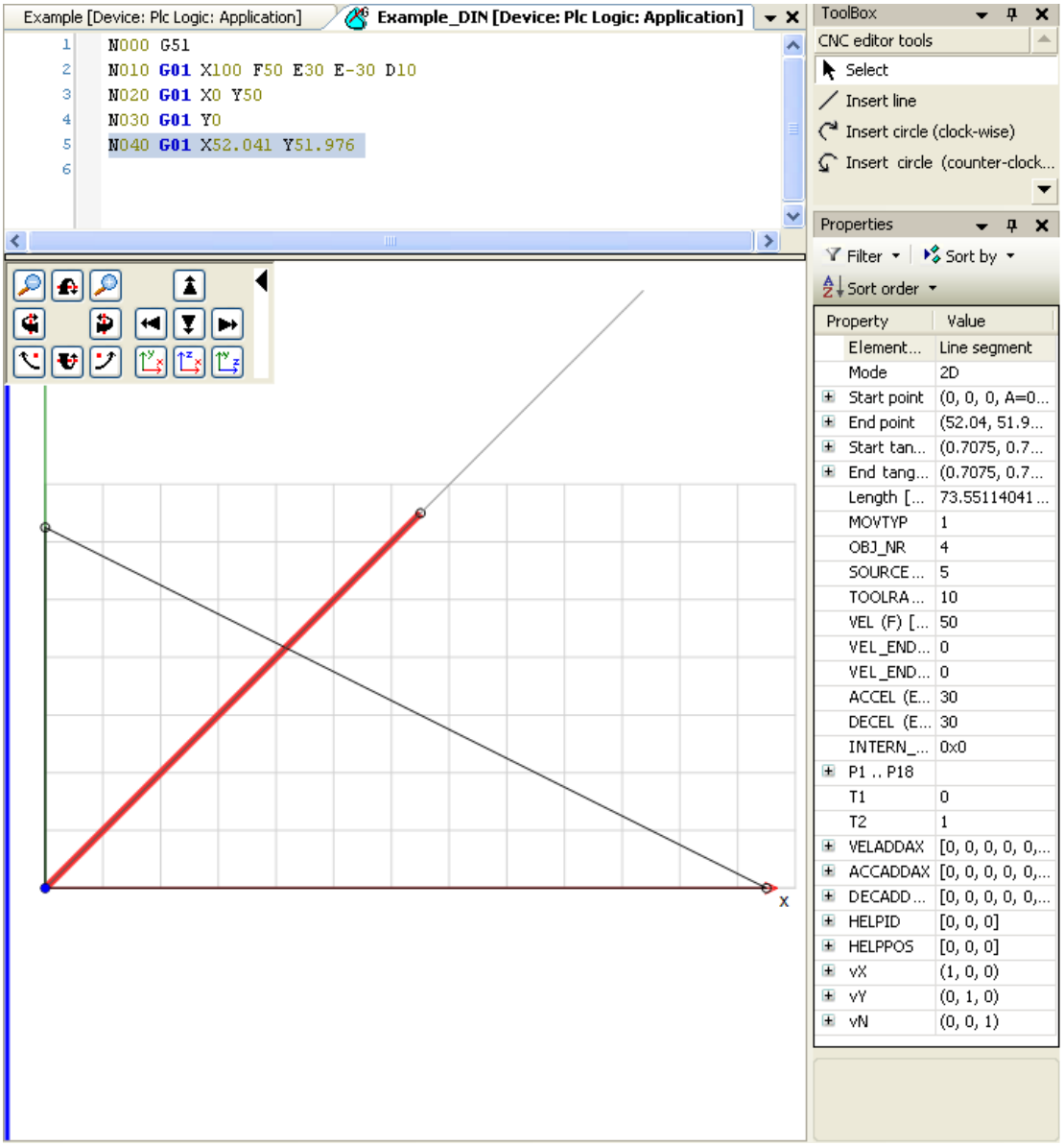


Abb. 323 CNC program in DIN66250

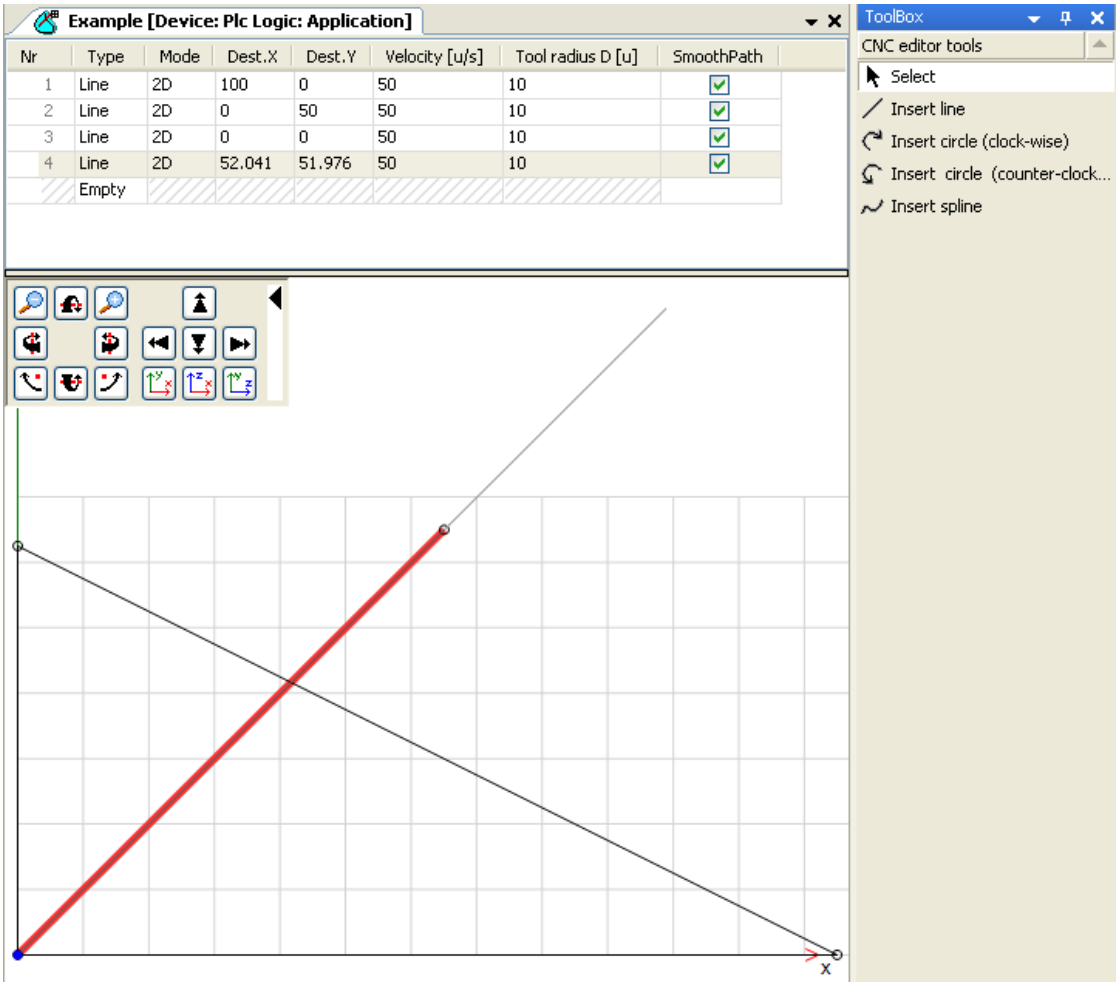


Abb. 324 CNC program in table view

A CNC program can be transformed from table view to text editor and vice versa.

Toolbox

The ToolBox is providing tools for selecting and inserting path elements.

The path element properties window

Properties

Filter | Sort by | Sort order

Property	Value
Element type	Ellipse (counter clockwise)
Mode	2D
Start point	(112.9, 85.11, 0, A=0, B=0, C=0, P=0, Q=0, U=0, V=0, W=0, AUXDATA=0x7, SPROFILE=0x0)
End point	(212, 71.19, 0, A=0, B=0, C=0, P=0, Q=0, U=0, V=0, W=0, AUXDATA=0x7, SPROFILE=0x0)
Start tangent	(0.6288, 0.7775, 0)
End tangent	(-0.6288, -0.7775, 0)
Length [units]	191.48477105960015
Apex (T2) [deg]	-179.99999999999997
Center	(162.5, 78.15, 0)
Pitch (K) [deg]	397.00000000000006
Ratio (R)	0.5
MOVTYP	8
OBJ_NR	2
SOURCELINE_NR	3
TOOLRADIUS (D) [u]	0
VEL (F) [u/s]	100
VEL_END [u/s]	0
VEL_END_STORED [u/s]	0
ACCEL (E+) [u/s²]	100
DECEL (E-) [u/s²]	100
INTERN_MARK	0x0
P1 .. P18	
T1	116.56505117707795
T2	-179.99999999999997
VELADDAX	[0, 0, 0, 0, 0, 0, 0, 0]
ACCADDAX	[0, 0, 0, 0, 0, 0, 0, 0]
DECADDAX	[0, 0, 0, 0, 0, 0, 0, 0]
HELPID	[0, 0, 0]
HELPPOS	[0, 0, 0]
vX	(1, 0, 0)
vY	(0, 1, 0)
vN	(0, 0, 1)

Abb. 325 CNC path element properties window

The properties of a path element are displayed in the "Properties" window, which by default is available next to the toolbox window. Beside that it can be opened by a click on the entry **Element properties** of menu item **View**. Then all properties of the selected path element are listed.

CNC object properties window

The CNC object properties editor is containing general information on the CNC. It is opened either via the entry **Properties...** within the context menu of the CNC object in the device window or via the command **Properties...** () within the category **View**.

Compilation of the CNC

During compilation structure variables of type SMC_OutQueue, SMC_CNC_REF (set with the drop-down list Compile mode) are created independent of using the text editor or the table view for generating code. These structures are part of the library SM3_CNC.The data structures are listed in a global variable and allow an easy access. See CNC data structures.

A further option is to create a file including the programed CNC path during compilation and load it on the PLC while doing a login. For this use the option File in the compile mode list must be selected.

For a further description see 'compile mode'.

Open an existing project file

To open an existing project file a standard dialog opens. A further description can be found in 'Open Project...'. If it is a »PLC Designer« V2.3 project, an appropriate converter is available.

The following tasks are taken from the converter, if a SoftMotion capable device is available:

- The converter transforms every CNC program into CNC objects with same name and implementation type DIN 66025. There the compile mode, the queue size and the start position of the converting program are stored just as the status of step suppression.
- If in a CNC program in a »PLC Designer« V2.3 project the option Exclude from build is activated, the appropriate flag Exclude from build in »PLC Designer« V3 is also set.
- The previously used values epsilon and angle leeway are ignored.
- A 'CNC Settings' object is automatically generated.
- All CNC objects and the CNC settings object are sorted below an application which is capable of doing »PLC Designer« SoftMotion.
- If the device is not capable to SoftMotion, no CNC program is transformed and an error message is generated.

13.3.1 Text Editor

CNC Text Editor

The text editor is in the upper part of the 'CNC' window. Here you can enter and modify a CNC program according to DIN66025. The syntax and supported path elements of Din66025 are displayed accordingly in the graphic editor and can be modified there. In this case the text part will be updated vice versa.

General

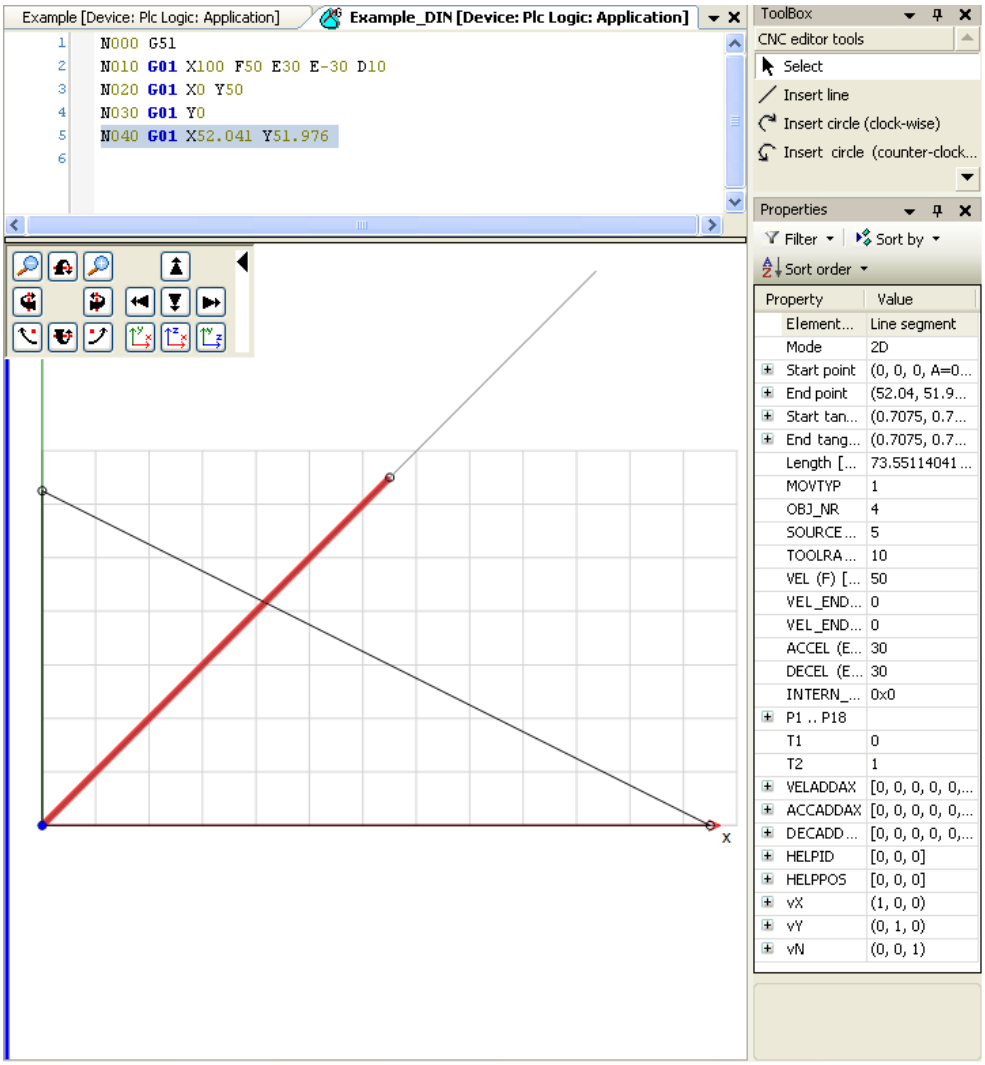
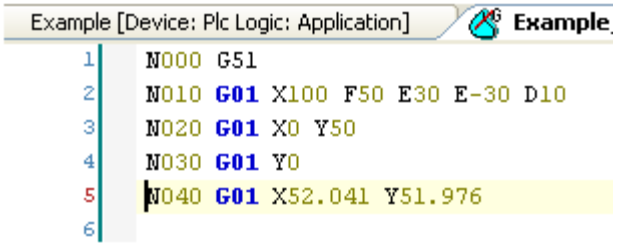


Abb. 326 CNC program in DIN66250

By click in the text editor the sentence is highlighted in **light yellow** and the associated path element becomes **red** with a **blue** start point and a black bordered end point. When a path element in the graphic path diagram is selected the associated sentence is highlighted in in a **light blue** inking.



To add a sentence

For inserting or appending a path element generate a new line and program there the path element in conformance with DIN66025.

To modify a sentence

All sentences can be changed by editing them directly in the text editor. Then the changed element is marked in the graphic editor as usual. By pressing <F2> the input assistant will open and you can select global variables to be inserted in the CNC program. If you have chosen compile mode SMC_OutQueue, which makes »PLC Designer« creating a ready OutQueue-structure during compilation, the reference on the global variable of course will be replaced by the initial value already during compilation which makes the use of global variables senseless in this context.

To delete a sentence

A selected sentence can be deleted by using the **Del** key.

Commands and options

Various commands and options are available for visualize additional sets of data of the CNC program or manipulate it. They are all placed in the "CNC" menu.

Note: Please regard, that references to global variables in the decoder module will be evaluated when the decoder process the lines. This may be some cycles in advance to passing this object.

Example

Various CNC programs are enclosed by the setup. Have a closer look at the comprehensive description of them.

Din66025

CNC Language DIN66025

In order to provide an easy way for programming geometric motion profiles SoftMotion supports **parts** of the CNC language DIN66025. Since the whole SoftMotion concept is embedded in the much more powerful language IEC61131, only those parts of DIN66025 are supported, which serve to create a path.

Prescribed structure of a CNC program:

```
N<number> G<number> ...
...
```

Example:

```
% MyExample
N10 G01 X100 Y100 E100 F100 E-200
N20 G01 Z40 F20
N30 G03 X-100 R200 F100
...
```

Syntax

A SoftMotion CNC program can start with a "%". In the same line optionally can be added – separated by an empty space or by a TAB – the program name. The actual CNC program is composed of several **sentences**.

Each sentence consists of as many words as needed.

A word consists of a letter (word identifier) and a subsequent number (e.g. G01; see also the list below). There is no case sensitivity and leading zeros are ignored (G01 = g1).

The first word of each sentence is built by the **sentence number** (N<number>), e.g. "N01". The words of a sentence are separated one or more space or TAB character. They are processed from the right to the left. Thereby all words except for the positioning commands (G<number>, e.g. "G02"; see the list below), will effect that the sentence number will be assigned to a variable as defined by the sentence letter. This variable finally will be accessed by the positioning command.

Each sentence only may contain one command, which must follow right to the sentence number. If you do not enter an positioning command in a sentence, automatically that of the last sentence will be added.

Each positioning command can be seen as a path object (line, arc, ...). The velocity at which the path objects are interpolated, basically complies with the scheduled velocity (command speed), - acceleration and - deceleration. The interpolator must make sure that these limits are not exceeded. The velocity during the transition of two adjacent objects is determined according to the following rules:

- One of both objects is a positioning (G0): Transition velocity = 0
- The angle between the tangents of the two objects at the transition is bigger than the angle tolerance: Transition velocity = 0
- Otherwise: The transition velocity is the lower command speed of the both path objects.
- Basically a position command effects that there will be an interpolation from the target position of the last positioning command to the target position specified by the current positioning command. The first positioning command starts at the specified position (specified in the Decoder or CNC-Editor). If that position has not been defined, it will start at X=0, Y=0, Z=0.

Additionally there is the possibility to set the position in the CNC program via G92. This is allowed at the beginning of the CNC program (there the start position is set) as well as in the middle, where it will result in a jump of the target position to the position defined by G92. If there are several successive G92-commands, the last will be regarded; the preceding ones will be skipped. If you however want to make sure that also the preceding G92 positions are given out (for the length of one cycle), you must insert the command G1 with identical coordinates between. This means is used if the path between those positions is not of interest, but the target position should get there as fast as possible. The modules SMC_ControlAxisByPos in this case detect a jump of the target positions, stop the interpolator and interpolate each axis separately on the fastest way to the target position.

Example:

```
%MyExample
N10 G92 X100 Y100 (Set target position to 100/100)
N20 G1 X100 Y100 (Make sure a one-time output of the position)
N20 G92 X50 Y100 (Set target position to 50/100)
```

A sentence starting wit the character "/" will be skipped during processing, if the option step suppression in the CNC menu is activated .

Characters which are embraced by parenthesis "( )", will be interpreted as comments and do not have any effect on the programmed path. Nested comments are not supported.

The sentence number (N<number>) is need as a marker and for defining jump targets, but has no further meaning.

All numbers except for that of the command (G<number> or M<number>) and the switch number (H<number>) can be floating values.

See also:

- Positioning
- Switch functionality (H-function)
- Additional function (M-function)
- Usage of variables
- Circular interpolation
- Ellipse interpolation
- Spline interpolation
- Conditional jumps
- Change variable values

Word identifiers :

Letter	Function
A	Target position of additional spline axis A.
B	Target position of additional spline axis B.
C	Target position of additional spline axis C.
D	Tool radius (for correction G40-42 resp. for 'Round of Path' G50-51) resp. variable value (G36/G37)
E	Max. acceleration (>0) / deceleration (<0) [path-units/sec^2]
F(feed)	Velocity [path-units/sec]
G (geometry)	Positioning command
H	Switch on switch point (>0) / switch off (<0)
I	X-coordinate of the circle/ellipse center (G02/G03/G08/G09) or X-coordinate of the parabel-tangent-intersection-point
J	Y-coordinate of the circle/ellipse center (G02/G03/G08/G09) or Y-coordinate of the parabel-tangent-intersection-point
K	Direction of the ellipse main axis in mathematical sense (0° O, 90° N,...) resp. jump condition (G20) resp. dT1 parameter value (M-function) or Z-coordinate of the circle center (G02/G03 in 3D mode only)
L	Absolute switch position "H"), measured from start position (>0) resp. end position (<0) of the path object, resp. jump condition (G20) resp. dT2 parameter value (M-function)
M (machine)	Additional option, M-option (M-function)
O	Relative switch ("H") position [0..1] resp. the variable to be changed (G36/G37) or M-parameter data structure (M)
P	Target position of the additional linear axis P

Letter	Function
Q	Target position of the additional linear axis Q
R	Radius (G02/G03) – alternatively to "I","J" or ratio of the minor to the major elliptical axis (G08/G09) [0..1]
S	Switch on (>0) / off (<0) the S-profile for linear axes 3: Z-axis (2D-mode only), 7: P-axis, 8: Q-axis, 9: U-axis, 10: V-axis, 11: W-axis
T	Command specific parameter.
U	Target position of the additional linear axis U
V	Target position of the additional linear axis V
W	Target position of the additional linear axis W
X	X-coordinate of the target position
Y	Y-coordinate of the target position
Z	Z-coordinate of the target position

Instructions :

Letter	Function
G00	Direct movement without tool contact, positioning
G01	Linear (straight) movement with tool contact
G02	Circle(-segment) clockwise
G03	Circle(-segment) counterclockwise
G04	Delay Time
G05	Point of a 2D cardinal spline
G06	Parable
G08	Ellipsis(-segment) clockwise
G09	Ellipsis(-segment) counterclockwise
G10	Point of a 3D cardinal spline
G15	Change to 2D.
G16	Change to 3D by activating the 3D mode with plane normal I/J/K.
G17	Change to 3D by activating the 3D mode in X/Y plane.
G18	Change to 3D by activating the 3D mode in Z/X plane.
G19	Change to 3D by activating the 3D mode in Y/Z plane.
G20	Conditional jump (to L if K<>0)
G36	Write value (D) to variable (O)
G37	Increment variable (O) by value (D)
G40	End of the tool radius correction
G41	Start the tool radius correction to the left of the workpiece

Letter	Function
G42	Start the tool radius correction to the right of the workpiece
G50	End of RoundPath/SmoothPath function
G51	Start the SmoothPath function
G52	Start the RoundPath function
G53	End the shift coordinate function.
G54	Set offset of all following coordinates to the specified position
G55	Add the specified position to the current offset.
G56	Set the offset such that the current position equals the specified position
G60	End of the avoid-loop function
G61	Start of the avoid-loop function
G75	Time synchronisation with interpolator
G90	Start interpreting the following coordinate values (for X/Y/Z/P-W/A/B/C) as absolute values (default)
G91	Start interpreting the following coordinate values (for X/Y/Z/P-W/A/B/C) as relative values
G92	Setting the position without move
G98	Start interpreting the following coordinate values of I/J/K as absolute values
G99	Start interpreting the following coordinate values of I/J/K as values relative to the starting point (standard)

Please regard that the library "SM3_CNC " must be included to enable an error-free compilation of the project.

Positioning

Command G0, G92

These commands are used for changing interpolator positions.

In case of G0 the changing is done by moving on a line to the next position. In contrastto G1 the G0 elements aren't modulated with path-preprocessors like tool radius, smooth path or round path correction.

With G92 the position of the interpolator is set by jump. The consequence is, that these jumps are executed on the axes, except **when programming features like** MC_ControlAxisByPos.bAvoidGaps:=TRUE prevent this. „smc_controlaxisbypos“. If G92 is applied in a CNC programm before the frirst positioning element, then it can have the designed use of setting the start position.

Allowed syntax

G0 X Y Z A B C P Q U V W F E H L/O S

G92 X Y Z A B C P Q U V W H L/O

Parameters of the positioning commands	
X Y Z	Target position of the Cartesian axes
A B C P Q U V W	Target positions of the additional axces
F E	Path velocity, acceleration, deceleration G-Code word identifiers (character address)
H L/O	Switch point (H-function)
D	Tool radius
S	S profile G-Code word identifiers (character address)

Example

Positioning with line interpolation with defined feed.

G-Code

```
N0 G92 X10 Y10 (Set the position to 10/10)
N10 G1 X20 F10 (Line interpolation to 20/10 with feed rate 10)
N20 G0 Y20 F100 (Line interpolation to 20/20 with feed rate 100)
```

Example

Various positioning modes.

G-Code

```
N0 G92 X100 Y100 F100 (Set the position to 100/100)
N10 G1 X100 Y100 (Make sure the output of this position for one cycle)
N20 G92 X50 Y100 (Set the position to 50/100 by jump)
```

Line

Command G1

The line interpolation effects, that the interpolation point from the current position to the defined target coordinate is moving on a line. All axes reach the target position simultaneously. For this, the target coordinates (X/Y/Z) must be defined.

Additionally to the movement in the space, target positions for the additional axes A/B/C/P/Q/U/V/W can be given, which are reached simultaneously with the Cartesian target positions.

The line object can be inserted with help of the „ToolBox“.

Allowed syntax

G1 X Y Z A B C P Q U V W F E H L/O D S

Parameters of the positioning commands	
X Y Z	Parameters of the positioning commands
A B C P Q U V W	Target position of the additional axes
F E	Path velocity, acceleration, deceleration G-Code word identifiers (character address)
H L/O	Switch point (H-function)
D	Tool radius
S	S profile G-Code word identifiers (character address)

Graphical representation

In the graphic editor the position command is represented by a black line with black bordered small circles for start and end points.

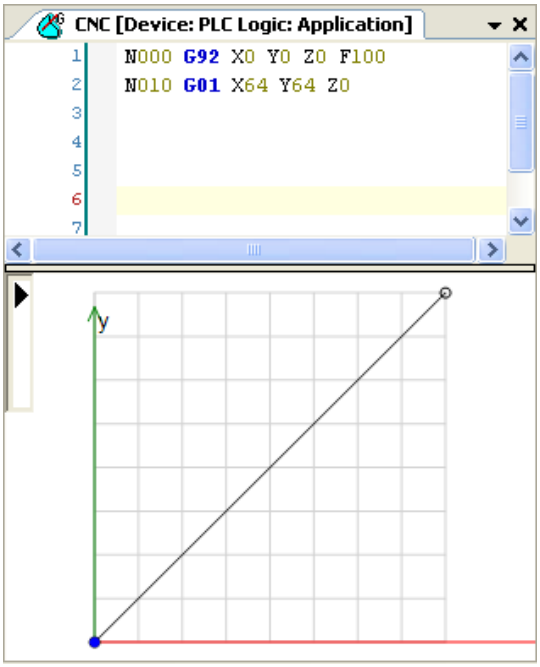


Abb. 327 CNC element line

Example:

```
N000 G92 X0 Y0 Z0 F100
N010 G01 X64 Y64 Z0
```

Example

Line from start position (default 0/0/0) to 1/2/3 with velocity 1, acceleration 10, deceleration 20. Simultaneously the orientation axis A is moved to 180 degree.

G-Code

```
N010 G01 X1 Y2 Z3 A180 F1 E10 E-20
```

Example

Start position is set to 50/50/50. Then the line interpolation with feed 100 is done from 50/50/50 to 64/30/0.

G-Code

```
N000 G92 X50 Y50 Z50 F100 (Start position to 50/50/50)
N010 G01 X64 Y30 Z0 (Line interpolation from 50/50/50 to 64/30/0, feed 100)
```

Arc

Command G2, G3

The arc interpolation effects, that the interpolation point moves from the current position along the cricular path to the target pos**ition. the command G2 effects a** movement in clockwise, the command G3 in counter-clockwise direction.

For the description of a circle in DIN66025 the target coordinates (X/Y/Z) must be defined. The bending can be defined in two ways: either you enter the radius (R) of the arc or the center coordinates (I/J/K).

Additional you have the facility to define the circular arc via center coordinates or aperture angle. Then the target **position is set automatically.**

Please regard, that not executable requirements (e.g. the radius is shorter than the half distance between start and target position, or the defined center point has got different distances to start and target position) effects, that a line is moved instead of a circle.

Allowed syntax

G2 X Y Z R A B C P Q U V W F E H L/O D S

G2 X Y Z I J K A B C P Q U V W F E H L/O D S

G2 I J K T A B C P Q U V W F E H L/O D S

G3 X Y Z R A B C P Q U V W F E H L/O D S

G3 X Y Z I J K A B C P Q U V W F E H L/O D S

G3 I J K T A B C P Q U V W F E H L/O D S

Parameters	
X Y Z	Target position of the Cartesian axes
R	Circular radius
I J K	Coordinates of the circular center
T	Aperture angle in degree
A B C P Q U V W	Target positions of the additional axces
F E	Path velocity, acceleration, deceleration
H L/O	Switch point (H-function)
D	Tool radius
S	S profile

Graphical representation

In the graphic editor the arc is visualized black colored with a black circle for start and end point. If the path element is selected it is displayed red colored with a blue start point and a black bordered end point. Additionally a light gray colored circle in the middle of the arc is visualize, if the radius is relocatable. It is possible to change the path element by clicking the points and move it by pressed right mouse key on the desired position.

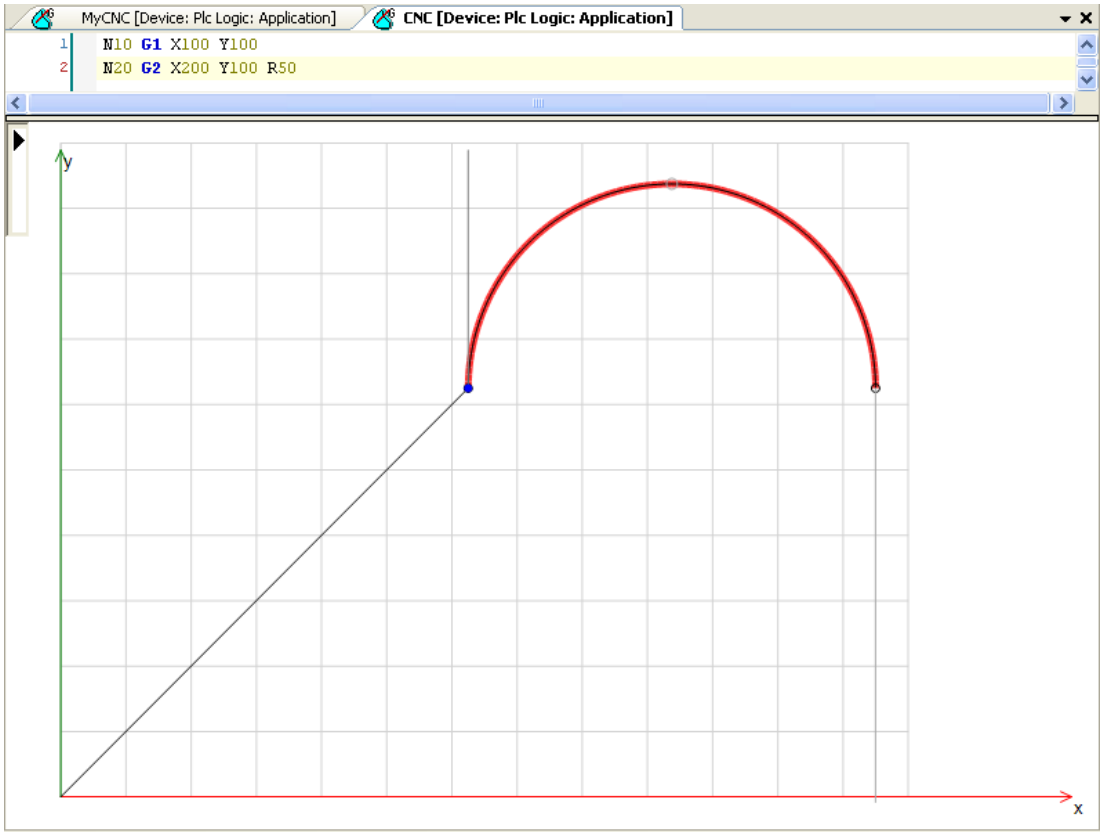


Abb. 328 Circle in the graphic editor

Radius method

If the radius is used, regard that only arcs <180° (semi-circle) are possible, because the radius-method (except for the case of a semi-circle) always describes two possible solutions: one smaller than a semi-circle, one bigger than a semi-circle. The system always takes that one which is smaller than a semi-circle.

Example for a semi-circle:

```
N10 G1 X100 Y100 F100
N20 G2 X200 Y100 R50
```

If a circular arc with an opening angle >180° is desired, that must be defined via (I/J/K). This method is unique, except for the case that start- and endpoint of the circle are identical. There either a null- or a full-circle could rise. In this case the system will insert a full-circle.

Center method

Also, when using the(I/J/K)-method, regard that it is decisive whether the I/J/K (G98/G99) are given relatively or absolutely. If I and J are not given correctly (the distance between the center and start resp. end point must be identical), then no circular arc is possible and the system will replace it by a line.

Example for the same semi-circle (as shown above), specified via the relative center:

G-Code

```
N10 G1 X100 Y100 F100
N15 G99
N20 G2 X200 Y100 I50 J0
```

Example for the same semi circle (as shown above), specified via the absolute center:

```
N10 G1 X100 Y100 F100
N15 G98
N15 G98
N20 G2 X200 Y100 I150 J100
```

Aperture angle method

If the arc is defined with the aperture angle T, you have to regard, that maximal three full circles (T >= 180) are allowed.

Example for the same semi circles specified via aperture angle method

G-Code

```
N10 G1 X100 Y100
N15 G99
N20 G2 I50 J0 T180
```

Example for a helix (two full circles)

G-Code

```
N10 G1 X100 Y100
N15 G99
N20 G2 I50 J0 T720
```

Arcs in 3D-mode

In 3D-mode, the system places the circular arc through the interpolation plane defined via the commands G15-G19: „CNC-Languages DIN 66025“.

Example for semi circle in the X/Z-plane and semi circle in the X/Y-plane

G-Code

```
N10 G1 X100 Y100
N15 G18
N20 G2 X200 R50
N25 G17
N30 G3 X100 R50
```

Delay Time

Command G4

With this command in DIN66025 a delay time is implemented. The interpolator pauses for a certain length of time on its current position.

Example:

```
N0 G4 T1 (delays for 1s)
```

Spline interpolation

Command G5, G10

For the description of splines in DIN66025 it suffices to enter the target coordinates of the next spline segment. The spline segment will be calculated automatically by the system in a way that the end vector of the preceding element and the start vector of the spline, as well as the end vector of the spline and the start vector of the subsequent element are matching, so that no kink will arise in the path.

Allowed syntax

G5 X Y Z A B C P Q U V W F E H L/O D S

G10 X Y Z A B C P Q U V W F E H L/O D S

Parameters of the command	
X Y Z	Target position of the Cartesian axes
A B C P Q U V W	Target positions of the additional axces
F E	Path velocity, acceleration, deceleration G-Code word identifiers (character address)
H L/O	Switch point (H-function)
D	Tool radius
S	S profile G-Code word identifiers (character address)

Please regard, that the spline interpolation is more CPU-intensive than the interpolation via a line or an arc. This brings the necessity to set up the interpolation cycle of controllers with limited performance.

Example

Stair profile rounded with splines:

```
N0 G0 X0 Y0 Z0 F100 (Start position)
N10 G5 X20 Y0
N20 G5 X20 Y20
N30 G5 X40 Y20
N40 G5 X40 Y40
```

Abb. 329 Treppenprofil mit Splines verrundet

This illustrates how the system defines the start and end tangents of every single spline section: At the start of the spline the start tangent depends, whether elements with tool meshing are exist in advance (G1, G2, G3, G8, G9) or not (e.g. G0, G92, M). In the first case their end tangents are used as start tangent for the spline. In the second case the connection line between start point and first spline point is applied as a tangent. In the center of the spline the adjoining points are connected. Then the tangent of the point is in parallel with this connection line (see the green line). The end of the spline is treated equal to the start of the spline: Consist the next element of tool meshing, then its start tangent is used. If not, the line between the last point and next to last point is used.

Example

Profile rounded with duplicated splines:

```
N0 G0 X0 Y0 F100 (Start position)
N5 G1 X5 Y0
N10 G5 X20 Y0
N20 G5 X20 Y20
N21 G5 X20 Y20
N30 G5 X40 Y20
N40 G5 X40 Y40
N45 G1 X0 Y40
```

This example differs from the previous in that the spline point at X20 Y20 is duplicated. Thus, the spline is disconnected and restarted. This is reflected in the determination method of the tangent at this point. Another difference from the preceding example is the start and end of the splines. Here are the starting tangents defined by the previous or next line segment.

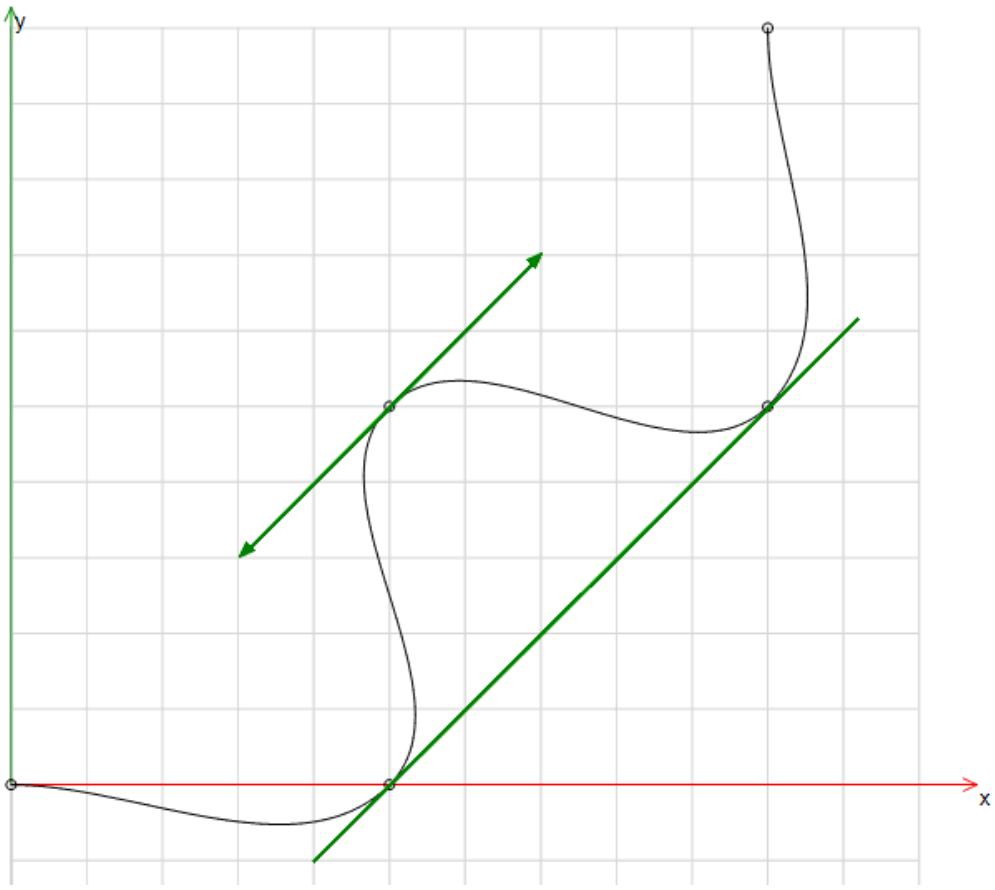


Abb. 330 Treppenprofil mit Splines verrundet

Parable

Command G6

For the description of a parable in DIN66025 the end point coordinates (X/Y/Z), the direction of the main axis of the ellipse (K) and the main- and subaxis (R) length ratio must be specified.

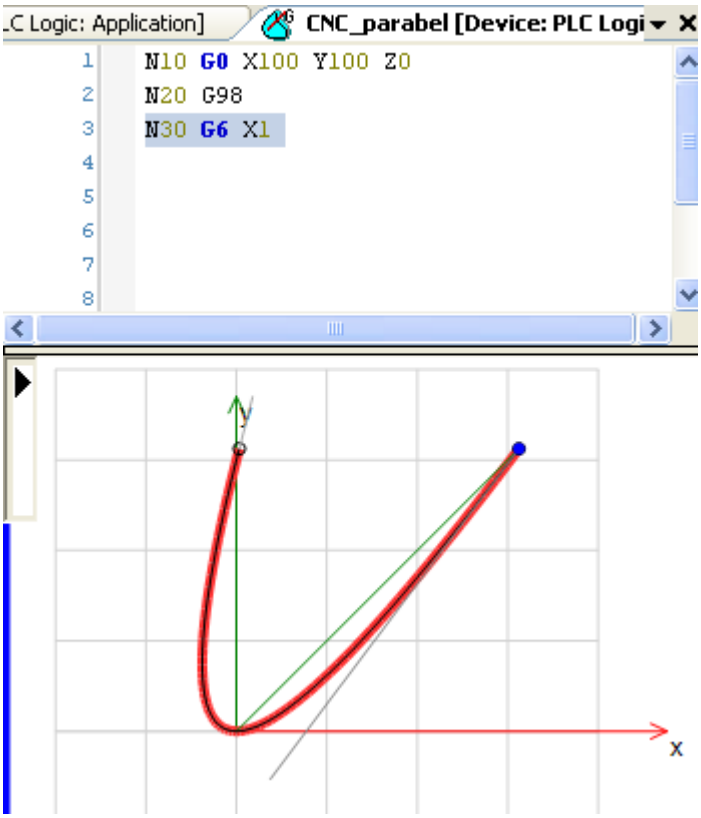


Abb. 331 CNC parable element

Allowed syntax

G6 X Y Z I J A B C P Q U V W F E H L/O D S

Parameters	
X Y Z	Target position of the Cartesian axes
I J	Vertex of the parabola
A B C P Q U V W	Target positions of the additional axces
F E	Path velocity, acceleration, deceleration G-Code word identifiers (character address)
H L/O	Switch point (H-function)
D	Tool radius
S	S profile G-Code word identifiers (character address)

Note: Regard, that this object is a 2,5-D object. So the parabola is always located in the X/Y-plane. Parabolas in other space planes aren't supported.

Example

G-Code

```
N010 G00 X100 Y200 Z0 F100
N020 G98
N030 G06 X-100 Y200 Z00 I0 J0
```

Ellipse interpolation

Command G8, G9

For the description of an ellipse in DIN66025 the target coordinates (X/Y), the center (I/J), the direction of the main axis of the ellipse (K) and the minor- to major-axis (R) length ratio must be specified.

Allowed syntax

G8 X Y Z R I J K A B C P Q U V W F E H L/O D S

G9 X Y Z R I J K A B C P Q U V W F E H L/O D S

Parameters	
X Y Z	Target position of the Cartesian axes
R	Ratio of the lenght between major and minor axis
I J	Center point
K	Direction of the major axis of the ellipse in degree (0: in direction to the X-axis, 90: in direction to the Y-axis, -90: in diametrical direction to the Y-axis
A B C P Q U V W	Target positions of the additional axces
F E	Path velocity, acceleration, deceleration
H L/O	Switch point (H-function)
D	Tool radius
S	S profile

The length ratio of major to minor axis (R) must be in range]0, 1]. It is used only if the ellipse arc through the endpoints, the midpoint and the inclination of the axis are not clearly defined. This is the case when both endpoints have the same distance to the main axis. (Then both end points have the same distance to the minor axis, else there isn't a ellipse through these points and the system replaces the ellipse with a straight line.)

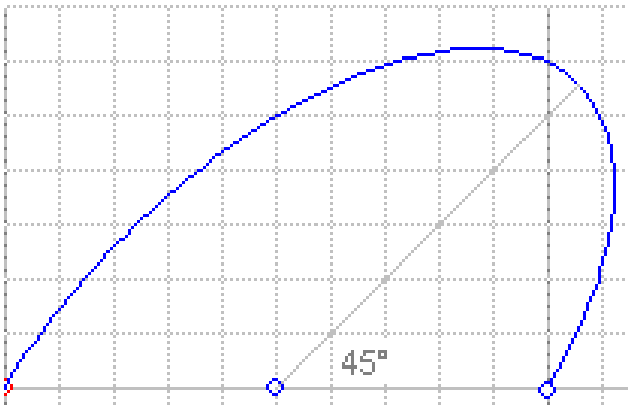


Abb. 332 Ellipse interpolation

Note: Regard, that this object is a 2,5-D object, so the ellipse is always located in the X/Y-plane. Ellipses in other space planes aren't supported.

Example:

G-Code

```
N10 G0 X100 Y100
N15 G98
N20 G8 X200 Y100 I150 J100 K45 R0.5
```

3D mode

In this mode of DIN66025 the user has the possibility to use three dimensional path elements. So several points in the 3D space can be set and proceeded with 3D splines or the CNC path contains arcs placed in space as desired. Lines in 3D mode can be connected with 3D splines and so a smooth movement in 3D is possible. In different to the 2,5-dimensional path elements, which uses X/Y coordinates, the path velocity and acceleration relates on the movement in the coordinates X/Y/Z. (These Z axis has become the same use as X and Y axis and so every movement can be implemented in Z direction).

Main differences between 2.5D and 3D

- The velocity and acceleration definitions (F,E) are applied in different ways:
  - in 3D: it is used for the velocity/acceleration on the path in 3D
  - in 2.5D: it is used for the velocity/acceleration on the projection of the path in X/Y plane.
- Splines in 3D are supported in 3D mode (G10) with a smooth Z-component. The splines in 2.5D are not smoothed in Z direction.
- Circular arcs can be defined in 3D in every possible plane in space. In 2.5D this was limited to the X/Y plane.

Example

```
N0G1 X1 Z100 F1
```

In 3D this movement will take about 100 seconds, because the length of the object is icluding the Z component.

length = $(1^2 + 100^2)^{1/2}$

With velocity 1 it will take about 100 s. Hence, the velocity or the Z component will be about 1.

In 2.5D this movement will take 1 second for calculating the length = 1. Hence, the velocity on Z direction will be about 100.

General

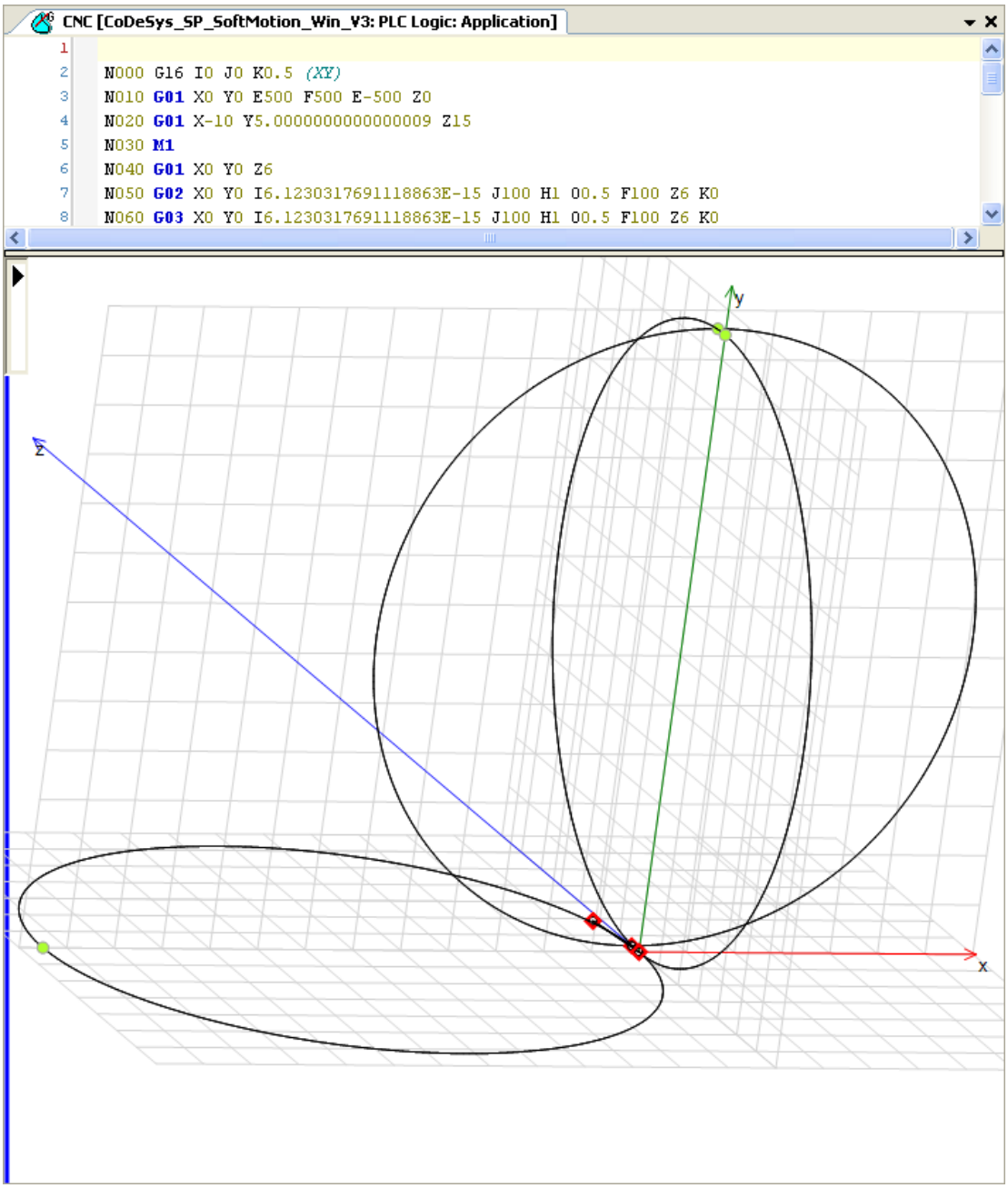


Abb. 333 CNC path in 3D mode

Ellipses, parables, and 2D-splines are not supported in 3D mode and an error message is generated.

The SMC_CheckVelocities function block provided in SM3_CNC library checks the Z component in different to the SMC_ToolCorr and SMC_AvoidLoop function blocks. They generates errors. The Z component of the 3D elements defines the length or the object and hence the interpolation time is affected.

Command G10

This command generates 3D cardinal splines.

Command G15

This command changes the element and all further elements to 2D mode.

Command G16

This command changes the path element and all further elements to 3D mode by activating the 3D plane function in normal plane with coordinates I/J/K. Lines, arcs and splines support this function.

Command G17

This command changes the path element and all further elements to 3D mode by activating the 3D plane function in X/Y plane. Lines, arcs and splines support this function.

Command G18

This command changes the path element and all further elements to 3D mode by activating the 3D plane function in Z/X plane. Lines, arcs and splines support this function.

Command G19

This command changes the path element and all further elements to 3D mode by activating the 3D plane function in Y/Z plane. Lines, arcs and splines support this function.

Jumps

Command G20

Using command G20 in DIN66025 a **conditional** jump can be realized. For this purpose the sentence number of the jump target (L) and the condition for the jump (K) are entered. If no jump condition is specified, automatically the implicit decoder variable will be used. The jump will always be executed, when the condition is not 0.

The graphical editor does not support the use of G36/G37 commands with explicit variables.

Allowed syntax

G20 L O

Parameters of the command	
L	Sentence number of the jump target.
O	Condition: The jump is executed, when the value of O is not 0 (O <>0). If O is not specified, then the intern decoder variable is used.

Example

G-Code

```
N00 G36 D10       (Set counter to 10)
N10 G91           (Relative mode)
N20 G01 X10 Y10 F100 (Motion to the distance 10/10)
N30 G37 D-1       (Decrement counter)
N40 G20 L20       (Jump, if counter != 0)
```

This example effects that ten lines are concatenated - by the relative mode, which results in a straight line movement to 100/100.

Example

Regard, that the jump condition is interpreted at the time of decoding. So, if a variable X in the logic part of the application is programmed in a way, that it gets FALSE, when the X-axis exceeds the position 15, then it is obvious that the program jumps to sentence 20 in the first cycle. Currently X is initially TRUE and the X-position is now 10. In the second cycle you might think, that the condition for X=FALSE is fulfilled, and so no jump in sentence 20 is done, because the X-position is 20. But actually the loop is continuously executed, because the evaluation of the condition is performed at the time of decoding, when the interpolator is not started yet or is engaged in the processing of the objects from the beginning of the buffer. The decoder can jump out of this loop, when so many objects have been created that the buffer got full and started the interpolator with the execution. Then the condition is fulfilled and the decoder can exit the loop.

G-Code

```
N0 G92 X0 Y0
N10 G91           (Relative mode)
N20 G01 X10 Y10 F100 (Motion to the distance 10/10)
N30 G20 L20 O$x$    (Jump, if counter != 0 )
```

Remedy this, use the command G75. This causes, that the decoder pauses as long in the processing until the interpolator and with it the mechanic reach the appropriate position.

G-Code

```
N0 G92 X0 Y0
N10 G91          (Relative mode)
N20 G01 X10 Y10 F100 (Motion to the distance 10/10)
N25 G75
N30 G20 L20 O$x$     (Jump, if counter != 0)
```

Change variable values

Command G36, G37

In DIN66025 the user can write or modify variables via the commands G36 and G37. The variable to be changed is specified by Ovar. The value to be written (G36) resp. added (G37) to the variable is specified by D.

Allowed syntax

G36 O D

Parameters of the command	
O	Variable to be written. If O is not specified, the internal decoder variable is used.
D	New variable value

G37 O D

Parameters of the command	
O	Variable to be incremented. If O is not specified, the internal decoder variable is used.
D	Increment

Example:

The following example will set the global variable g_i to 5:

```
N1000 G36 O$g_i$ D5
```

The following example effects that the lines 1010 and 1020 will be proceeded five times:

```
N1000 G36 O$g_i$ D5
N1010 G1 X100 F100 E100 E-100
N1020 G1 X0
N1030 G37 O$g_i$ D-1
N1040 G20 L1010 K$g_i$
```

Regard that this mechanism only will work, if the path is online-processed, because only then variables can be used! So the mechanism will not work in the CNC editor! Instead you can process in the following way:

If by O no variable is specified, an implicit decoder variable (type: INT) will be used. This mechanism also works in offline mode in the editor. However regard that only one variable will be available and no nested jumps and loops can be programmed.

Preprocessor

General

In DIN66025 some preprocessors (SMC_SmoothPath, SMC_RoundPath, SMC_AvoidLoop, SMC_ToolCorr , selectable in 'CNC settings') need to be activated with G commands. Then the enclosed path segment is corrected and with enabled 'Show preprocessed path' option in the CNC menu the correction is visible in the graphic editor.

Abb. 334 CNC path with slur path

Via the following commands the preprocessor function blocks can be switched.

G40

It ends the tool radius correction.

G41

It starts the tool radius correction to the left of the work-pieces.

G42

It starts the tool radius correction to the right of the work-pieces.

Allowed syntax

G40

G41 D

G42 D

Parameters of the command	
D	tool radius

Note: A detailled function description of tool radius correction is located with the preprocessing function block "SMC_TOOLCORR (FB)“ of the library "The library SM3_CNC.Library“.

Commands for managing the smooth and round functionaliy

G50

It ends the round-path (SMC_RoundPath) resp. the smooth-path functionality (SMC_SmoothPath).

G51

It starts the smooth-path functionality.

G52

It starts the round-path functionality.

Allowed syntax

G50

G51 D

G52 D

Parameters of the command	
D	Distance of smoothing

Note: A detailled function description of the smooth and round functionality is located with the preprocessing function blocks "SMC_TOOLCORR (FB)“ of the library "The library SM3_CNC.Library“.

Example

G-Code

```
N0 G51 D10 F100
N10 G01 X$g_x$ Y0 F50 E30 E-30
N20 G01 X0 Y$g_y$
N30 G01 X0 Y0
N40 G50
```

Commands for managing the avoid-loop functionality

G60

It ends the avoid-loop function (SMC_AvoidLoop).

G61

It starts the avoid-loop function.

Allowed syntax

G60

G11

Note: A detailled function description of the avoid-loop functionality is located with the preprocessing function block "SMC_TOOLCORR (FB)“ of the library "The library SM3_CNC.Library“.

Coordinate Shift

The functions G53/54/55/56 modify an position offset that shifts the position of all following commands. The offset is handled completely inside the NCDecoder FB. So the coordinates of the path objects (GeoInfo data structures) that are stored in the OutQueue of the Decoder are offset. Basically the functions G54,55,56 operate not only with X/Y/Z axis but also with all additional linear axis (P,Q,U,V,W).

Allowed syntax

G53

G54 X Y Z A B C P Q U V W

G55 X Y Z A B C P Q U V W

G56 X Y Z A B C P Q U V W

Parameters of the command	
X Y Z A B C P Q U V W	Offset value

Via the following commands of DIN66025 the shift of coordinate system can be handled:

G53

This command ends the shift coordinate function and the coordinate system is set to the previous one.

G54 :

It sets an (absolute) offset in the CNC program (so all following X/Y/Z coordinates are offset with the specified position)

Example:

```
N0 G0 X100 Y100
N0 G54 X50 Y50 (set offset 50/50)
N0 G1 X0 Y0 (movement to 50/50)
N0 G54 X100 Y100 (set offset 100/100)
N0 G1 X0 Y0 (movement to 100/100)
N0 G53   (set offset to 0)
N0 G1 X0 Y0 (movement to 0/0)
```

G55:

It adds to the existing offset the values of X/Y/Z.

Example:

```
N0 G0 X100 Y100
N0 G54 X50 Y50 (set offset to 50/50)
N0 G1 X0 Y0 (movement to 50/50)
N0 G55 X100 Y100 (set offset to 150/150)
N0 G1 X0 Y0 (movement to 150/150)
N0 G53    (set offset to 0)
N0 G1 X0 Y0 (movement to 0/0)
```

G56 :

It modifies the internal offset in the decoder so that the current position is X/Y/Z.

Example:

```
N0 G0 X100 Y100
N10 G56 X0 Y0 (actual Position (100/100) becomes (0/0))
N20 G1 X10 (movement to (110/100))
N30 G56 X20 Y0 (actual Position (110/100) becomes (20/0))
N40 G1 X0 (movement to (90/100))
```

Note: G54, G55, and G56 change the offset only for those coordinates that are given as parameters. The remaining offset coordinates are unchanged.

Example:

```
G54 X10 Y20 Z30  U7 ( offset: X=10, Y=20, Z=30, U=7  )
G55 X-10 U7       ( offset: X= 0, Y=20, Z=30, U=14 )
```

Time synchronisation with interpolator

Command G75

This command in DIN 66025 prevents the function block decoder from processing the G-Code until the interpolator has passed all preceding objects. When the interpolator has reached G75, all queues are empty and the NCDecoder (and all preprocessing FBs) continue.

This function is useful when using variables. The variables are replaced by the decoder when it processes the corresponding line, but the executing by the interpolator happens, due to the Queue mechanism a certain time later. G75 can be used to wait and "synchronize" the evaluation and the variable and the processing by the interpolator.

Allowed syntax

G40

G75

Example:

```
N10 G1 X100
N20 G75
N30 G1 Y$g_y$
```

In this example the variable g_y is provided in element G10 and is not known before. If G75 is not used, the NCDecoder will process line 30 immediately, before the value is known.

G75 has no effect in the CNC editor or paths that are generated as OutQueue by the CNC editor.

Modes

Command G90, G91, G98, G99

They define how coordinate values are interpreted.

G90

It effects a interpreting of the following coordina***te values (for A/B/C/P/Q/U/V/W)*** as absolute values. It is the default mode.

G91

It effects a interpreting of the following coordinate values (for A/B/C/P/Q/U/V/W) as relative values.

G98

It effects a interpreting of the following coordinate values of I/J/K as absolute ***values.***

G99

It effects a interpreting of the following coordinate values of I/J/K as values relative to the starting point. It is the default mode.

Allowed syntax

G90

G91

G98

G99

Example

The coordinates interpreted as absolute values. The interplator moved to postion 10/10 and further to 100/10.

G-Code

```
N0  G90
N10 G1 X10 Y10 F100 (Start position)
N20 G1 X100 (Next position)
```

Example

The coordinates interpreted as relative values. The interpolator moved to position 10/10 and further to 110/10.

G-Code

```
N0  G91
N10 G1 X10 Y10 F100
N20 G1 X100
```

Example

The coordinates interpreted as absolute values. The center of the semicirle is 150/0.

G-***Code***

```
N00 G98
N10 G92 X100 Y0 (Start position)
N20 G2 X200 Y0 I150 J0 F100
```

Example

The coordinates are interpreted as values relative to the starting point. The center of the semicirle is 150/0.

G-Code

```
N00 G99
N10 G92 X100 Y0 (Start position)
N20 G2 X200 Y0 I50 J0 F100
```

Switch point (H-function)

Function H

The switch functionality or H-function in DIN66025 enables the programmer to operate binary path-dependent switches. Basically always first the number of the switch must be specified ("H<number>"), then the switch position must be defined, either absolutely by using the word "L<position>" or relatively by using the word "O<position>".

Allowed syntax

[G-command] H O/L

Parameters of the command	
H	Number of the H-function
O	Relative position in the object [0..1] 0: Start 1: End
IL	L> 0: Distance to the start point L< 0: Distance to the end point

H-numbers

If the H-number is positive, the corresponding H-function is switched on, otherwise it is switched off. H-numbers must not be 0 and have to lie in the range from -32767 to 32767.

The H-numbers 1-32 are logged in the interpolator. The output dwSwitches of the function block SMC_Interpolator is a bit mask. Bit number i indicates whether the H-function (i +1) is currently set or not. H-numbers greater than 32 are not logged.

The interpolator outputs with iLastSwitch the last switched H-number. iLastSwitch is positive, when the H-function was switched on, and otherwise it is negative.

Graphical representation

In the graphic editor the switches are represented by a small green filled circle on the path. A switch point position only can be inserted in the CNC text editor or in the table editor. In the graphic editor they can be shifted along the path.

When moving the mouse on a H-function a tooltip with the sentence number of the H-function is displayed.

Example:

Switch2 is turned off at position X=40/Y=25 (after a fourth of the object):

```
N90 G1 X20 Y20
N100 G1 X100 Y40 H-2 O0.25
```

Example

Switch point 2 is turned on at position X=40 (20 units after the start of the object) and is turned off at position X=90 (10 units before the end point).

```
N90 G1 X20
N100 G1 X100 H2 L20 H-2 L-10
```

Note: For each path object only a limited number of switch point switch actions (MAX_SWITCHES) can be processed. Default value: MAX_SWITCHES = 3.

M-function

Using the additional functions or M-functions in DIN66025 a binary output can be set, which starts another action. In this case in contrast to the switch points it will be waited at the current position until the M-function has been confirmed by setting an input of the interpolator. This often is used if the further processing of the program depends on other processes.

It is possible to pass two numeric parameters using the K and L word. If more parameters are needed, a variable of type SMC_M_PARAMETERS can be passed using Ovar. The application can query the values of these parameters at runtime using the function block SMC_GetMParameters. All used parameters are analyzed at the time of decoding and stored in the SMC_GEOINFO structure in the SMC_OUTQUEUE buffer. This usually take place on the interpolator a certain length of time before the execution of the M-function.

Allowed syntax

M K L O

Parameters of the command	
M	Number of the M-functiom, M > 0
K	Passing parameter (LREAL)
L	Passing parameter (LREAL)
O	Reference to a variable of type SMC_M_PARAMETERS, which contains further passing parameters SMC_M_PARAMETERS (STRUCT)

Reference to a variable of type SMC_M_PARAMETERS, which contains further passing parameters „SMC_M_PARAMETERS (STRUCT)“

Regard, that die signification of the M-function is not defined in the system, but is defined machine-specific by the application and programmed out there.

Graphical representation

In the graphic editor the outputs are represented by a small thick red bordered rhombus on the path. A output position only can be inserted in the CNC text editor or in the table editor.

Example

The following sentence starts the M-function 10 and wait until this gets confirmed:

```
N90 M10 K100.7
```

The following example sets a self declared data structure g_myMParams (type: SMC_M_PARAMETER) with help of Ovar. With a instance of function block SMC_GetMParameters, you can get the variable values of K and the structure g_myParam.

```
N150 M13 O$g_myMParam$
```

A M-function allways causes a stopping of the path, i.e. the interpolator brakes on velocity 0, outputs at wM the number or the upcoming M-functions and waits for its confirmation in bAcknM before accelerating again.

With function block SMC_PreacknowledgeMFunction it is possible to confirm a M-funciton ahead of schedule, i.e. the interpolator doesn't stop, but it behaves - if the function executed in time - like the M-function doesn't exist.

Usage of Variables

In DIN66025 there can be defined a variable or a term embraced by two $ signs. It can be set by using the CNC command Set variables or in the properties window of the CNC object.

Example:

One item of a global array of a structure:

```
$CNC[1].X$.
```

Regard in this context, that the usage of variables only will work, if the program is compiled as program variable (SMC_CNC_REF) and is online processed by the decoder module. The variables will be replaced at that time at which the decoder is processing the corresponding line.

If the CNC program however is compiled as SMC_OutQueue, the variable mechanism will not work because in this case the path is offline created and passed as unmodifiable data structure to the application. In this case – as also for an offline display – the editor will replace variables by their initial values.

The usage of variable in online read G-code programs needs additional preparations.

13.3.2 CNC Table Editor

In the table editor the path elements are listed in a table. The columns of the table are configurable in sequence and visibility in the CNC Settings in the **Table editor** tab.

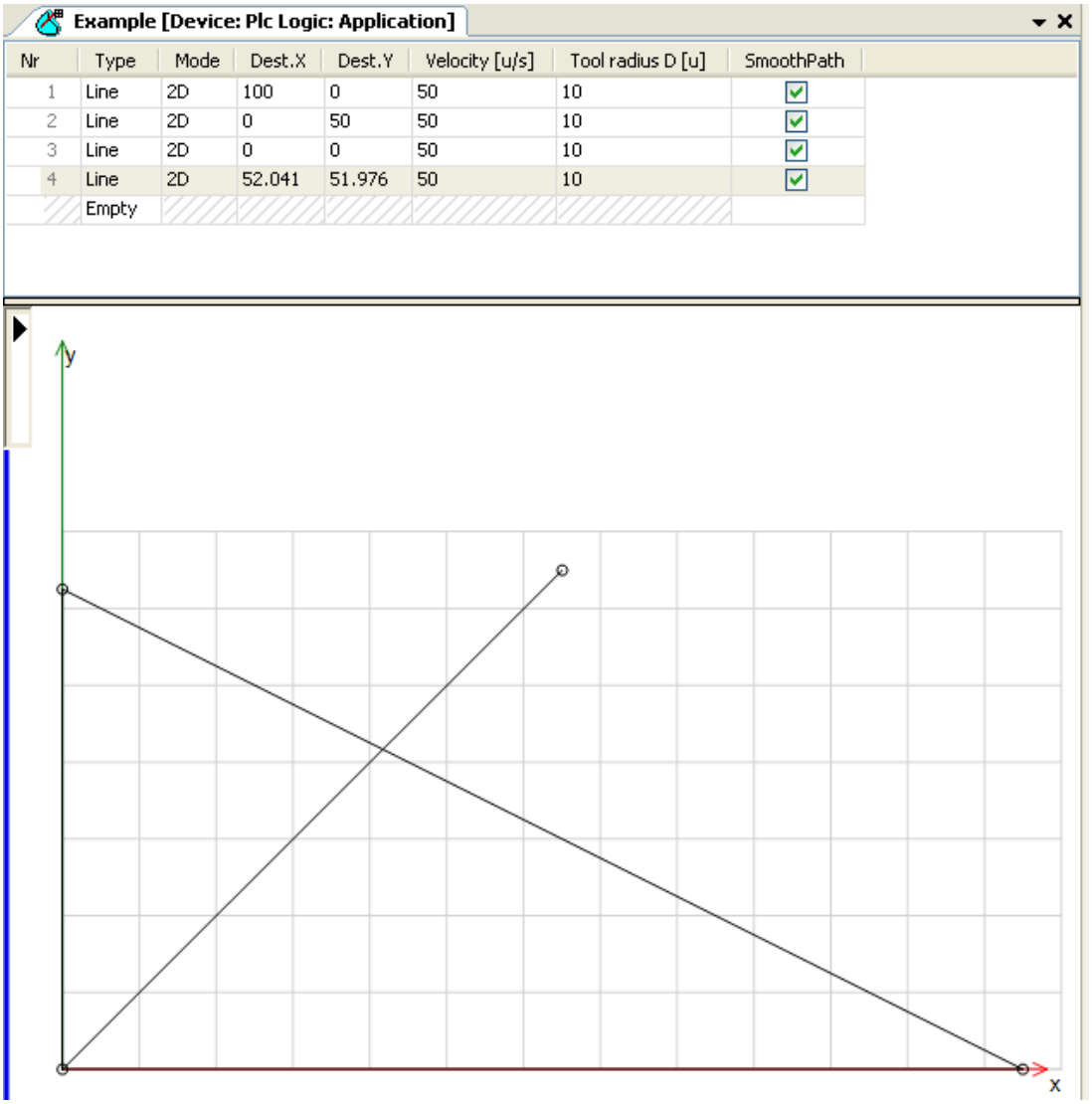


Abb. 335 CNC program in table view

General

By click in the table editor the focus of marked row is highlighted in **light blue** and the selected table entry will be indicated by a thin blue frame in addition. The associated path element becomes **red** with a **blue** start point. By click in the graphic editor the currently selected element path is displayed red colored with a blue start point, while the corresponding row in the table entry is highlighted in light gray. The usability or the column depends on the element type. A column with no use is crosshatched and empty. The find and replace functionality is not supported.

To add a element

The last row of the table is an empty row with the type **Empty**. When double-clicking this entry and selecting a type of the pull down menu, a new row is appended. An other way to get a new element is to open the context menu of the table entries. It provides the command **Insert element**. This command insert a new element with the same values of the element inserted before in the row before the selected row.

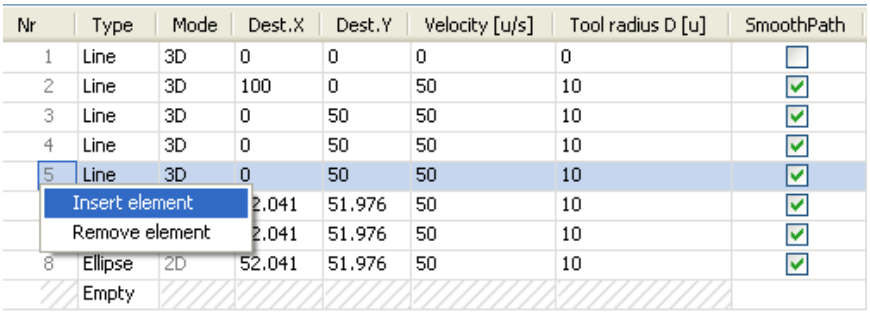


Abb. 336 Context menu of the table editor

To modify a table entry

The content of the changeable columns are displayed in a black font. Such entries can edited by double clicking on them or by selecting it with the mouse and pressing any key. Selected entries will be indicated by a thin blue frame. Then the in line editor or a pull down menu opens and the desired value can inputted. Numbers are formatted in American style. The number of the row can not be changed.

The context menu of the table entries provides the command **Apply value for all elements**. This command copies the current value of the entry in all other entries or the same column, when the value is valid.

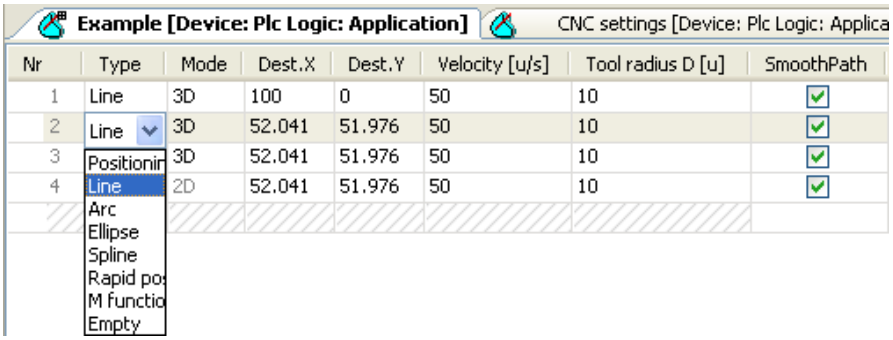


Abb. 337 Pull down menu in the table editor

To delete a element

A selected row can be deleted with the **Del** key or by opening the context menu and command **Remove element**.

Unsupported elements

In different to the text editor the table editor do not support the following elements and commands: parable, IEC variables, jumps, shift coordinates, step suppression, G4 and G75.

13.3.3 Graphic Editor

The graphic editor is in the lower left part of the CNC editor. On the one hand it visualizes the CNC program which is described in the text editor or the table editor, on the other hand you can modify the program using the ToolBox. All changes will be updated automatically and immediately in the Din66025 editor or in the table view in the upper window part.

General

By click in the graphic editor a blue border on the left side of the window gets visible and indicates the window focus. As long as the focus is on the graphic editor the blue border is there. A coordinate system with green y-axis, red x-axis and blue z-axis is displayed. A light gray colored grid is in the background, which can switched on and off via the command **Show grid** in menu "CNC". Positioning commands (G0) including switch positions are displayed green colored in different to all other elements which are black colored. If the end point is relocatable, it is visualized as small black bordered circle. Start and end tangents are gray colored. By click on a path element it gets red colored with a blue filled start point and a thin black end point. The corresponding code sentence in the text editor is light-blue colored. In table view the table row is highlighted in light gray. In the status line the current position of the selected path element is shown. The coordinates of the position are displayed in world coordinates, if possible.

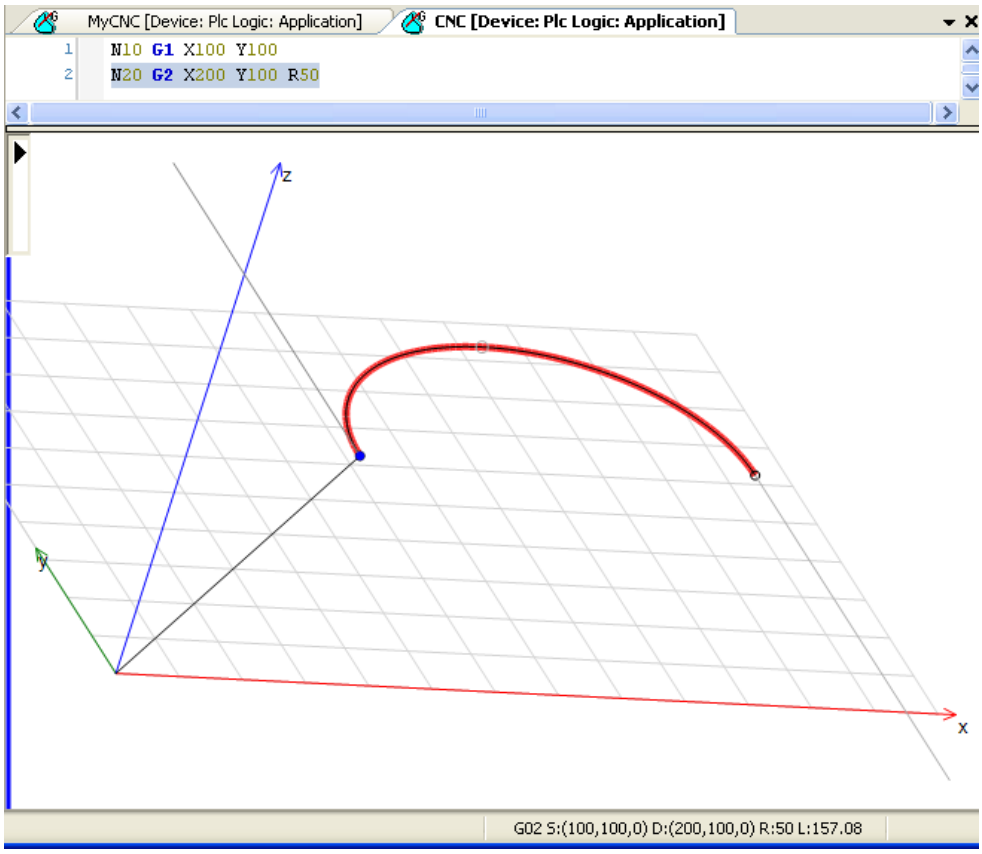


Abb. 338 Graphic view of a CNC path

Moving the 3D program path

By keeping pressed the **left mouse** button, the CNC program is shifted along the x-, y-, or z-axis. By keeping pressed the **right mouse** button the CNC program is turned on the x-, y-, or z-axis. Using the **mouse wheel** while pressing **Ctrl** key, the zoom factor is to be changed. The control panel in the upper left part of the CNC graphic display provides various commands for moving the CNC program, too.

To add a path element

For inserting or appending a path element the ToolBox provides miscellaneous commands.

To modify a path element

A selected element can be modified by clicking on the relocatable points visualized with small circles. Then the circle becomes red and thick. If tangents are relocatable they become red after a mouse click as well. Then they can be modified by moving their start or end point in the diagram plane of the selected point. Due to the type of the element the number of relocatable points are different. The current values of the associated variables are simultaneously visible in the CNC text editor or in table view.

To delete a path element

A selected element can be deleted by using the **Del** key.

Commands and options

Various commands and options are available to visualize additional sets of data of the CNC program or to manipulate it. They are all placed in the "CNC" menu.

Example

Various CNC programs are enclosed by the setup. Have a closer look at the comprehensive description of them.

Control Panel

In the upper left edge of the CNC editor a control panel provides various commands for changing the appearance of the path.

Abb. 339 CNC control panel

When clicking in the black arrow, the control panel disappears, and the view on the path is enlarged. A little arrow allows access to the panel. With clicking in the minimized panel it maximizes and the whole control panel is visible.

Abb. 340 Minimized CNC control panel

The control panel provides the following commands:

Zoom in.

Zoom out.

Change the camera's pitch .

Change the camera's pitch.

Change the camera's yaw clockwise.

Change the camera's yaw counter-clockwise.

Roll the camera clockwise.

Roll the camera counter-clockwise.

Set view direction to negative z-axis and rescale.

Set view direction to x-axis and rescale.

Set view direction to y-axis and rescale.

13.3.4 Properties of a Path Element

The properties of an CNC path element currently selected in the graphical editor of the CNC editor is displayed in the "Properties" window, which by default is available next to the toolbox window.

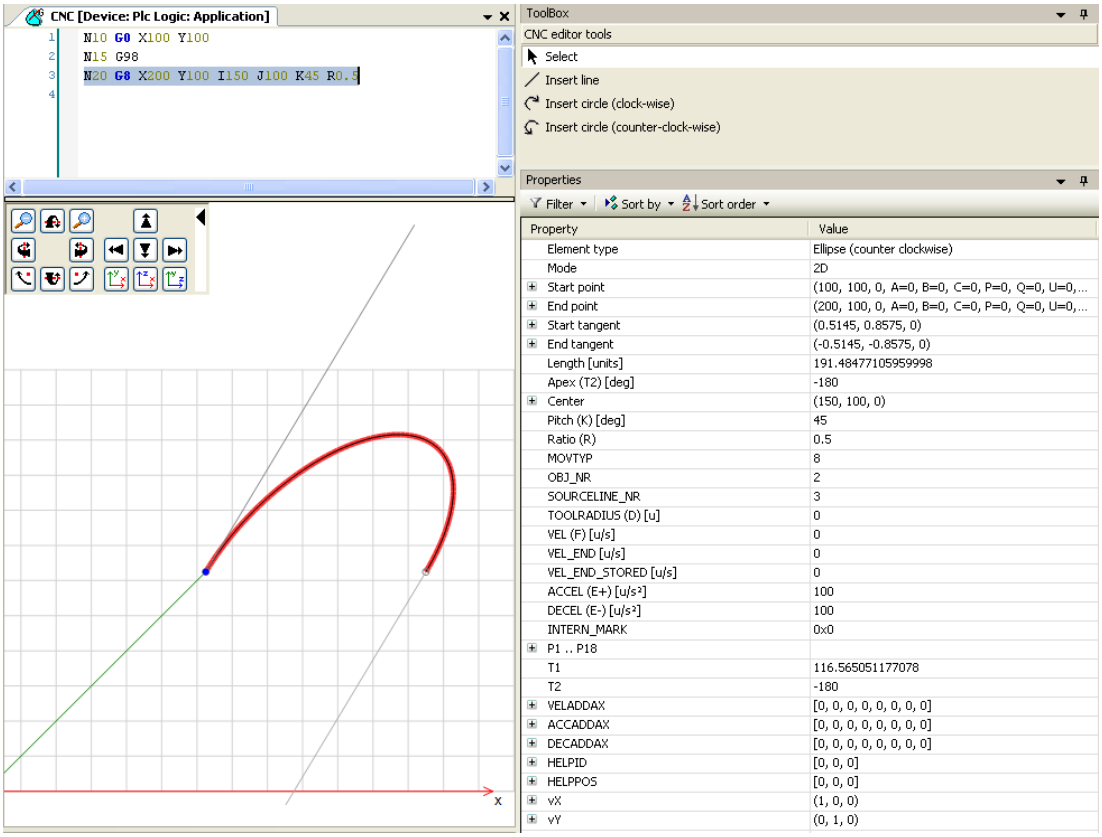


Abb. 341 Example of element type ellipse

13.3.5 ToolBox

The CNC editor ToolBox allows the user to program and extend motion paths in the graphic editor. The changes simultaneously are visible in the text editor or in the table view. Various tools support the programming of the motion path by mouse. Default tool is the 'Select' tool, who provide the selection of a path element in the graphic edtor. To change to insert or append mode you have to click on one of the other tools depending on the desired path element.

- The **Select** tool is by default activated. In active mode a mouse click in the graphic editor will select a point or a path element, who becomes **red** with a **blue** start point. With click on an inactivated select tool gets activated and the cursor in the graphic editor is a pointer.
- The **Line** tool is activated by a click on . In active mode a mouse click in the graphic editor will insert or append a new line.
- The **Circle (clockwise)** tool is activated by a click on . In active mode a mouse click within the graphic editor will insert or append a new clockwise circle.
- the **Circle (counter-clockwise)** tool is activated by a click on , whereon a mouse click within the graphic editor will insert or append a new counter- clockwise circle.
- the **Spline** tool is activated by a click on , whereon a mouse click within the main editor will insert or append a new spline. When the last or selected element is a G5 spline then the new spline is a G5 spline,too. Otherwise the spline is a G10 spline.

Insert a path element

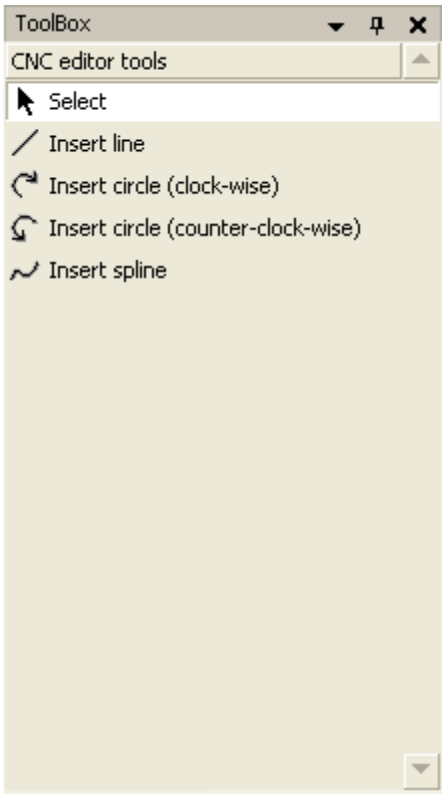


Abb. 342 CNC ToolBox in insert mode

To insert a path element select a element. Before this the new element is to be inserted. Then select the element type in the ToolBox, who at once is in insert mode. Every click in the pathview editor marks the connection point between selected and new element. Accordingly the new element is inserted. It will be displayed **red** with a **blue** start point and hence it is the next selected element and the ToolBox stays in insert mode.

Append a path element

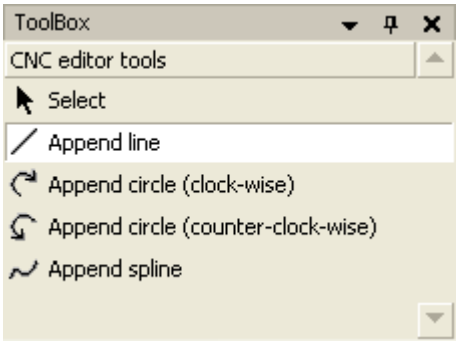


Abb. 343 CNC Toolbox in append mode

To append a path element no element have to be selected in the pathview editor. Select the desired element type in the ToolBox, who now is in append mode. Every click in the pathview editor marks the new end point of the motion path and a new element is appended. The new element will be displayed only in black with two little circles for start and end point.

General

In insert or append mode the cursor in the graphic editor is represented by a cross. The mode can be changed to **Select** by using the **Esc** key. Then the cursor becomes a pointer again.

If the selected implementation mode is text editor, this editor is updated simultaneously with following colouring:

The associated sentence in the text editor is highlighted in a **light blue** inking.

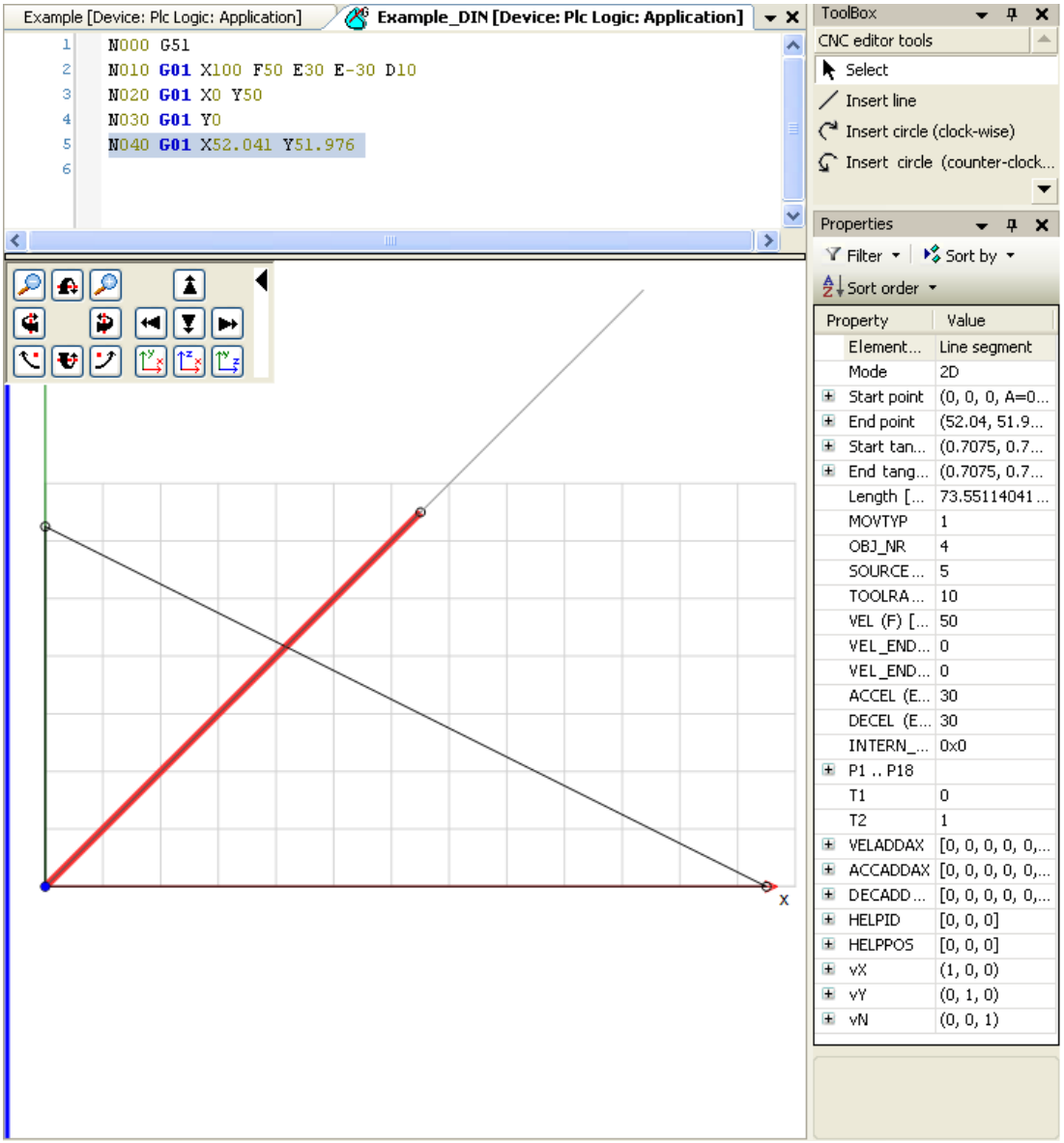


Abb. 344 CNC program in DIN66250

If the selected implementation mode is table view, the table is updated simultaneously with following colouring:

The row in the text editor is highlighted in a **light gray** inking.

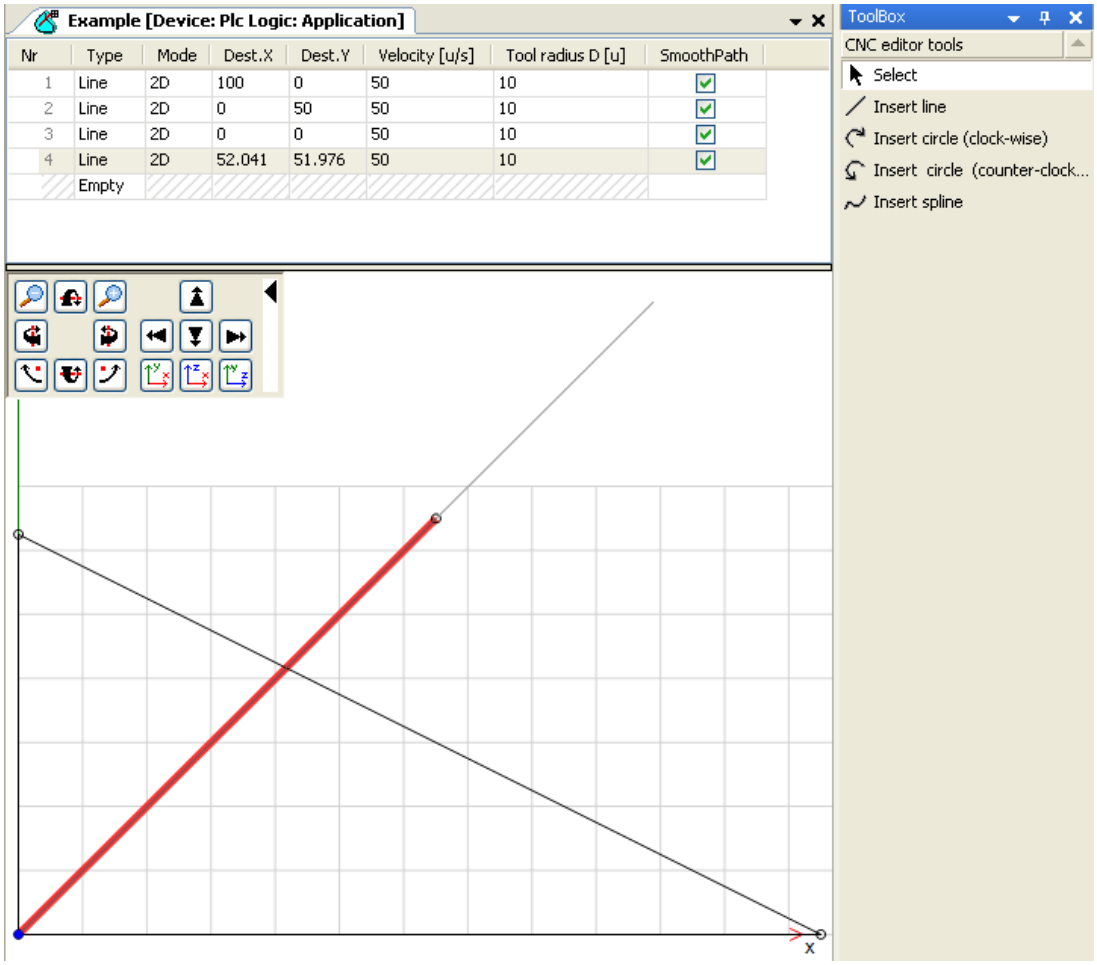


Abb. 345 CNC program in table view

13.3.6 CNC object

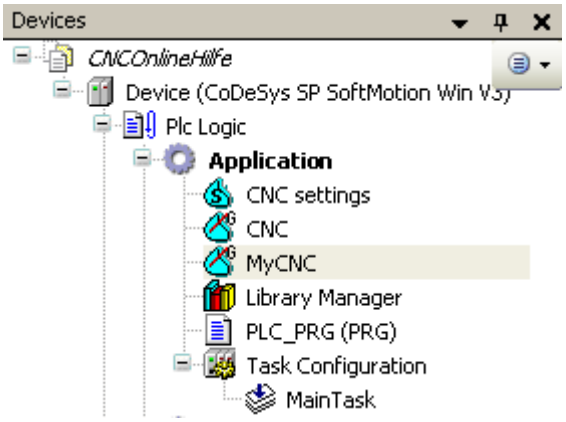


Abb. 346 Device tree with CNC objects

New Object:

A CNC object has to be placed under an application which is capable of doing »PLC Designer« SoftMotion. So with selecting an application and activating the **Add object...** entry of the context menu the associated dialog will open. There an apposite name and the desired implementation and compile mode have to be entered. With clicking the OK button a new CNC object is placed in the device tree. Beside the "CNC settings" object is generated, if it is not yet present. There the application-wide CNC settings are stored. The object-associated CNC settings are stored with the CNC object and can be edited in the "Properties- CNC " dialog.

Delete Object:

Like it is with all other objects, it is possible to delete the CNC object.

Context Menu:

When opening the context menu of the CNC object the "Properties- CNC " dialog and the CNC Info dialog are additionally available.

Properties of a CNC object

With commanding "Properties..." on a CNC object the "Properties - CNC" dialog opens where the CNC program specific settings can be changed. The 'Properties...' command is available in the context menu or in the 'View' category. The dialog provides four tabs: 'Common','CNC', 'Access Control' and 'Build'.

Common

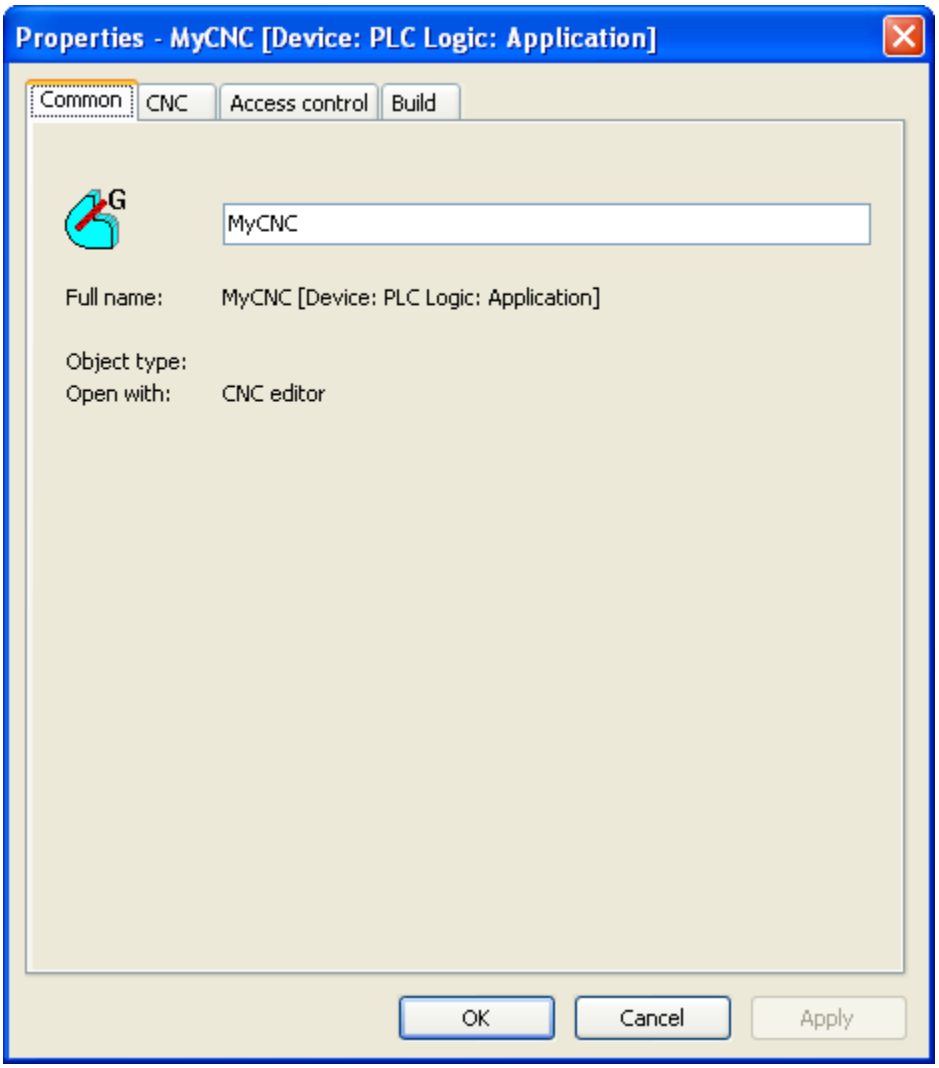


Abb. 347 Properties, Common

The name of the CNC object can be changed here.

CNC

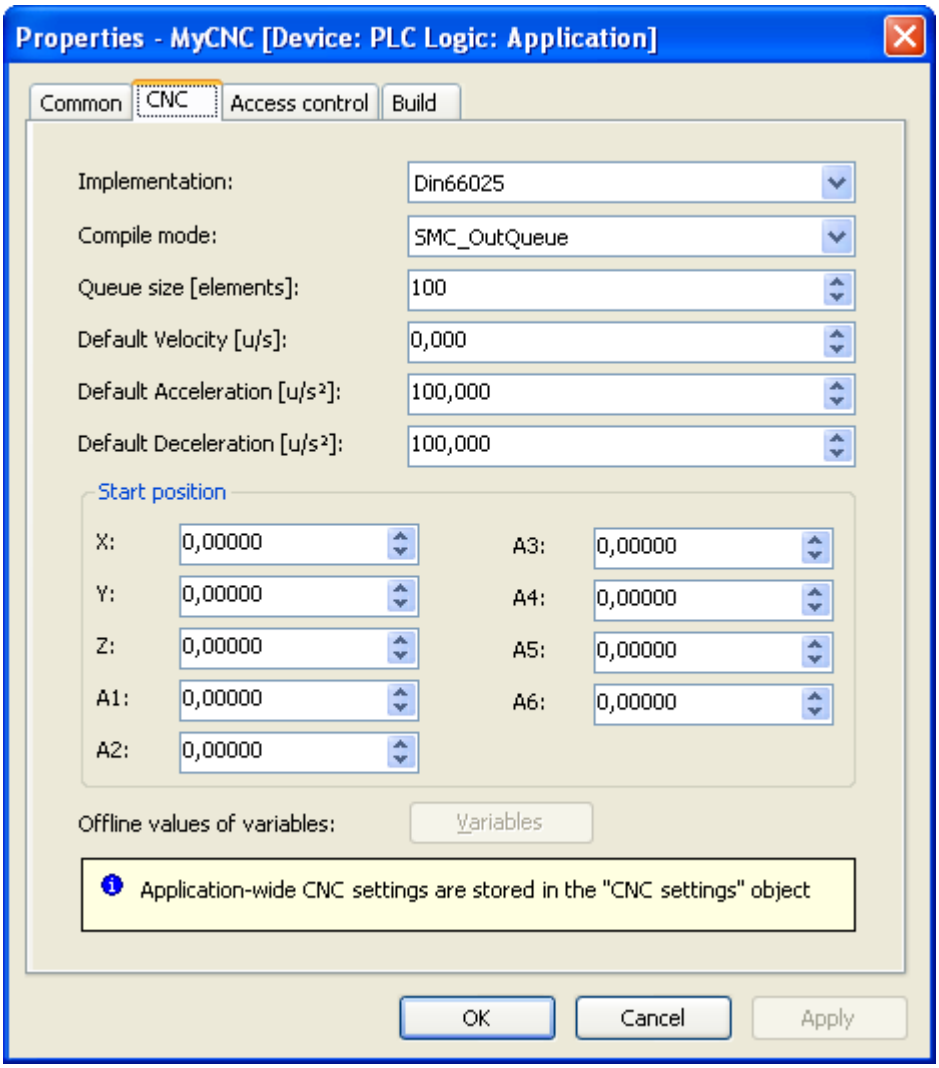


Abb. 348 Properties, CNC

- **Implementation:**
  Here the implementation type is defined.
  Din66025
  Table View

- **Compile mode:**

  The compilation mode depends on the implementation mode.
  - SMC_OutQueue: Creates SMC_OutQueue file on compile.
  - SMC_CNC_REF: Creates program variables on compile.
  - File: With this option a file of the programed G-code is created and with doing a login the file is loaded to the PLC. There the application can read and execute the path from the file with help of function blocks like SMC_NCDecoder and others from library SM3_CNC. The G-Code files on the controller are updated by the G-code objects from the application with every login. This implies that every change of the files on the controller is overwritten with a new download.
- File name: If in the drop-down list of the compile mode File is selected, then you can define here the file name, which is to use. The default name is $ObjectName$.cnc, which represents a placeholder that will be replaced by the object name of the CNC object (effective file name), when you login. If you change the file name, please regard the allowed characters for file names are [a-zA-Z0-9$_.]. The file name field must not be empty, mustn't start or end with a point and must contain only one point and it's not allowed to put in a name with a path.
- **Queue size:** Default value is 100.
- **Default velocity:** Default value is 0. This value is used if no velocity of a path element in the CNC program is set. Value range: [0, 1e9].
- **Default acceleration:** Default value is 100,000. This value is used if no acceleration of a path element in the CNC program is set. Value range:]0, 1e9].
- **Default deceleration:** Default value is 1000,000. This value is used if no deceleration of a path element in the CNC program is set. Value range:]0, 1e9].
- **Start position:** Default value is 0.
- **Offline values of variables:** If offline variables are defined, they can be shown and edited by clicking the **Variables** button. Then the Offline values or variables dialog opens. The same can be done by using the CNC command **Set Variables.**

The application-wide CNC settings are stored in the CNC settings object.

Access control

Here the user group rights can be changed.

Build

Here the CNC object can be excluded from the build. Then no IEC data are generated and it is possible to store the program as an ASCII or as an OutQueue file in the file system of the controller. If you want to read it at run time, use one of the modules of the library SM3_CNC.

CNC Info

The **CNC Information of Program <CNC program name>** dialog provides the following information on the program which is currently loaded to the editor: program name, version of function blocks, number of lines, number of objects, path length and duration. Additionally with successfully generated path preprocessing and interpolation points, the calculated path duration for the preprocessed path is displayed, too.

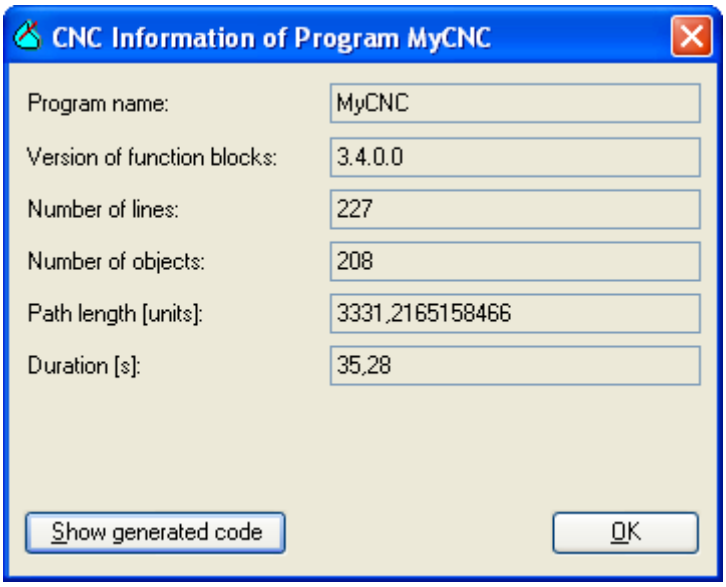


Abb. 349 CNC Info of program MyCNC

Show generated code

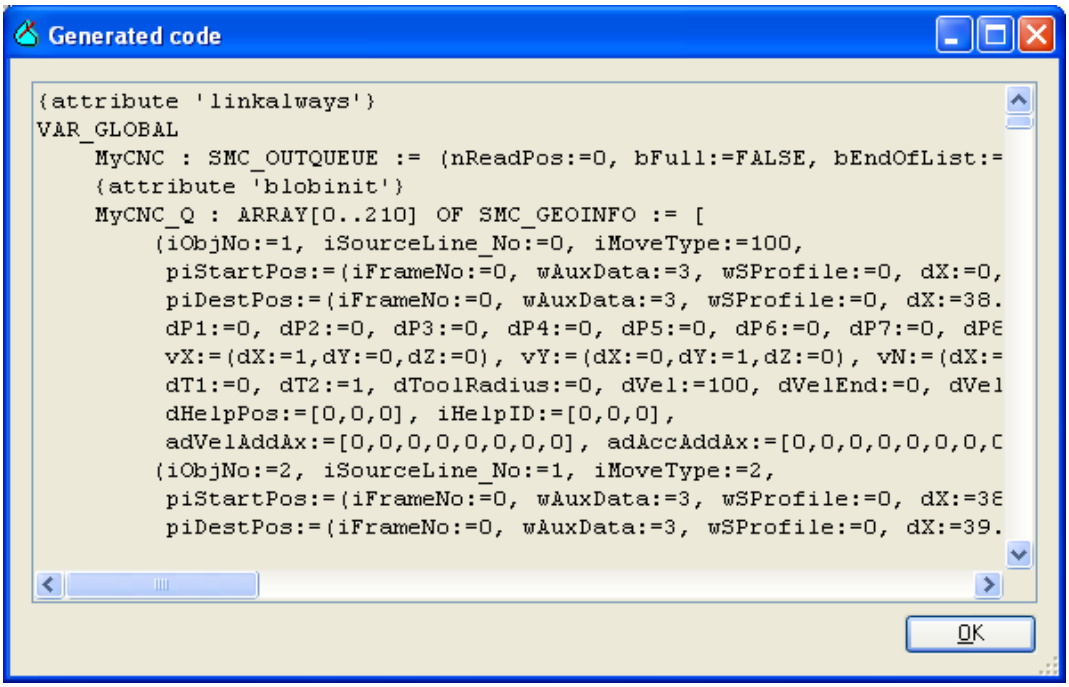


Abb. 350 Generated code dialog

When using the **Show generated code** button a listing of the programmed motion of the dedicated CNC object is shown. This can only be done, when the CNC program has no errors. Then you can see the generated ST-code of the CNC program and there the declaration part, who initializes the SMC_OutQueue respectively the SMC_CNC_REF.

13.3.7 CNC Online

In online mode the status line is invisible.

13.4 CNC Settings Editor

Symbol:

The **CNC Settings object** is generated automatically, when a CNC object is generated and can't be renamed or removed. It is located in the device tree under a application. Only one CNC setting object may exist there. The settings stored here are valid for all CNC objects of the application. Settings only valid for a particular CNC program are stored in the properties dialog of the CNC program.

A double-click on the CNC Settings object opens the CNC Settings dialog. There you can define the settings for the path preprocessor, the preinterpolation and the CNC table editor. The object can only be deleted, when no CNC object exists under the application object.

Path Preprocessor

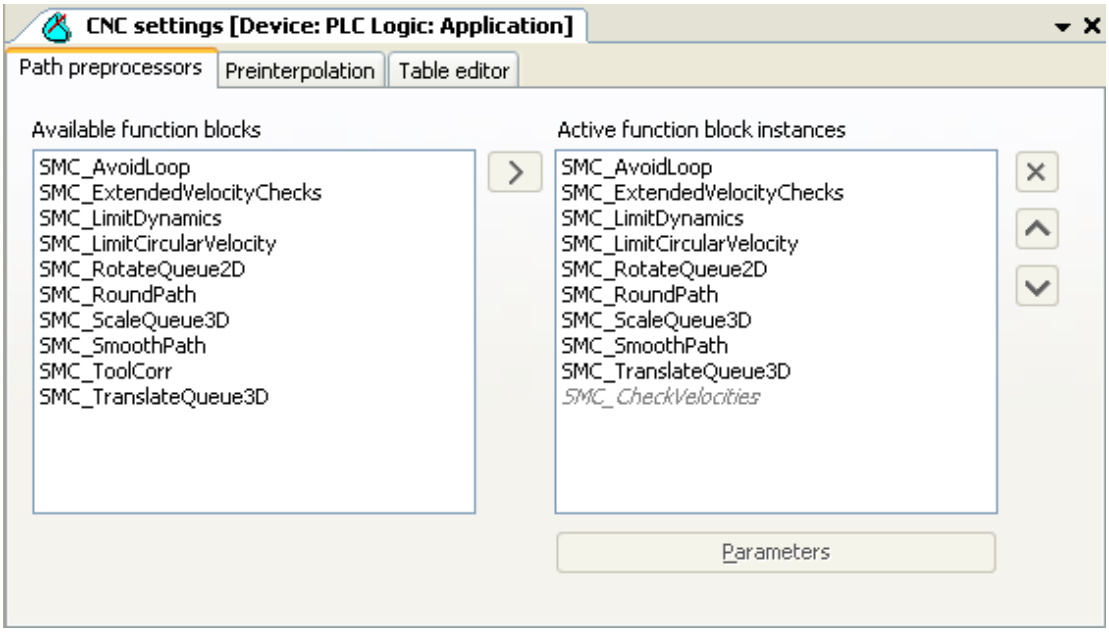


Abb. 351 CNC Settings for path preprocessor

Here the modulation program for the path preprocessor can be programmed by selecting the desired **function blocks**. The position in the list of the active function block instances defines the turn in the modulation program and can be changed. It is possible to instantiate the same function block more than once in the active function block instances list.

In the left part the available function blocks are listed, in the right part the active function block instances. It is not possible to delete function block instances displayed in italic light gray font.

Use this button or the **Space** key for copying a selected available function block into the list of the active function block instances. The same can be done by **double clicking** the desired function block in the left list .

Use this button or the **Del** key for deleting the selected instance.

Use this button for moving up the selected function block instance by one position. The same can be done by using the **Arrow Up** key while pressing the **Shift** key.

Use this button for moving down the selected function block instance by one position. The same can be done by using the **Arrow Down** key while pressing the **Shift** key.

Some instances have parameters, which can be set and changed by clicking the button **Parameters** or by double clicking on them in order to open the respective parameter editor dialog. The same can be done by using **Shift** + **P**, **Alt** + **P** or **P**.

Preinterpolation

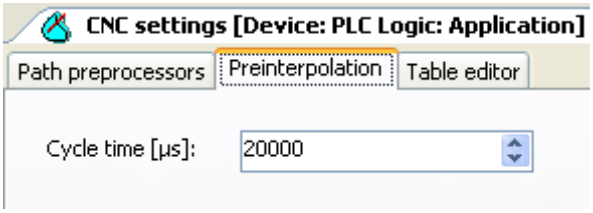


Abb. 352 CNC Settings for preinterpolation

Here the interpolation time can be set. Default value is 20000µs.

Note: Interpolation time and cycle time of the associated task have to match for identical behaviour of CNC editor and online application.

Table Editor

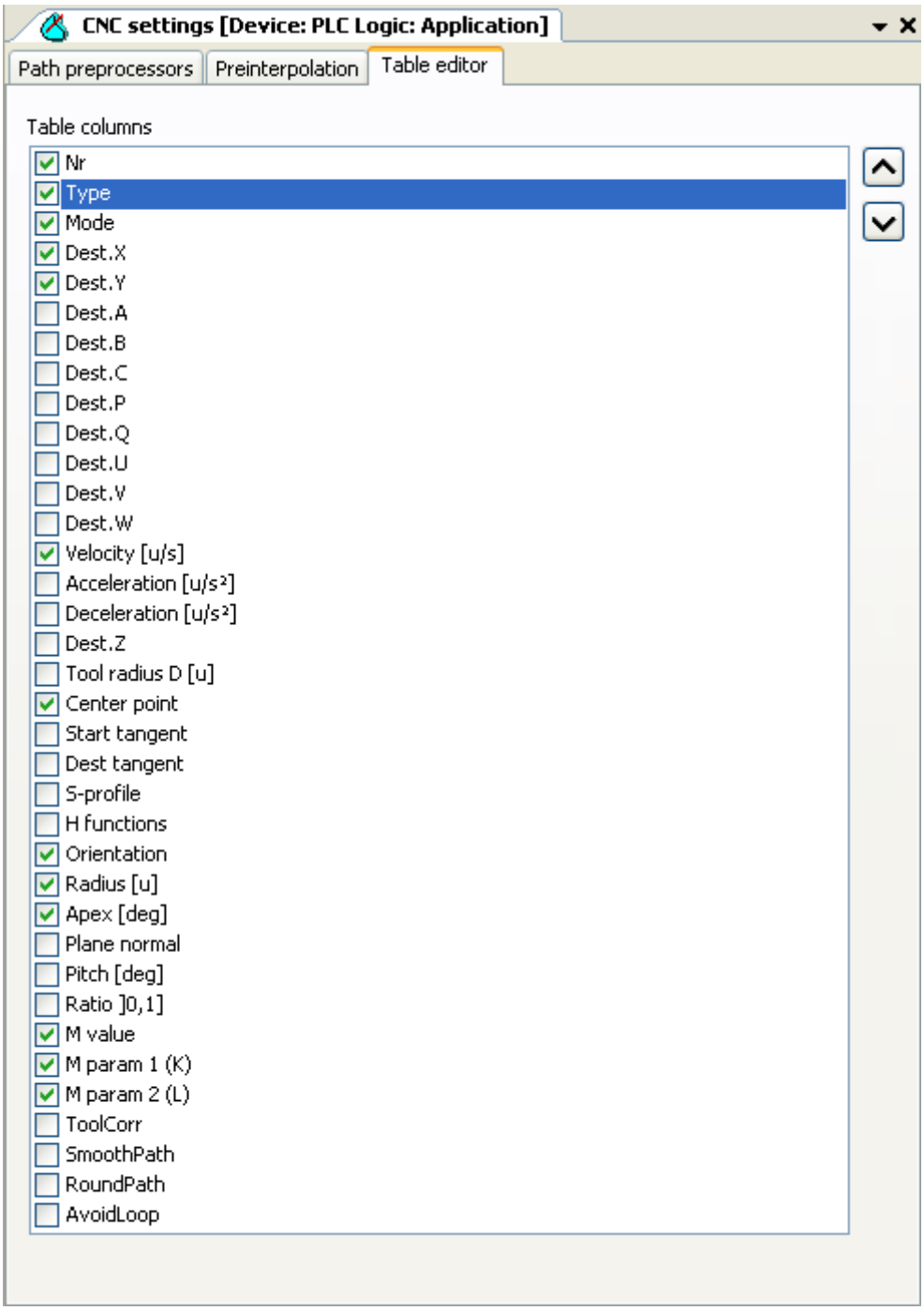


Abb. 353 CNC Settings for the table editor

Here the columns and their order in the table editor can be defined. In the table columns list all available attributes are listed. Only attributes for which the checkmark is set, will be visible in the table editor. The checkmark can be set or removed by a click on the check box or by using the **Space** key. To define the order of the attributes use the or **Shift** + **Arrow Up** key for moving up and the or **Shift** + **Arrow Down** key for moving down the attribute within the list. The type column cannot be set visible. The changes immediately effect all table editors, also the open ones.

13.4.1 Parameter editor

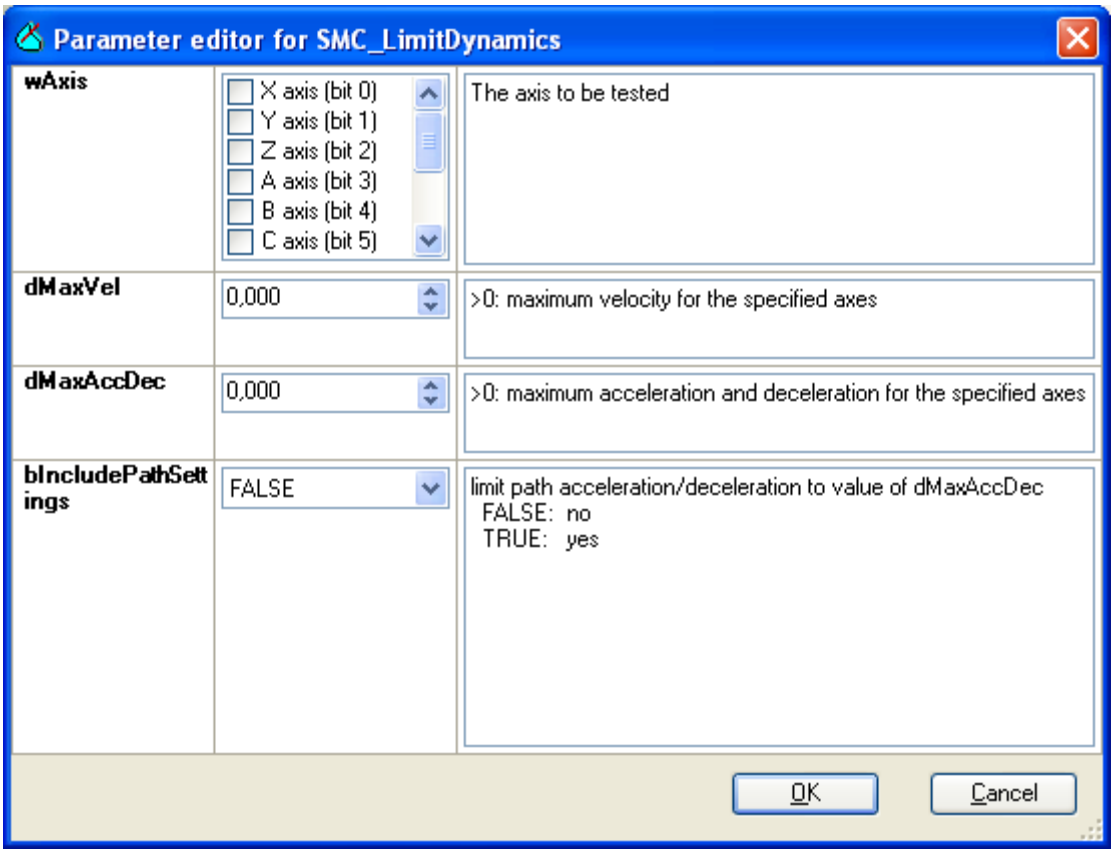


Abb. 354 Parameter editor dialog

In this dialog it is possible to edit the inputs of the function block instance for the modulation program of the path preprocessor. Please note that not all inputs are visible. (The editable ones have got the initial value before the first editing.) The first column contains the input name, in the second column the changeable value is displayed and can be changed with the help of pull down menus or check boxes. In the last one the input comment can be read.

13.5 Data Source Editor

Note: This functionality is only available if supported by the currently used feature set.

A Data Server can be added to the local application and serves to establish a point-to-point connection between the local application and a remote data source. Thus **data items**, i.e. variables, provided by other devices or (currently not yet possible) by OPC Servers can be used and visualized in the local application. Multiple data sources might be added to the data server of the local application.

Note: The data server does not replace the Network Variables functionality which is already used with »PLC Designer« V2.3, but provides another, different way of data exchange. Network variables allow data transfer basing on defined variables lists and in broadcast mode.

Note: Bit accesses used in visualizations will only work over a Data Server connection if the bit index is defined literally (and not by a defined constant).

The **data server** must be added as an object in the devices tree below an application. Only one data server per application can be configured. Automatically the required data server libraries will be included in the project.

The **data source(s)** to be handled by the data server also are to be added as objects, below the data server.

The data source editor provides dialogs for the configuration of the particular data sources. This concerns: which variables (data items) should be used, of which access type should they be, which are the settings for the communication with the data source, i.e. update of the values.

The data server and data source functionality is provided by the plug-ins 'DataSourceEditor' and 'DataSourceObject'.

13.5.1 Add data server

A Data Server object can be added to an application via command 'Add Object': Only one Data Server can be added per application and the object cannot be renamed.

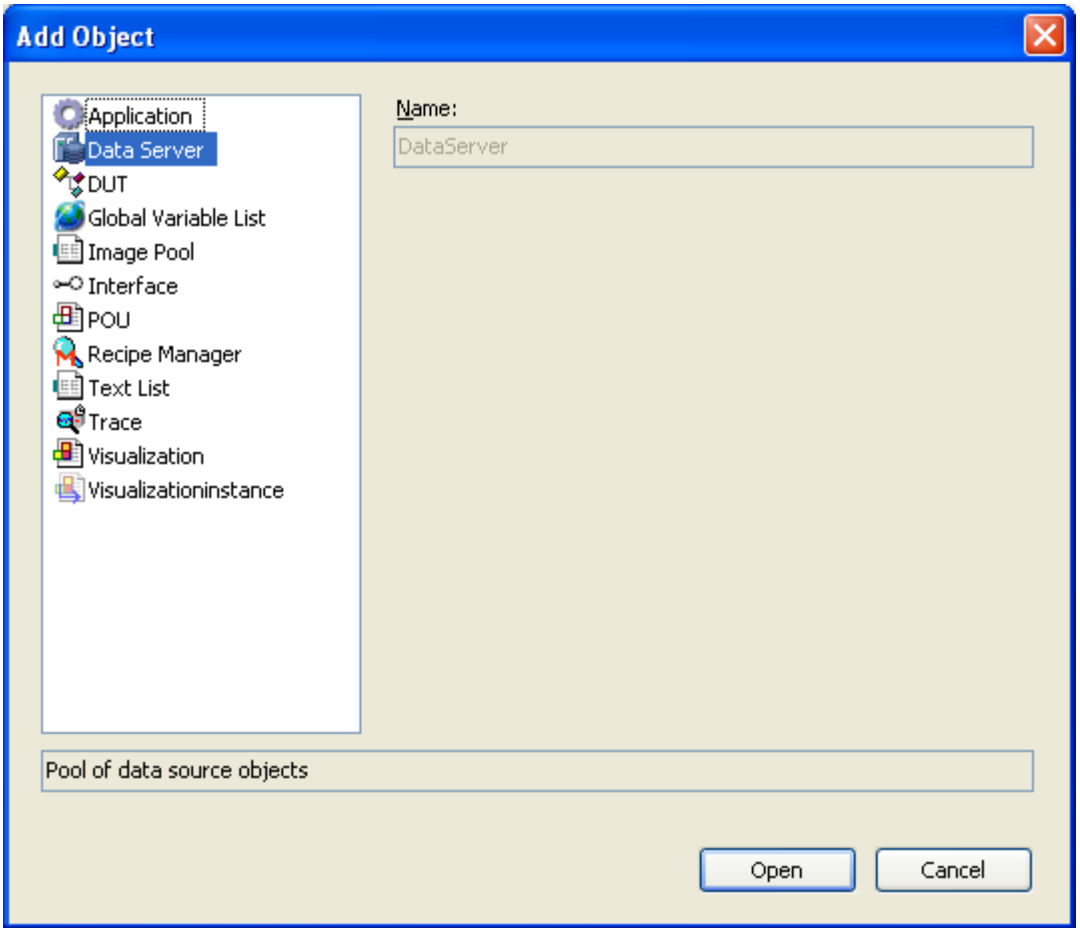


Abb. 355 Add Object dialog, adding a data server

With the Data Server object the required Data Server libraries will automatically be included in the Library Manager and a Data Server task will be added to the Task Configuration.

Below the Data Server **data source** objects can be inserted by command Add Object - Data Source.

13.5.2 Add data source

A data source can be added to the data server of an application by the Add Object' command.

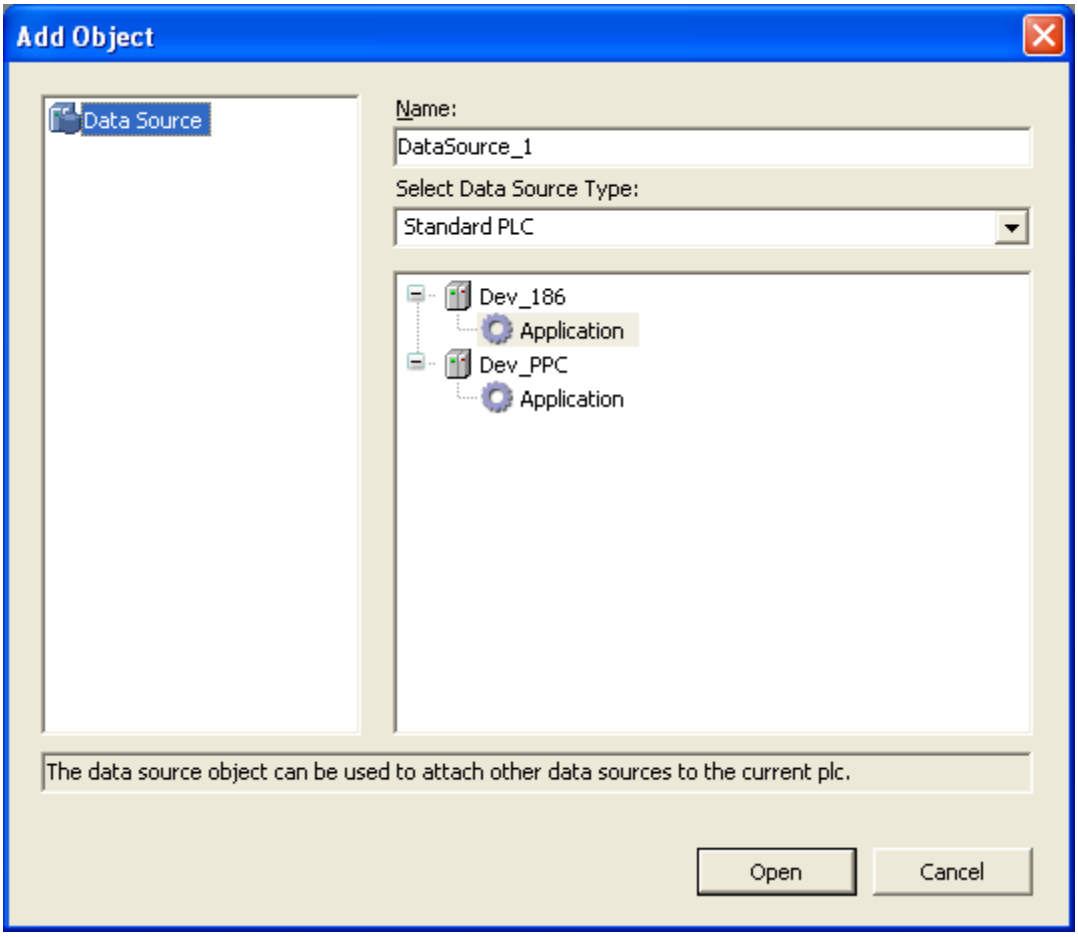


Abb. 356 Add Object dialog for data source selection

The Add Object dialog provides a selection list (**Select Data Source Type:**) where you can choose the desired type of data sources: currently available: "Standard PLC" for 3S devices; not yet implemented: OPC Server.

In the window below the selection list the currently available data sources of the chosen type will be displayed. Select the desired entry and define a symbolic name for the data source in the **Name** field.

After having confirmed with **Open** the data source object will be added - with the given name - in the devices tree below the application.

Example:

The devices tree currently contains three Device entries: Device1, Device2, MyPlc.

Application App1 below MyPlc wants to use the data of Device1 and Device2. So below App1 first enter a data server object and below that a data source object. Choose data source type '3S PLC" in the Add Object dialog. Device1/Application<xy> and Device2/Application<xy> will be offered. Choose the application of Device2, enter Name "DataSource_Dev1" resp. "DataSource_Dev2" and close the dialog. The data source objects will be added in the Devices tree below App1 and now can be configured.

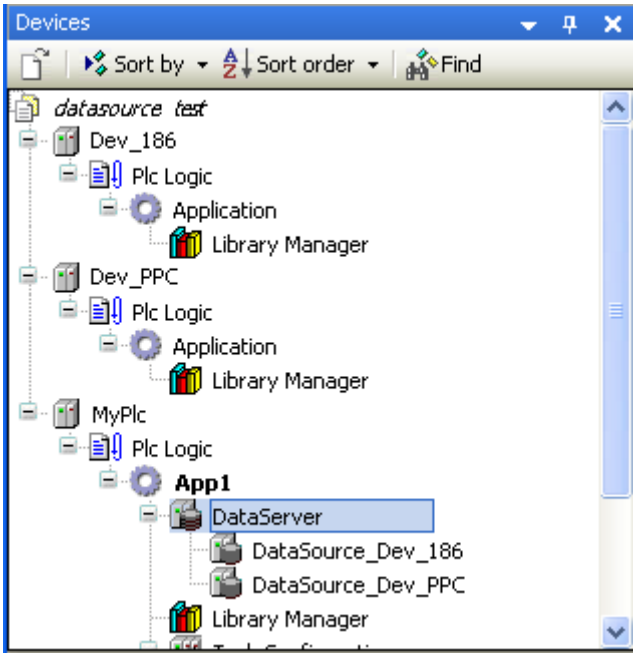


Abb. 357 Example, data server and data sources

13.5.3 Data Source configuration

The data source objects added to a data server can be configured concerning the selection of the data items and the communication with the real data source:

Open the configuration dialog via command 'Edit Object' or by a double-click on the data source object in the devices tree.

The dialog provides two tabs: Data Source Items, Communication.

Data Source Items

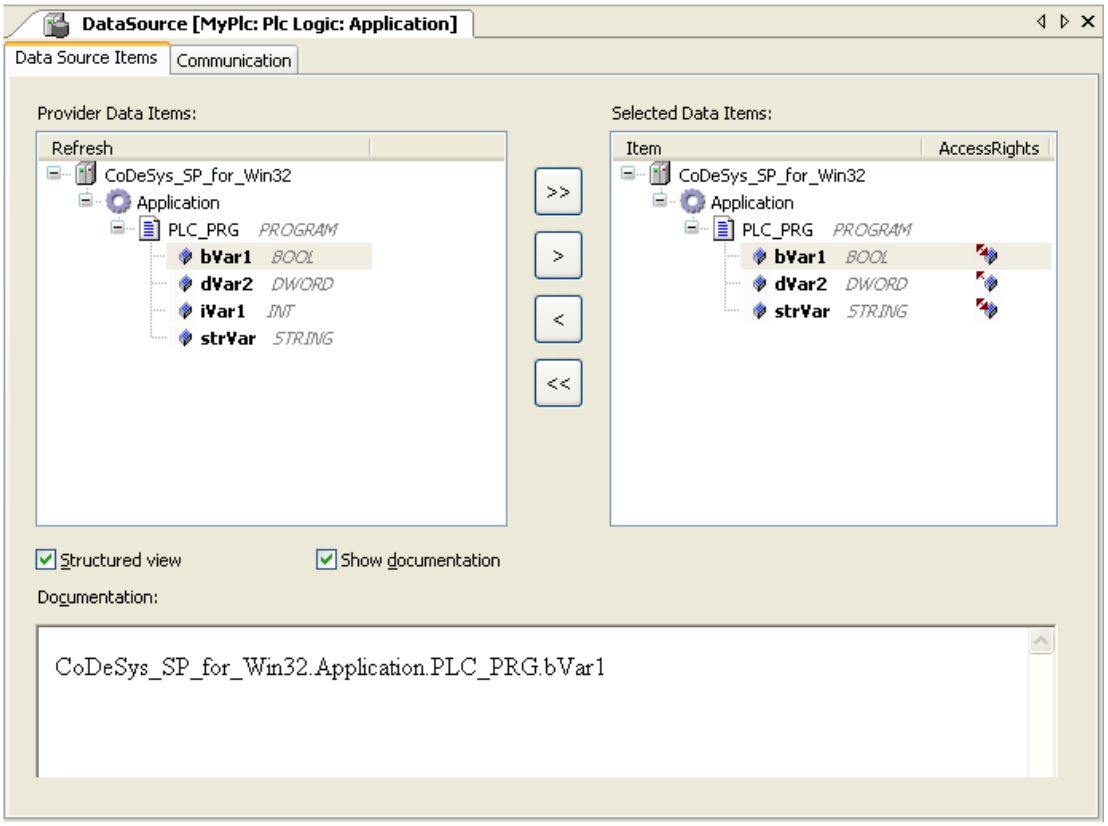


Abb. 358 Data Source Items dialog

Here you define which data items (variables) of the remote data source should be usable by the local application and which access rights should be valid.

Note: Bit accesses used in visualizations will only work over a Data Server connection if the bit index is defined literally (and not by a defined constant).

The variables provided by the chosen data source are displayed in the **Provider Data Items** field. The **Refresh** button at top of this field can be used to update the list of items. The variables will only be added to the list if code has been created before for the related application!

If option **Structured view** is activated, the items will be listed in a tree structure, otherwise in a "flat" list. Below node 'Application' you find the POUs used by the data source application and their variables. If function blocks and data types are instanciated, those and their member variables will be displayed below node 'Datatypes'. Attention: if non-structured display is active, the datatypes will not be listed!

If option **Show documentation** is activated, in the **Documentation** field a description for the item will appear, if any is provided by the data source.

Selection of the variables (data source items) which should be available for the current application:

Basically the items are selected in the 'Provider Data Items' field and either via a double-click or via button > (single) or >> (all) can be inserted in the **Selected Data Items** field in the right part of the dialog. Vice versa selected items correspondingly can be removed by the < and << buttons..

In the structured view concerning **function block** and **data types** there is the possibility to reduce the **selection** on particular member variables. Proceed as follows:

In the 'Provider Data Items' field under node 'Application' select the desired variables. If instances of function blocks or data types are involved, automatically all member variables of those POUs will be published. If this is not desired, the finally to be published items can be reduced to particular POU members by selecting those explicitly under node 'Datatypes'.

This configured selection will immediately be usable by the application.

Access rights:

By default each item is 'readable and writable', see the **AccessRights** column. To modify the access right to 'read only', click in the respective AccessRights field to get into edit mode and then click again to toggle the access right symbol.

Symbol 'read and write':

Symbol 'read only':

The meaning of the access rights of a data item, if option **Update by default** is activated in the eponymous column:

read: the item will be updated in the current application if it has changed in the data source

write: the item will be updated in the data source if it has changed in the current application

Note: By default the selected data items will be updated only in the currently active visualization. If an item is also used in another POU of the application it has to be activated explicitly by using the functions of the Data Server Interface (Currently please contact your software supplier in this case).

Communication

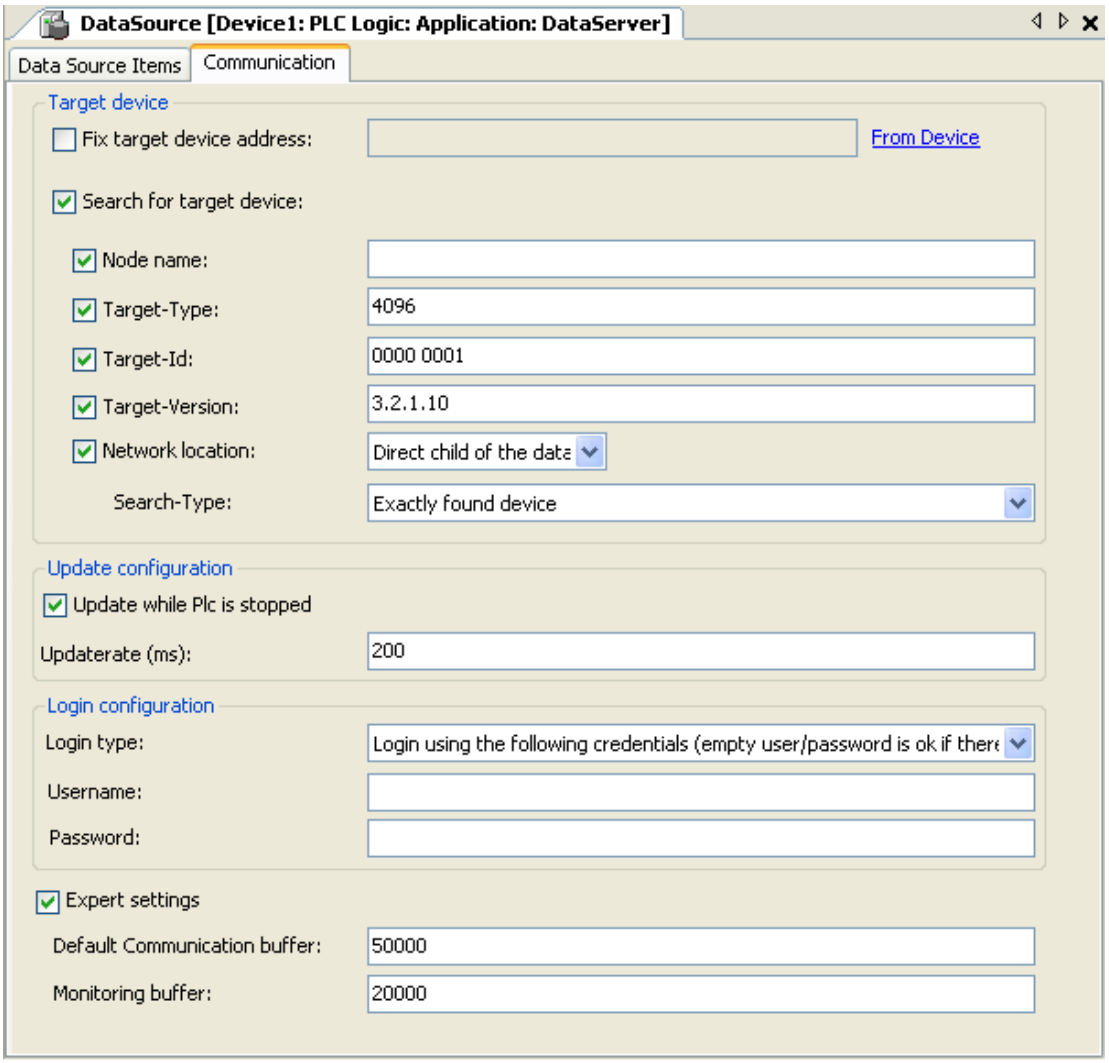


Abb. 359 Communications dialog

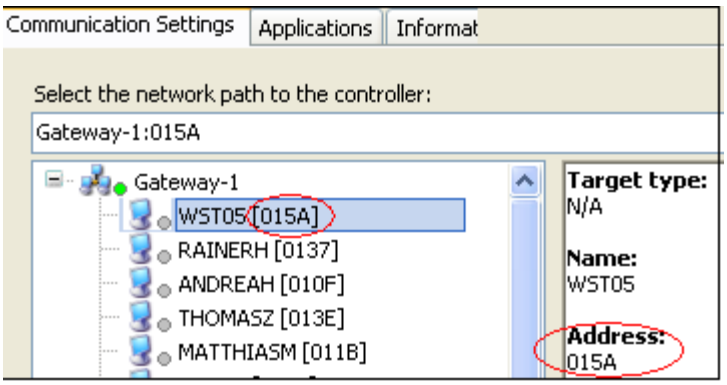
Here the parameters for the communication with the respective remote data source are configured. Regard the note above.

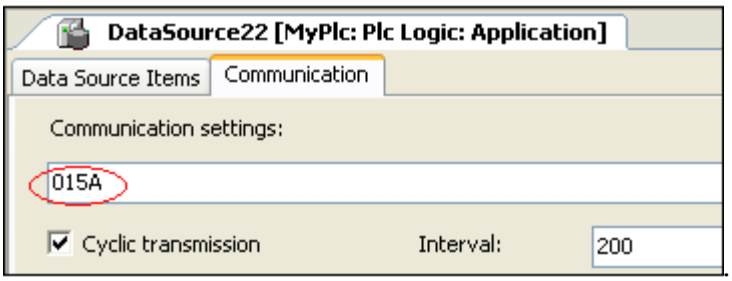
1. Specify the data source device : Either a fix address is defined or criteria for an current search for an available device.

Target device

By a mouse-click on '**From Device**' the parameters of the currently configured device will be entered automatically, as configured in the devices' Communications dialog.

Fix target device address: If this option is activated, the communication will be done via the specified address. So enter the address of the data source device. Enter here the address of the desired data source device, either manually or using the automatic presetting by 'From Device' (see above). Example Example, Entering the address of the data source device





Search for target device: Alternatively to the use of a fix address (see above) this option can be activated, if a currently available device should be searched according to certain search criteria. Activate the desired criteria by setting a checkmark in the option box:

Node name: Node name, e.g. WST06

Target-Type: Type number of the device, e.g. "4096"

Target-Id: e.g. "0000 0001"

Target-Version: e.g. "1.0.0.0"

Network location: Chose where to look for the target device from the selection list:

- Direct child of the data server plc:
  In this case the runtime system running the data server must implement a router component that is accessible from IEC. In this case the device description file of the device where the data server is located on must additionally include the compilerdefine CMPROUTER_AVAILABLE to signal that the router library can be used.
- Direct child of node with address:
  Insert the address of the father in the edit field appearing to the right.

- Direct child of the data server plc or of the node with address:
  This selection combines the two first options. Please insert the address of the father in the edit field appearing to the right and pay attention to the requirements on the router described under the first option.

Search-Type: From the selection list choose the type of search:

- First found device: The first device which matches the defined criteria, will be used.
- Exactly found device: Only a device exactly matching the specified criteria, will be used.

2. Set the communication parameters

Login configuration

Login configuration: From the selection list choose one of the following options how the login to the target device should be handled resp. which credentials will be necessary:

- Login using the following credentials (empty user/password is ok if there is no user management on the PLC): You must enter the credentials which are required by the target device in the following fields Username and Password If the target has no user management the fields can be left empty.
- Login using the credentials determined at runtime: This option should be activated, if the user at runtime will be asked via a login dialog in a visualization for entering the appropriate credentials.
- Omit the login to the device: This option should be activated if you are using an old target device without any user management, so that no login procedure is necessary.

Update configuration

Update while PLC is stopped: If this option is activated, the data source items will be updated according to the specified update rate even if the PLC is currently stopped. Otherwise no update will be done in stop status.

Update rate (ms): Update rate for the data source items in milliseconds. Default: 200 ms.

Expert settings: If this option is activated, the dialog will get extended by the following settings:

Default Communication buffer: The buffer size for the communication data in Bytes. Default: 50000.

Monitoring buffer: The buffer size for the monitoring data in Bytes. Default: 20000.

13.5.4 Use data sources in visualizations

All visualizations assigned to an application can use all data sources which are also assigned to that application.

The data items (variables) of the data source can be used in the properties configuration of a visualization. They might be entered via the Input Assistant or "List Components" function, but also manually. If entered manually, the complete identifier path must be typed in according to syntax: <name of data source device>.<POU name>.<variable name>.

Example:

See below the Properties dialog for a visualization element. As text variable variable "iCount" from a defined data source is used. This data source is "Device2", containing application "App1" with POU "prg1".

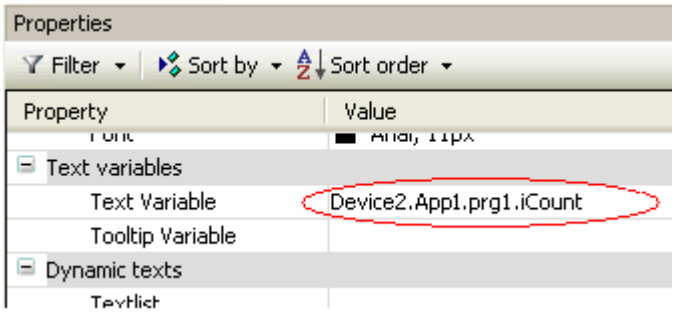


Abb. 360 Example input of data source item in visual element properties

Note: Bit accesses used in visualizations will only work over a Data Server connection if the bit index is defined literally (and not by a defined constant).

Note: The Diagnosis Visualization - because it is only designated to inspect the current application - in contrast to the Target-Visualization will not monitor the current values of variables which are provided via the Data Server from another data source! For those just the initial values will be displayed.

Note: By default the selected data items always will be updated only in the currently active visualization. If an item is also used in another POU of the application it has to be activated explicitly by using the functions of the Data Server Interface.

13.6 Declaration Editor

The **textual** declaration editor is provided by plug-in 'Variable Declaration Object' and serves to create the declaration part of a POU object. With plug-in 'TablularDeclarationEditor' it can be supplemented by a **tabular** view. Any modification made in one of the views always is immediately applied to the other one.

Depending on the current settings in the Declaration editor options either only the textual or only the tabular view will be available, or you can switch between both via buttons (**Textual** / **Tabular**) at the right border of the editor window.

Usually the declaration editor is used in combination with the programming language editors, that means it will be placed in the upper part of the window which opens when you are going to edit or view (monitor) an object in offline or online mode. The declaration header describes the POU type (for example PROGRAM, FUNCTION_BLOCK, FUNCTION ...) and might be extended by POU-global pragma attributes.

The online mode of the declaration editor is structured like that of a watch view.

Variable declaration also is done in for example Global Variable Lists and Data Unit Types, which however are created in separate editors.

Regard the general information on variables' declaration.

Textual declaration editor

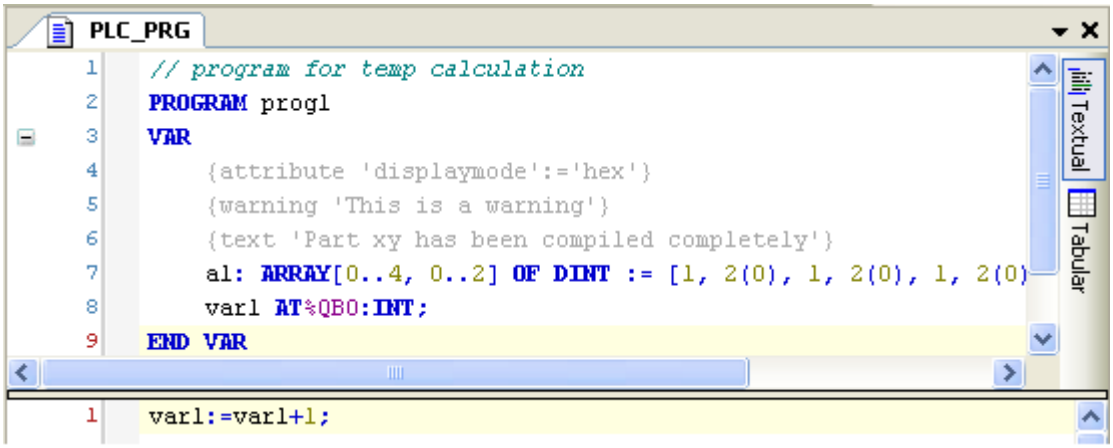


Abb. 361 Textual editor view

Behaviour and appearance are determined by the respective current text editor **settings** in the Options and Customize dialogs. There you can define the default settings for highlight coloring, line numbers, tabs, indenting and many more. The usual Windows functions are available and even those of the IntelliMouse can be used if the corresponding plug-in is installed. Notice that **block selection** is possible by pressing <Alt> while selecting the desired text area with the mouse.

Tabular declaration editor

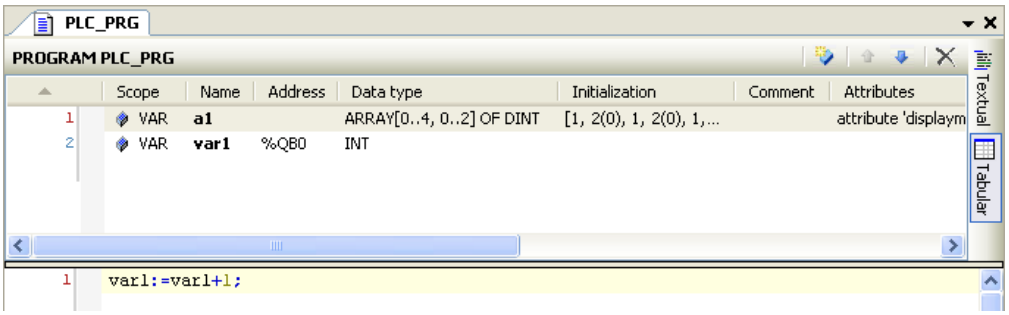


Abb. 362 Tabular editor view

The tabular view of the editor provides columns for the usual definitions required for a variables' declaration: **Scope**, **Name**, **Address**, **Data type**, **Initialization**, **Comment** and (pragma) **Attributes**. The particular declarations are inserted as numbered lines.

The declaration header can be edited in the **Edit Declaration Header** dialog which opens on the same-named command.

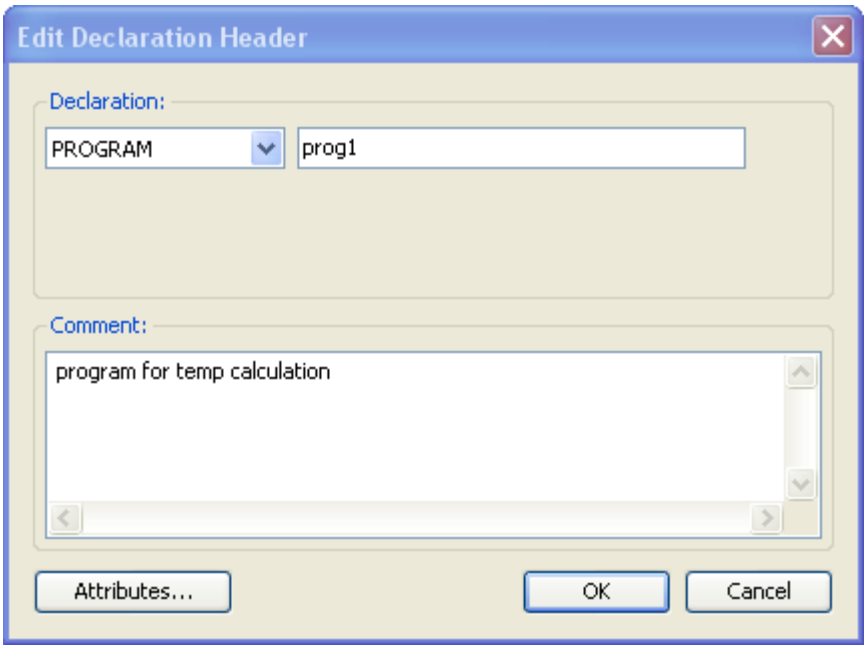


Abb. 363 Edit Declaration Header dialog

Attributes dialog for specifying pragma instructions and attributes.

To **add a new line** of declaration above an existing one, first select this line and use command **Insert** from the toolbar or the context menu. To add a new declaration at the end of the table, click beyond the last existing declaration line and also use the Insert command.

The newly inserted declaration by default first uses scope "VAR" and the recently entered data type. Automatically the input field for the obligatory variables' **name** will be opened where you must enter a valid identifier and close with <Enter> or by a mouse-click on another part of the view.

Each table cell on a **double-click** opens the respective possibilities to enter a value:

For editing the **Scope**, the double-click will open a list from which you can choose the desired scope and scope attribute keyword (flag).

The **Data type** can be typed in directly or via the › button the Input Assistant or the Array wizard can be used.

The **Initialization** value can be typed in directly or via the … button the **Initialization Value dialog** can be used, which is helpful for structured variables.

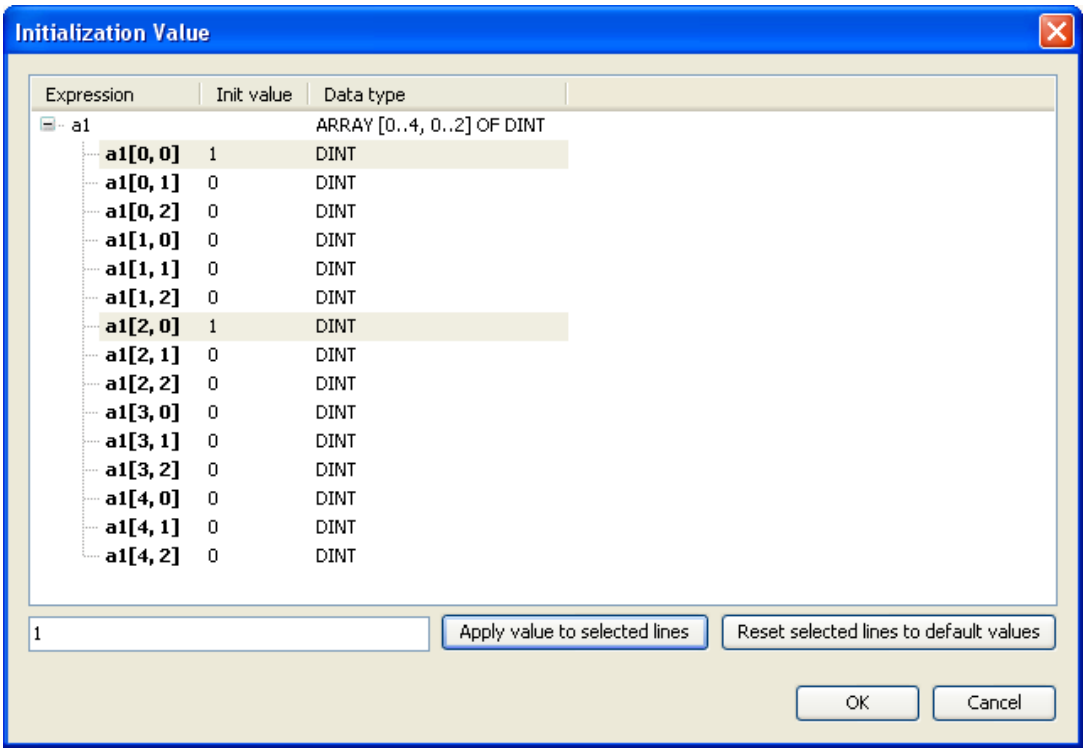


Abb. 364 Initialization Value dialog

All **expressions** of the variable will be displayed with the current **init values**. Select the desired ones and edit the initialization value in the field below the listing. Then use button **Apply value to selected lines**. To restore the default initializations, use button **Reset selected lines to default values**.

Line breaks in the **Comment** entry can be inserted via Ctrl+Enter.

The **Attributes** entries are done in the **Attributes dialog** where multiple attributes and pragmas can be entered in text format. They have to be inserted without enclosing {} braces, a separate line per each. For the example shown in the following image see the corresponding textual view above in image "Textual editor view".

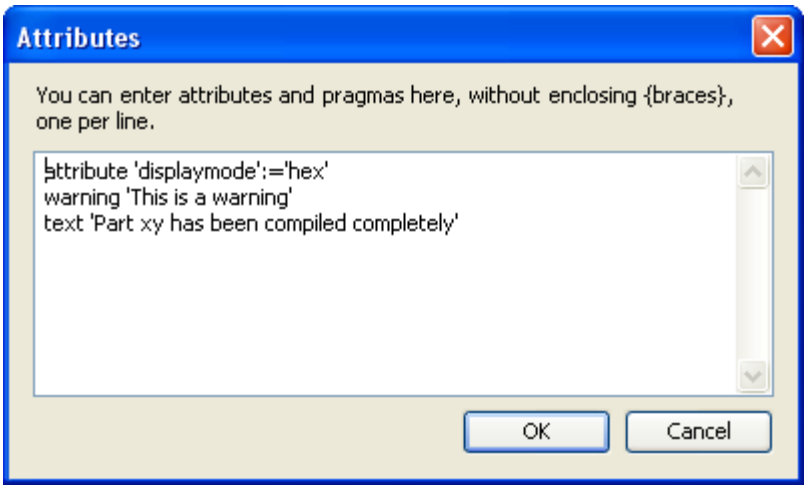


Abb. 365 Attributes dialog, example

Each variable is declared in a separate line, the **lines are numbered**.

You can **change the order** of lines (line numbers) by selecting a line and move it one up or down by the **Move up** resp. **Move down** command from the toolbar or the context menu.

The list of declarations can be **sorted** according to each of the columns by a mouse-click on the header of the respective column. The column which currently determines the order, is indicated by an arrow symbol: (ascending order) or (descending order). Each further click in the column header changes between ascending and descending order.

To **delete one or several declarations**, select the respective lines and use <Del> or the **Delete** command from the context menu or from the toolbar ()

Declaration Editor in online mode

After log-in to the target system each object which has been opened in a window already in offline mode, will automatically be displayed in online view.

The online view of the Declaration Editor presents a table like used in watch views. The header line shows the actual object path "<device name>.<application name>.<object name>". The table for each watch **Expression shows the Type** and current **Value** as well as - if currently set - a **Prepared value** for forcing or writing.

To appoint a prepared value to a variable you may either use the Prepare value dialog or you may click in the assigned field of the column **Prepared value** and directly type in the desired value.
In case of a boolean variable the handling is even easier: You can toggle boolean preparation values by use of the **Return** or **Space** key according to the following order: If the value is TRUE, the preparation steps are FALSE -> TRUE -> nothing; else, if the value is FALSE, the preparation steps are TRUE -> FALSE -> nothing.

If a watch expression (variable) is an instance, for example of a function block, a plus- resp. minus-sign is preceded. Usually via a mouse-click on this sign the particular elements of the instanced object can be additionally displayed (see fbinst in the following image) resp. hidden (see fbinst2). Icons indicate whether the variable is an input, output or a "normal" one.

As long as the default setting Replace Constants (Compile Options) is active, **constants** are indicated by an symbol preceding the value in the Value column.

When pointing with the cursor on a variable in the implementation part, a **tooltip** will show the declaration and comment of the variable. See the following picture:

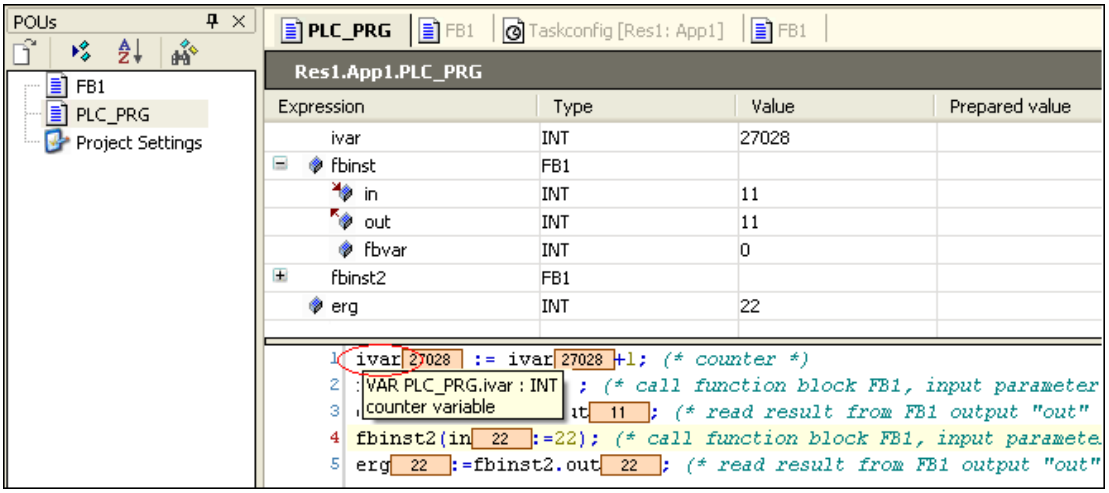


Abb. 366 Example: Declaration Editor in the upper part of a program object PLC_PRG, online view

13.7 DUT Editor

User defined data types can be created in the Data Unit Type editor (DUT editor). This is a text editor and behaves according to the currently set text editor options.

The DUT editor will be opened automatically in a window when adding a DUT object in the Add object dialog. In this case it provides by default the syntax of an extended structure declaration, which then might be changed as desired to a simple structure declaration or to the declaration of another data type unit, for example an enumeration.

The editor also opens when you open an existing DUT object currently selected in the POUs view.

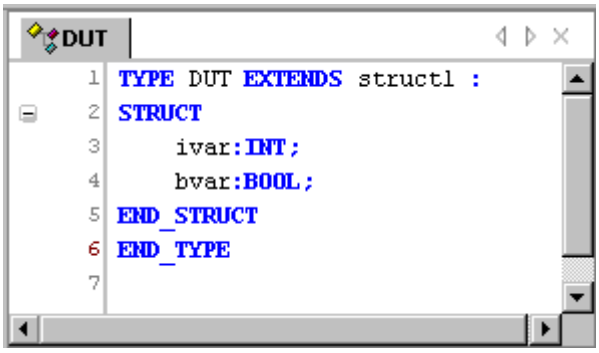


Abb. 367 DUT editor window

13.8 FBD/LD/IL Editor

A **combined editor** is available for editing POUs in the languages FBD (Function Block Diagram), LD (Ladder Diagram) and IL (Instruction List).

This means a **common set of commands and elements** is used and an automatic internal conversion between the three languages is done. In offline mode the programmer always can **switch** to one of the other editor views (View as...).

Notice anyway, that there are some special elements, which **cannot get converted** and thus will only be displayed in the appropriate language. Also there are some constructs which cannot get converted unambiguously between IL and FBD and therefore will be "normalized" at a conversion back to FBD. This concerns: Negation of expressions and explicit/implicit output assignments.

The FBD/LD/IL Editor is available via the 'Network List Editor' plug-in. **Behaviour, look and menus** are defined by the settings in the Customize and Options dialogs. See there especially for the options concerning the display of comments and addresses.

The editor will open in a **bipartite window**, when editing an object programmed in FBD/LD/IL, the upper part containing a **declaration editor**.

The **programming language** for a new object is specified when creating the object via command 'Add Object'.

13.8.1 Languages

Function Block Diagram - FBD

The Function Block Diagram is a graphically oriented programming language. It works with a list of networks whereby each network contains a graphical structure of boxes and connection lines which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

The FBD/LD/IL Editor is provided by the 'Network List Editor'-plug-in.

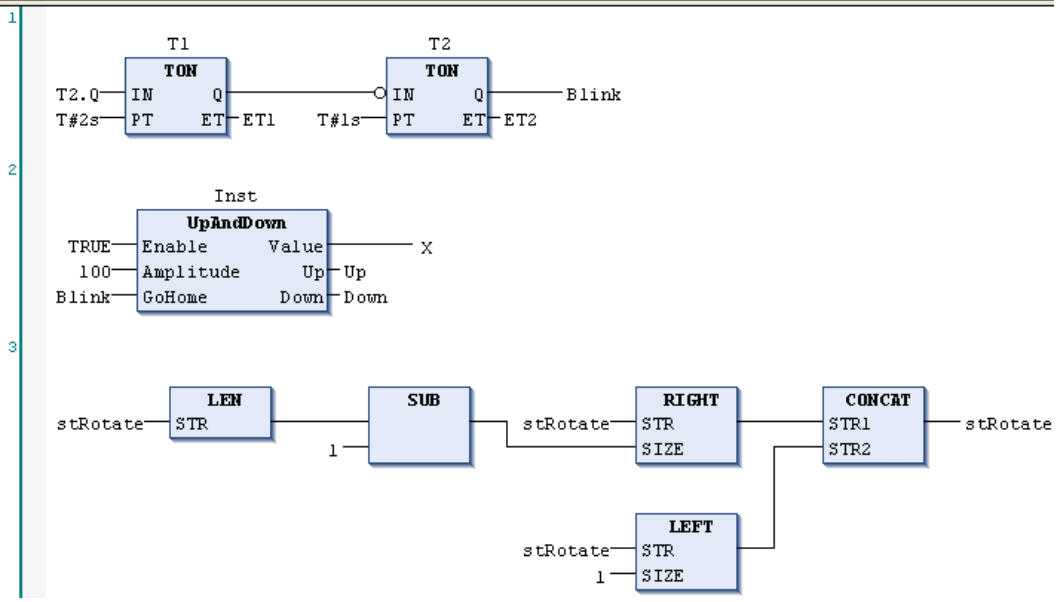


Abb. 368 Example of some Function Block Diagram networks

Ladder Diagram - LD

The Ladder Diagram is a graphics oriented programming language which approaches the structure of an electric circuit.

On the one hand, the Ladder Diagram is suitable for constructing logical switches, on the other hand one can also create networks as in FBD. Therefore the LD is very useful for controlling the call of other POUs.

The Ladder Diagram consists of a series of networks, each being limited by vertical current lines (**power rail**) on the left and on the right. A network contains a circuit diagram made up of contacts, coils, optionally additional POUs (boxes) and connecting lines. On the left side there is a series of contacts passing from left to right the condition "ON" or "OFF" which corresponds to the Boolean values TRUE and FALSE. To each contact a Boolean variable is assigned. If this variable is TRUE, the condition will be passed from left to right along the connecting line. Otherwise value OFF will be passed ON. Thus the coil(s), which is/are placed in the right part of the network, receive(s) an ON or OFF value coming from left and correspondingly copy TRUE or FALSE to an assigned boolean variable.

The FBD/LD/IL Editor is provided by the 'Network List Editor'-plug-in.

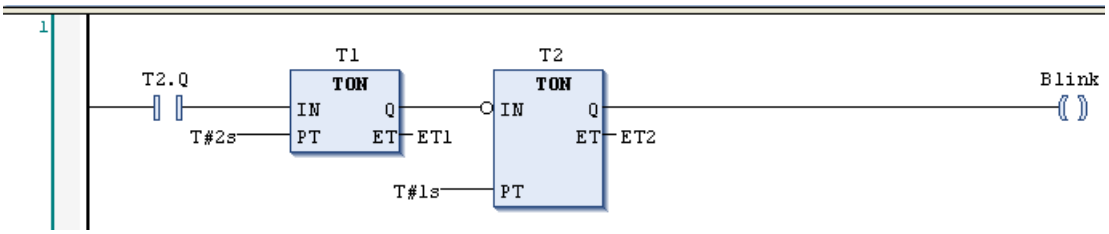


Abb. 369 Example of a Ladder Diagram network

Instruction List - IL

The Instruction List is an assembler-like IEC 61131-3 conformal programming language.

This language supports programming based on an accumulator. All IEC 61131-3 operators are supported as well as multiple inputs / multiple outputs, negations, comments, set / reset of outputs and unconditional / conditional jumps.

Each instruction is primarily based on the loading of values into the accumulator by using the LD operator. After that the operation is executed with the first parameter taken out of the accumulator. The result of the operation again is available in the accumulator, from where the user should store it with the ST instruction.

In order to program conditional executions or loops IL supports both comparing operators like EQ, GT, LT, GE, LE, NE and jumps. The latter can be unconditional (JMP) or conditional (JMPC / JMPCN). For conditional jumps the accumulator's value is checked on TRUE or FALSE.

Syntax:

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

Each instruction begins in a new line and contains an **operator** and, depending on the type of operation, one or more **operands** separated by commas. The operator might be extended by a **modifier**.

In a line before an instruction there can be an identification *mark* (**label**) followed by a colon (:), for example "ml:" in the example shown below. A label can be the target of a **jump** instruction, for example "JMPC next" in the example shown below.

A **comment** must be placed as last element of a line.

Empty lines can be inserted between instructions.

```
PROGRAM IL
VAR
    instl: TON;
    dwVar: DWORD;
    dwRes: DWORD;
    tl: TIME;
    toutl: TIME;
    inst2: TON;
    bVar: BOOL;
END_VAR
```

```
1

LD      bVar                                    variable
ST      instl.IN        starts timer with risin…
JMPC    ml
CAL     instl(
        PT:=tl,
        ET:=>toutl)
LD      instl.Q         is TRUE, PT seconds aft…
ST      inst2.IN        starts timer with risin…

2

 ml:
LD      dwVar
ADD     230
ST      dwRes
```

Abb. 370 IL program example in IL table editor

The IL Editor is a table editor integrated in the FBD/LD/IL editor provided by the 'Network List Editor'-plug-in.

13.8.2 Modifiers and operators in IL

The following modifiers can be used in Instruction List:

C	with JMP, CAL, RET:	The instruction only will be executed if the result of the preceding expression is TRUE.
N	with JMPC, CALC, RETC:	The instruction will only be executed if the result of the preceding expression is FALSE.
N	otherwise:	Negation of the operand (not of the accumulator)

The following table shows which operators can be used in combination with the specified modifiers:

The "**accumulator**" always stores the current value,resulting from the preceding operation.

Operator	Modifiers	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 into the operand variable	ST iErg
S		Sets the operand (type BOOL) to TRUE when the content of the accumulator is TRUE	S bVar1
R		Sets the operand (type BOOL) to FALSE when the content of the accumulator is TRUE	R bVar1
AND	N,(	Bitwise AND of the accumulator and the (negated) operand	AND bVar2
OR	N,(	Bitwise OR of the accumulator and the (negated) operand	OR xVar
XOR	N,(	Bitwise exclusive OR of the accumulator and the (negated)operand	XOR N,(bVar1,bVar2)
NOT		Bitwise negation of the accumulator's content	
ADD	(	Addition of accumulator and operand, result is copied to the accumulator	ADD (iVar1,iVar2)
SUB	(	Subtraction of accumulator and operand, result is copied to the accumulator	SUB iVar2
MUL	(	Multiplication of accumulator and operand, result is copied to the accumulator	MUL iVar2
DIV	(	Division of accumulator and operand, result is copied to the accumulator	DIV 44
GT	(	Check if accumulator is greater than operand, result (BOOL) is copied into the accumulator; >	GT 23

Operator	Modifiers	Meaning	Example
GE	(	Check if accumulator is greater than or equal to the operand, result (BOOL) is copied into the accumulator; >=	GE iVar2
EQ	(	Check if accumulator is equal to the operand, result (BOOL) is copied into the accumulator; =	EQ iVar2
NE	(	Check if accumulator is not equal to the operand, result (BOOL) is copied into the accumulator; <>	NE iVar1
LE	(	Check if accumulator is less than or equal to the operand, result (BOOL) is copied into the accumulator; <=	LE 5
LT	(	Check if accumulator is less than operand, result (BOOL) is copied into the accumulator; <	LT cVar1
JMP	CN	Unconditional (conditional) jump to the label	JMPN next
CAL	CN	(Conditional) call of a PROGRAM or FUNCTION_BLOCK (if accumulator is TRUE)	CAL prog1
RET		Early return of the POU and jump back to the calling POU	RET
RET	C	Conditional - if accumulator is TRUE,)early return of the POU and jump back to the calling POU	RETC
RET	CN	Conditional - if accumulator is FALSE)early return of the POU and jump back to the calling POU	RETCN
)		Evaluate deferred operation	

See also the help pages on IEC operators.

See also the help page on Working in the IL editor view for how to use and handle **multiple operands**, **complex operands**, function / method / function block / program / action **calls** and **jumps**.

See in the following picture an example program using some **modifiers**:

```
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   m1:

    LD      bVar2   store value of bVar2...
    ST      bRes    ... in bRes
```

Abb. 371 Example of an IL program while using some modifiers

13.8.3 Working in the FBD and LD editor view

"**Networks**" are the basic entities in FBD and LD programming. Each network contains a structure that displays a logical or an arithmetical expression, a POU (function, program, function block call, etc.), a jump, a return instruction.

When creating a new object, the editor window automatically contains one empty network.

Notice the general editor settings in the options dialog, category 'FBD, LD and IL'.

Tooltip containing information on variables or box parameters

The cursor being placed on the name of a variable or box parameter will prompt a tooltip showing the respective type and in case of function block instances the initialization value. For IEC operators SEL, LIMIT, MUX a short description on the inputs will appear. If defined, also the address and the symbol comment will be shown as well as the operand comment (in quotation marks in a second line).

Inserting and arranging elements:

- The **commands for working in the editor** by default are available in the FBD/LD/IL menu, the most important always also in the context menu. It always depends on the current cursor position resp. the current selection (multiselection possible) which elements currently can be inserted via menu command.
- Elements also can be directly **dragged with the mouse from the** toolb**ox** to the editor window or from one position within the editor to another ("Drag&Drop"). For this purpose select the element by a mouse-click, then keep the mouse-button pressed and drag the element into the respective network in the editor view. As soon as you have reached the network, all possible **insert positions** for the respective type of element will be indicated by grey position markers. When you place the mouse-cursor on one of these positions, the position marker will change to green and you can release the mouse-button in order to place the element at that position.
- The **Cut**, **Copy**, **Paste** and **Delete** commands, by default available in the Edit menu, can be used to arrange elements. Copying an element is also possible by drag and drop: Select the element within a network by a mouse-click and while keeping the mousebutton pressed, drag it to the target position. As soon as that is reached (green position marker), a plus-symbol will be added to the cursor symbol. Release the mouse-button to insert the element.
- For all possible cursor positions see: Cursor positions FBD/LD/IL.

Navigating:

- The **arrow keys** might be used to jump to the next resp. previous cursor position; also possible between networks.
- The **<Tab>** key might be used to jump to the next resp. previous cursor position within a network.
- **<Ctrl>**+<**Home>** scrolls to the begin of the document and marks the first network.
- **<Ctrl>+End>** scrolls to the end of the document and marks the last network
- **<PageUp>** scrolls one screen up and marks the topmost rectangle.
- **<PageDown>** scrolls one screen down and marks the topmost rectangle.

Selecting:

- An element, also network, can be selected via taking the respective cursor position by a mouse-click or using the arrow or tabulator keys
- Multiselection of non-adjacent elements resp. networks is possible by keeping the while selecting the desired elements one after the other.
- In the LD editor multiselection of adjacent elements or networks can be done by keeping pressed the <Shift>-key while selecting two contacts determining start and end of the desired network section. If you want to cut (copy) and paste a section of a network, it will be sufficient to keep the <Ctrl>-key pressed while selecting just two contacts defining the "borders" of this section. Then the elements between will be noticed automatically.

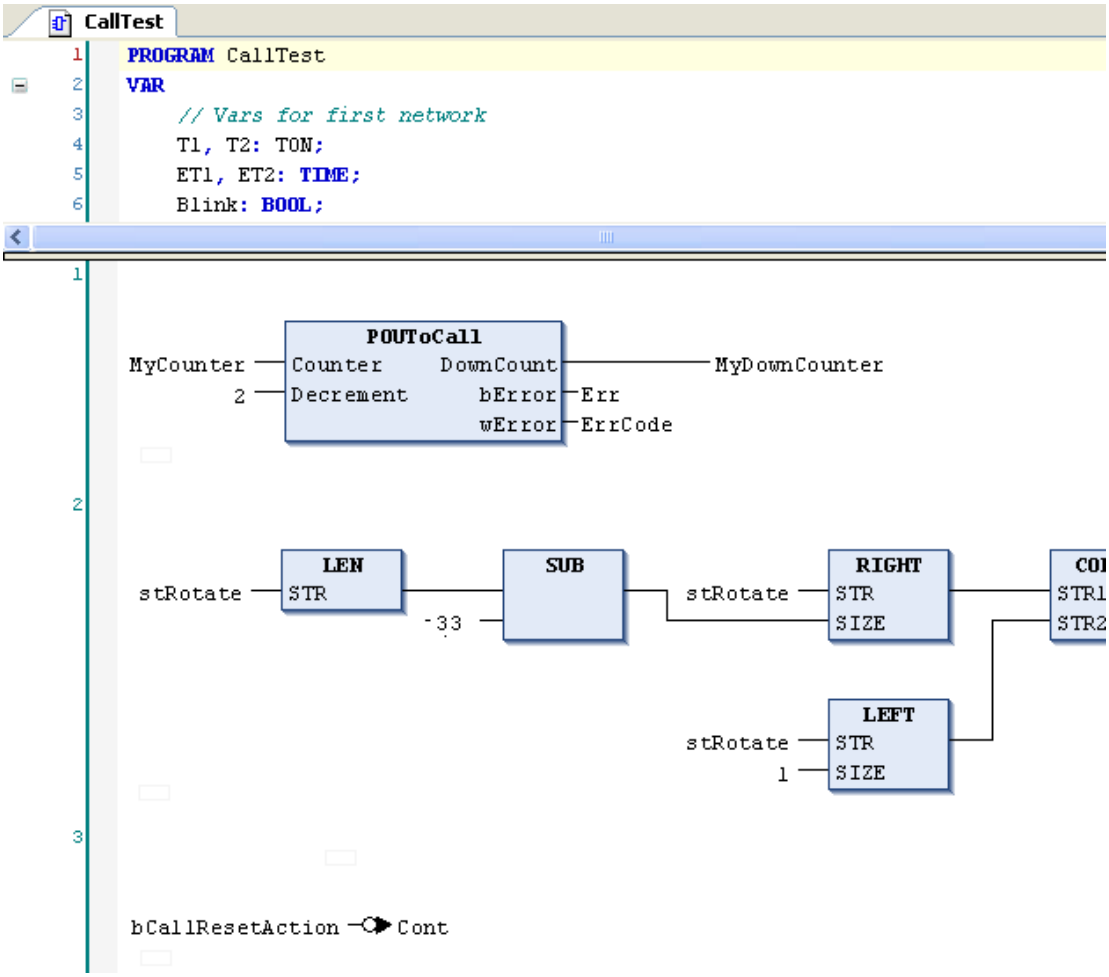


Abb. 372 Example of a FBD editor window

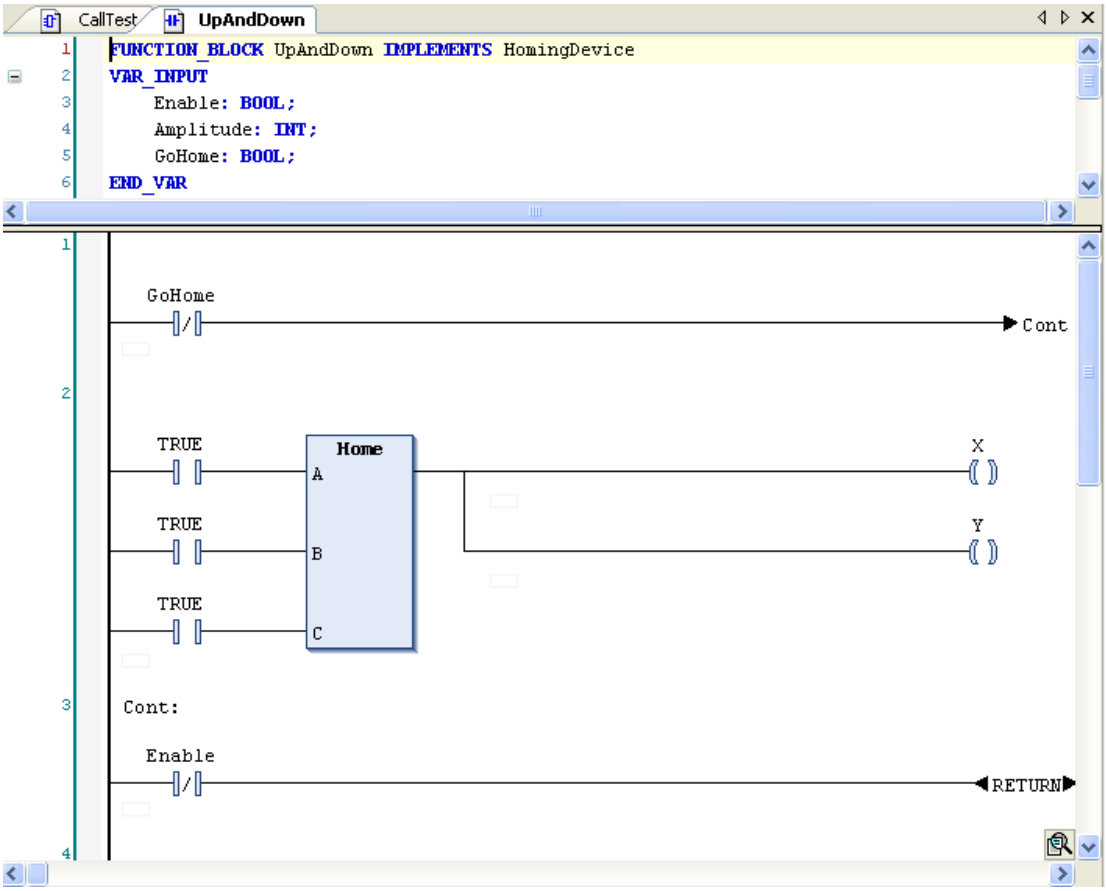


Abb. 373 Example of a LD editor window

For information on the languages see:

Function Block Diagram - FBD

Ladder Diagram - LD

13.8.4 Working in the IL editor view

The IL (Instruction List) editor is a **table editor** in contrast to the pure text editor used in »PLC Designer« V2.3. The **network** structure of FBD or LD programs is also represented in an IL program. Basically one network is sufficient in an IL program, but considering switching between FBD, LD and IL you also might consciously use networks for structuring an IL program.

Notice the general editor settings in the options dialog, category 'FBD, LD and IL'.

Tooltip containing information on variables or box parameters: Please see Working in the FBD and LD editor view.

Inserting and arranging elements:

- The **commands for working in the editor** by default are available in the FBD/LD/IL menu, the most important always also in the context menu.
- Programming units, that is **elements**, are inserted each at the current cursor position via the Insert-commands, by default available in the FBD/LD/IL menu.
- The **Cut**, **Copy**, **Paste** and **Delete** commands, by default available in the Edit menu, can be used to arrange elements.
- Regard the possible cursor positions.
- See some information on the Instruction List language

See in the following how the table editor is structured, how you can navigate in the editor and how to use complex operands, calls and jumps.

Structure of the IL table editor:

Each program line is written in a table row, structured in **fields** by the following table columns:

column	contains..	description
1	Operator	This field contains the IL operator (LD, ST, CAL, AND, OR etc.) or a function name. In case of a function block call here also the respective parameters are specified, in this case in the Prefix field ":=" or "=>" must be entered.
2	Operand	This field contains exactly one operand or a jump label. If more than one operand is needed (multiple/extensible operators "AND A, B, C" or function calls with several parameters), those must be written into the following lines where the operator field is to be left empty. In this case add a parameter-separating comma. In case of a function block, program or action call the appropriate opening and/or closing brackets must be added.
3	Address	This field contains the address of the operand as defined in the declaration part. The field cannot be edited and can be switched on or off via option 'Show symbol address'.
4	Symbol comment	This field contains the comment as defined for the operand in the declaration part. The field cannot be edited and can be switched on or off via option 'Show symbol comment'.
5	Operand comment	This field contains the comment for the current line. It is editable and can be switched on or off via option 'Show operand comment'.

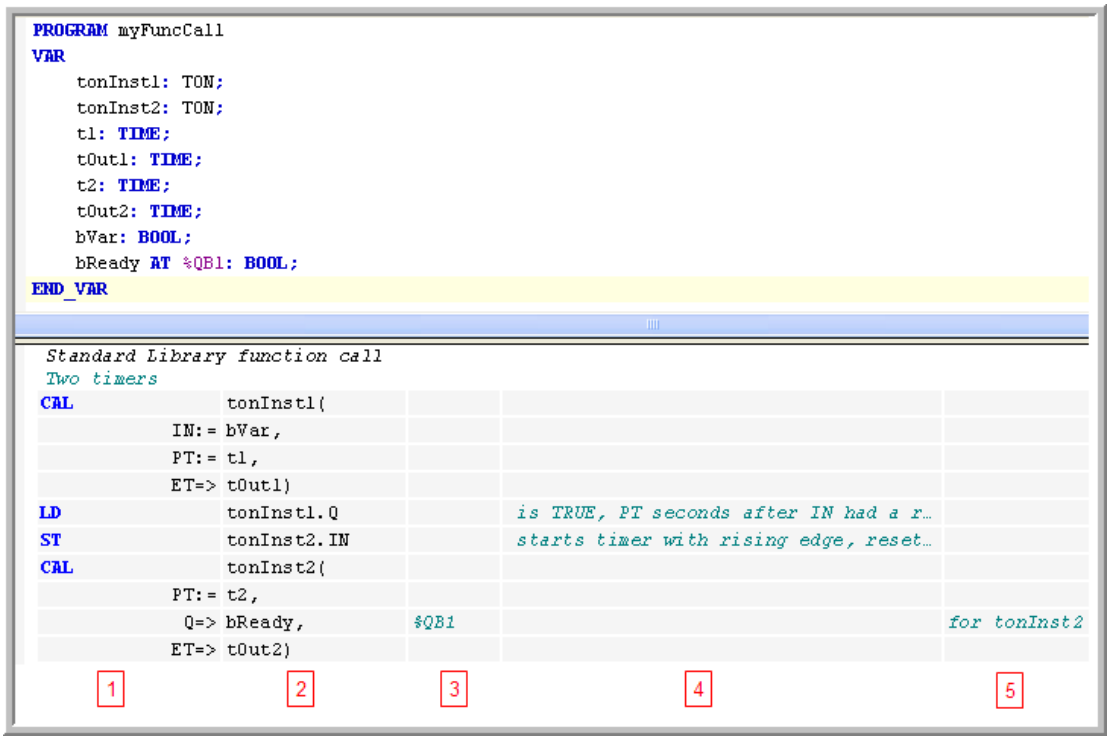


Abb. 374 IL table editor

Navigating in the table:

- **Up**- and **Down** arrow keys: Moving to the field above resp. below
- **<Tab>**: Moving within a line to the field to the right.
- **<Shift>** + **<Tab>**: Moving within in a line to the field to the left.
- **<Space>**: Open the currently selected field for editing. Alternatively perform a further mouse-click on the field. If applicable the input assistant will be available via the ... button. A currently opened edit field can be closed by <Enter>, confirming the current entries, or by <Esc>, cancelling the made entries.
- **<Ctrl>** + **<Enter>**: Enter a new line below the current one.
- **<Delete>**: Remove the current line, that is where you have currently selected any field.
- **Cut**, **Copy**, **Paste**: To copy one or several lines select at least one field of the line(s) and use the copy command. To cut a line, use the Cut command. Paste will insert the previously copied/cut lines before the line where currently a field is selected. If no field is selected they will be inserted at the end of the network.
- **<Ctrl>** + **<Home>** scrolls to the begin of the document and marks the first network.
- **<Ctrl>** + **<End>** scrolls to the end of the document and marks the last network
- **<PageUp>** scrolls one screen up and marks the topmost rectangle.
- **<PageDown>** scrolls one screen down and marks the topmost rectangle.

Multiple operands (extensible operators):

If the same operator is used with multiple operands, two ways of programming are possible:

1. The operands are entered in subsequent lines, separated by commas, example:

```
LD      7
ADD     2,
        4,
        7
ST      iVar
```

2. The instruction is repeated in subsequent lines, example:

```
LD      7
ADD     2
ADD     4
ADD     7
ST      iVAR
```

Complex operands:

If a complex operand is to be used, enter an opening bracket before, then use the following lines for the particular operand components and below those, in a separate line enter the closing bracket.

Example: rotating a string by 1 character at each cycle

(corresponding ST code: stRotate := CONCAT(RIGHT(stRotate, (LEN(stRotate) - 1)), (LEFT(stRotate, 1)));)

```
LD      stRotate
RIGHT(  stRotate
LEN
SUB     1
)
CONCAT( stRotate
LEFT    1
)
ST      stRotate
```

Function calls:

Enter the function name in the operator field. The (first) input parameter is to be given as an operand in a preceding LD operation. If there are further parameters, the next one must be given in the same line as the function name. The further ones can also be added in this line, separated by commas, or in subsequent lines.

The function return value will be stored in the accumulator, but ...

Notice the following restriction concerning the IEC standard: A **function call with multiple return values is not possible**, only one return value can be used for a succeeding operation.

Example: Function GeomAverage, which has three input parameters, is called. The first parameter is given by X7 in a preceding operation, the second one, "25" is given behind the function name. The third one is given by variable tvar, either in the same line or in the subsequent one. The return value is assigned to variable 'Ave':

(corresponding code in Structured Text: Ave := GeomAverage(X7, 25, tvar);)

```
LD          X7
GeomAverage 25
ST          Ave
```

Function block calls, program calls:

Use the CAL- or CALC operator. Enter the function block instance name resp. the program name in the operand field (second column) followed by the opening bracket. Enter the input parameters each in one of the following lines:

First column: operator (parameter name) and „:=“ for input parameters resp. „=>“ for output parameters

Second column: operand (actual parameter) and „,“ if further parameters follow resp. „)“ after the last parameter resp. in case of parameter-less calls „()“.

Example: Call of POUToCAll with two input and two output parameters

(corresponding ST code: POUToCall(iCounter := MyCounter, iDecrement:=1000, bError=>bErr, wError=>wResult);)

```
1  PROGRAM IL_EXAMPLE
2  VAR
3      bErr: BOOL;
4      wResult: WORD;
5  END_VAR
6
```

```
1
    CAL         POUToCall(
          iCounter:= 1,
        iDecrement:= 1000,
            wError=> wResult)
    LD          POUToCall.bError
    ST          bErr
```

It is not necessary to use all parameters of a function block or program.

Note: As a restriction to the IEC standard complex expressions cannot be used, those must be assigned to the input of the function block or program before the call instruction.

Action call:

To be done like a function block or program call. The action name is to be appended to the instance name or program name.

Example: Call of action ResetAction (corresponding ST code: Inst.ResetAction();)

```
CAL     Inst.ResetAction()
```

Method call:

To be done like a function call. The instance name with appended method name is to be entered in the first column (operator).

Example: Call of method Home (corresponding ST code: Z := IHome.Home(TRUE, TRUE, TRUE);)

```
LD      TRUE
IHome.Home  TRUE,
        TRUE
ST      Z
```

Jump:

A jump is programmed by "JMP" in the first column (operator) and a label name in the second column (operand). The label is to be defined in the target network in the label field. Remark that the statement list preceding the unconditional jump has to end with one of the following commands: ST, STN, S, R, CAL, RET or another JMP. This is not the case for a conditional jump being programmed by "JMPC" in the first column (operator) instead of "JMP". The execution of the jump depends on the value loaded.

Example: Conditional jump instruction; in case bCallRestAction is TRUE, the program should jump to the network labeled with "Cont".

```
LDN     bCallResetAction
JMPC    Cont
```

13.8.5 Cursor positions in FBD, LD and IL

IL editor:

This is a text editor, structured in form of a table. Each table cell is a possible cursor position. See also: Working in the IL editor view

FBD and LD editors:

These are graphic editors, see below examples (1) to (11) showing the possible cursor positions: text, input, output, box, contact, coil, line between elements, sub-network, network. Actions like Cut, Copy, Paste, Delete and other editor-specific commands refer to the current cursor position resp. selected element. See: Working in the FBD and LD editor view

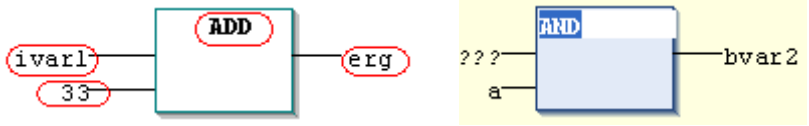
Basically in FBD a dotted rectangle around the respective element indicates the current position of the cursor, additionally texts and boxes get blue- resp. red-shadowed.

In LD coils and contacts get red-colored as soon as the cursor is positioned on.

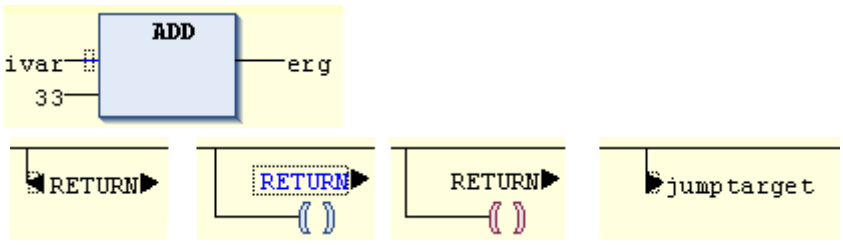
The cursor position determines which elements are available in the context menu for getting inserted.

Possible cursor positions:

(1) Every text field: in the left picture the possible cursor positions are marked by a red-frame, the right picture shows a box with the cursor being placed in the "AND" field. Notice the possibility to enter addresses instead of variables names if configured appropriately in the 'FBD and LD Editor Options dialog.



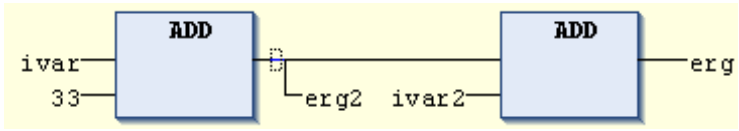
(2) Every input:



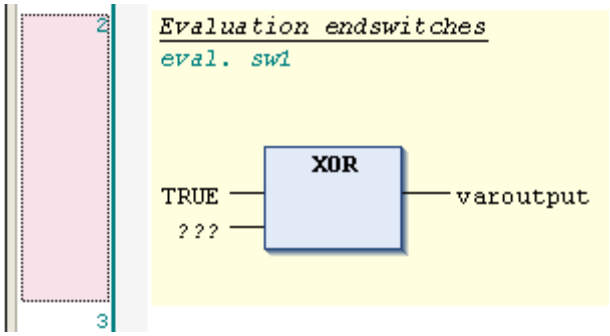
(3) Every operator, function, or function block:

(4) Outputs, if an assignment or a jump comes afterward:

(5) Just before the lined cross above an assignment, before a jump or a return instruction:

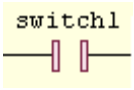


(6) The right-most cursor position in the network or anywhere else in the network besides the other cursor positions. This will select the whole network:

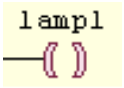


(7) The lined cross directly in front of an assignment:

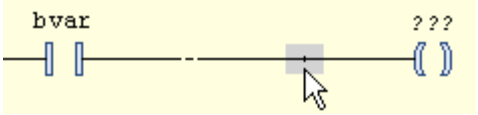
(8) Every Contact



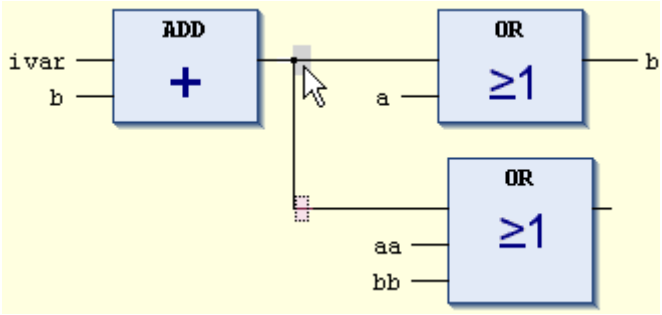
(9) Every Coil



(10) The Connecting Line between the Contacts and the Coils.



(11) Branch resp. sub-network within a network



13.8.6 FBD/LD/IL menu

When the cursor is placed in the FBD/LD/IL editor window, the FBD/LD/IL menu by default is available in the menu bar, providing the commands for programming in the currently set editor view.

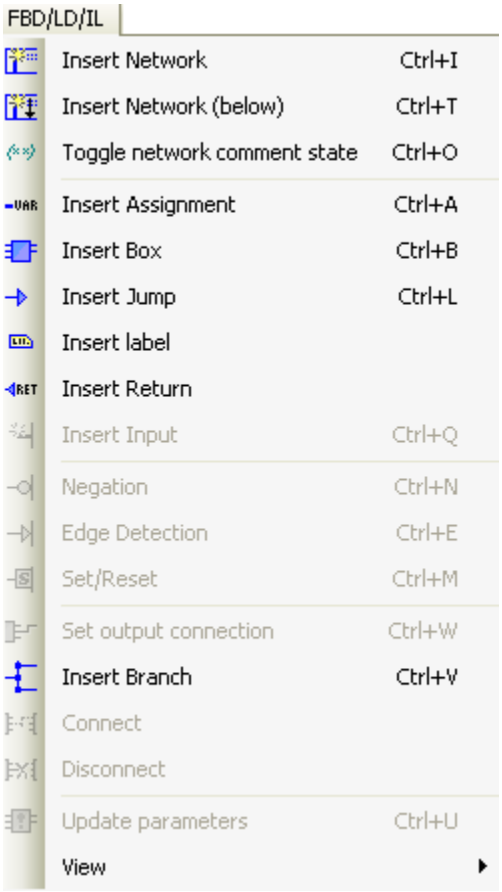


Abb. 375 Example, FBD/LD/IL menu in FBD editor view

For a description of the commands see : FBD/LD/IL Editor commands.

For configuration of the menu see: Customize Dialogs

13.8.7 FBD/LD/IL Elements

The FBD/LD/IL Editor provides a toolbox which offers the programming elements for being inserted in the editor window by drag&drop. The toolbox by default can be opened via command 'ToolBox' in the 'View' menu.

It **depends on the currently active editor view** which elements are available for inserting (see the respective description of the Insert commands). The elements are sorted in **categories**: General (general elements like Network, Assignment etc..), Boolean operators, Math operators, Other operators (e.g. SEL, MUX, LIMIT and MOVE), Function blocks (e.g. R_TRIG, F_TRIG, RS, SR, TON, TOF, CTD, CTU), Ladder elements and POUs (user-defined).

The "POUs" category lists all POUs, which have been defined by the user below the same application as the FBD/LD/IL object which is currently opened in the editor. If a POU has got assigned a bitmap in its properties, then this will be displayed before the POU name, otherwise the standard icon for indicating the POU type. The list will be updated automatically when POUs are added or removed from the application.

The category folders can be unfolded by a mouse-click on the button showing the respective category name. See in the following picture: Category "General" currently is unfolded, the others are folded. The picture shows an example for inserting an Assignment element by drag&drop from the ToolBox. Currently only section "General" in the toolbox is unfolded:

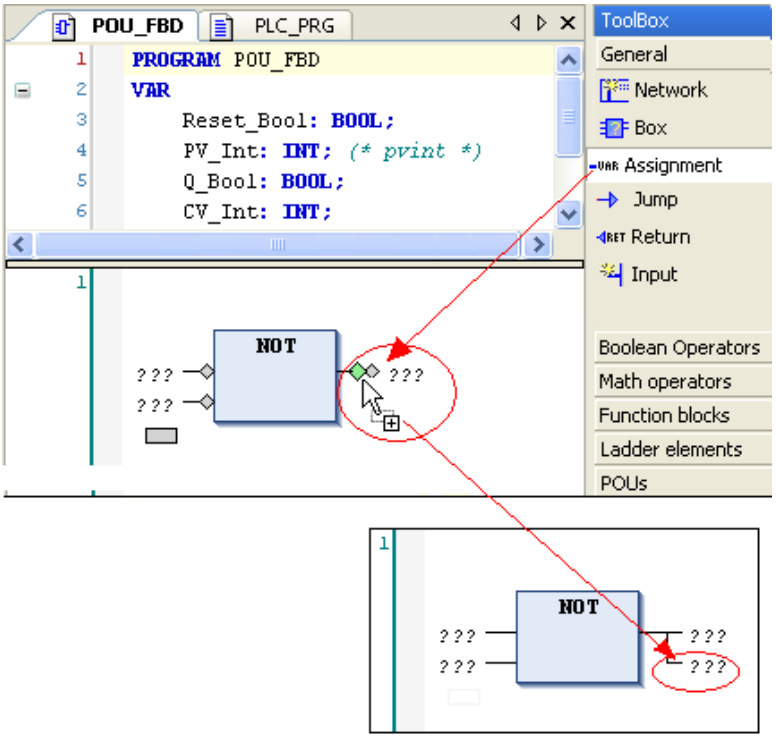


Abb. 376 Inserting from ToolBox, Example

To **insert** an element in the editor, select it in the toolbox by a mouse-click and by **drag&drop** bring it to the editor window. The possible insert positions will be indicated by **position markers** (usually rhombs), which appear as long as the element is drawn - keeping the mouse button pressed - across the editor window. Always the nearest possible position will light up **green**. When leaving the mouse button the element will be inserted at the currently "green" position.

If you draw a box element on an existing box element, the new one will replace the old one; if inputs and outputs already have been assigned, those primarily will remain as defined.

Network

A network is the basic entity of a FBD or LD program. In the FBD/LD editor the networks are arranged in a vertical list. Each network is designated on the left side by a serial network number and has a structure consisting of either a logical or an arithmetic expression, a program, function or function block call, and a jump or a return instruction.

The IL editor, due to the common editor base with the FBD and LD editors, also uses the network element. If an object initially was programmed in FBD or LD and then is converted to IL, the networks will be still present in the IL program. Viceversa, if you start programming an object in IL, you at least need 1 network element which might contain all instructions, but you also can use further networks to structure the program, for example if considering a conversion to FBD or LD.

A network optionally can get assigned a **title**, a **comment** and a **label**:

The availability of the title and the comment fields can be switched on and off in the 'FBD, LD, IL Editor Options dialog. If the option is activated, you can open an edit field for the title by a mouse-click in the network directly below the upper border. For entering a comment correspondingly open an edit field directly below the title field. The comment might be multi-lined. Linebreaks can be inserted via <Enter>, the input of the comment text is terminated by <Ctrl>+<Enter>.

To add a label, which then can be addressed by a jump, use the command Insert label. If a label is defined, it will be displayed below the title and comment field resp. - if those are not available - directly below the upper border of the network.

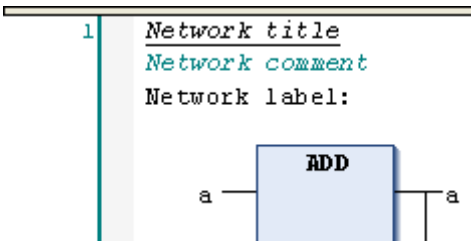


Abb. 377 Positions of title, comment and label in a network

A network can be set in "comment state", which effects that the network is not processed but displayed and handled like a comment.

On a currently selected network (cursor position 6) the default commands for copying, cutting, inserting and deleting can be applied.

Note: The right mouse click (cursor position 6) being executed over a title, comment or label will select this entry only instead of the whole network. So the execution of the default commands will have no influence on the network itself.

To insert a network, use command Insert Network or drag it from the toolbox. A network with all belonging elements can also be copied or moved by drag&drop within the editor.

Notice the possibility to create "subnetworks" by inserting "branches".

RET network:

In online mode automatically an additional, empty network will be displayed below the existing networks. Instead of a network number it is identified by "RET". It represents the position at which the execution will return to the calling POU and provides a possible halt position.

Assignment in FBD/LD/IL

Depending on the selected position in FBD or LD an assignment will be inserted directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6). In an LD network an assignment will be displayed as a coil. Alternatively drag the assignment element from the toolbox or copy or move it by drag&drop within the editor view.

After insertion the text string "???" can be replaced by the name of the variable that is to be assigned. For this via the ... button you can use the Input Assistant.

In IL an assignment is programmed via LD and ST instructions. See in this context: Modifiers and operators in IL.

Jump

Depending on the selected position in FBD or LD a jump will be inserted directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6). Alternatively drag the jump element from the toolbox or copy or move it by drag&drop within the editor.

After insertion the automatically entered "???" can be replaced by the label to which the jump should be assigned.

In IL a jump is inserted via an JMP instruction. See in this context the description of the operators and modifiers.

Label

Each FBD / LD or IL network below the network comment field has a text input field for defining a label (see command Insert label). The label is an optional identifier for the network and can be addressed when defining a jump. It can consist of any sequence of characters.

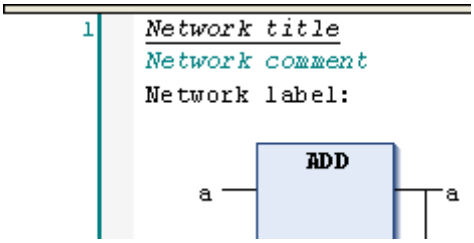


Abb. 378 Position of the label in a network

See the 'FBD, LD, IL Editor Options dialog for defining the display of comment and title.

Boxes in FBD/LD/IL

A box, insertable in a FBD, LD or IL network, is a complex element and can represent additional functions like eg. Timers, Counters, arithmetic operations or also programs, IEC functions and IEC function blocks.

A box can have any desired inputs and outputs and can be provided by a system library or be programmed by the user. At least one input and one output however must be assigned to Boolean values.

If provided with the respective module and if option Show box icon is activated, an icon will be displayed within the box.

Use in FBD, LD:

A box can be positioned in the left part of a LD network (like a contact) resp. in a FBD network by using command Insert Box, Insert Empty Box. Alternatively it can be inserted from the toolbox or copied or moved within the editor via drag&drop. Please see Insert Box the details.

Use in IL:

In an IL program a CAL instruction with parameters will be inserted in order to represent a box element.

An **update** of the box parameters (inputs, outputs) - in case the box interface has been changed - in the current implementation can be done without having to re-insert the box by the 'Update parameters' command.

RETURN instruction in FBD/LD/IL

With a RETURN instruction the FBD, LD or IL POU can be left.

In a FBD or LD network it can be placed in parallel or at the end of the previous elements. If the input of a RETURN is TRUE, the processing of the POU immediately will be aborted.

For inserting use command Insert Return. Alternatively drag the element from the toolbox or copy or move it from another position within the editor.

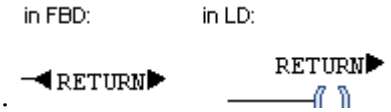


Abb. 379 RETURN element

In IL the RET instruction is used.

Branch / Hanging Coil in FBD/LD/IL

FBD, LD

In a FBD resp. LD network a branch resp. a hanging coil splits up the processing line as from the current cursor position. The processing line will continue in two "**subnetworks**" which will be executed one after each other from up to down. Each subnetwork can get a further branch, such allowing multiple branching within a network.

Each subnetwork gets an own "**marker**" (an upstanding rectangle symbol) which can be selected (cursor position 11) in order to perform actions on this arm of the branch.

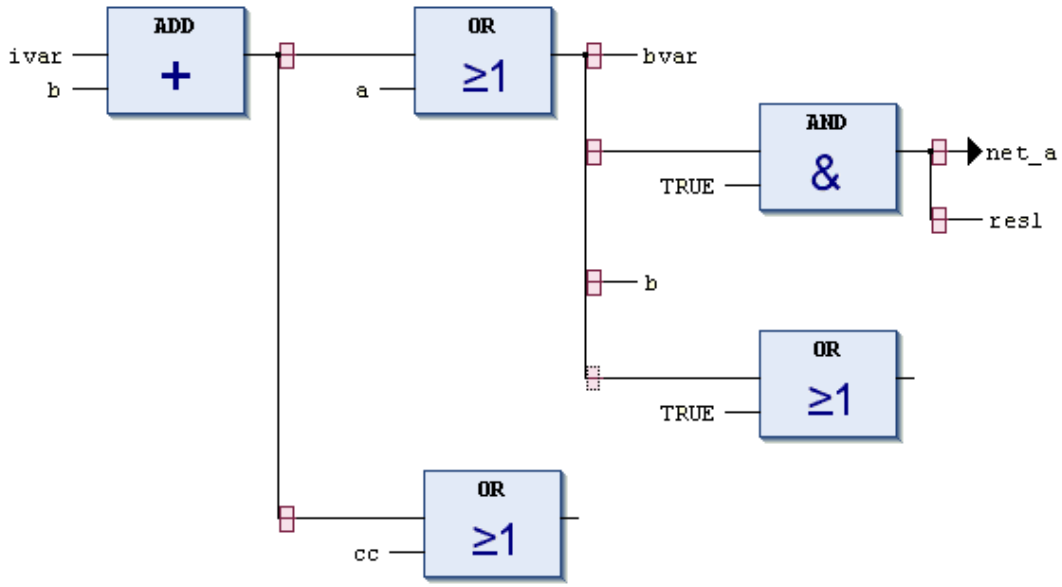


Abb. 380 Examples of branch markers

In FBD a branch gets inserted via command "Insert Branch". Alternatively drag the element from the toolbox. For the possible insert positions please see 'Insert branch".

Note: Cut&Paste is currently not possible for subnetworks.

See the example shown in the image below: A branch has been inserted at the SUB box output. This created two subnetworks, each selectable by its subnet marker. Atter that an ADD box was added in each subnetwork.

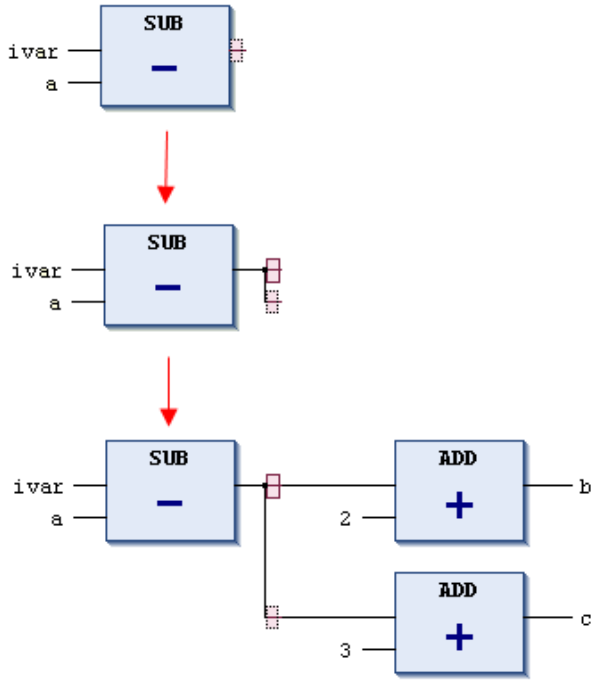


Abb. 381 Example in FBD, Inserting a branch

To delete a subnetwork, first remove all elements of the subnetwork, that is all elements which are positioned to the right of the subnetwork's marker, then select the marker and use the standard Delete command resp. <Del>. See in the following picture: The 3-input-OR element must be deleted before you can select and delete the marker of the lower subnetwork.

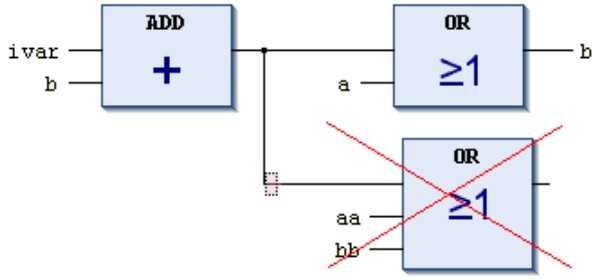


Abb. 382 Delete branch resp. subnetwork

Execution in online mode:

The particular branches will be executed from left to right and then from up to down.

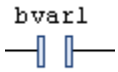
IL (Instruction List)

In IL a "branch" resp. "hanging coil" is represented by an appropriate order of instructions. See in this context: Modifiers and operators in IL.

Contact

This is a **LD element.**

Each network in LD in its left part contains one or several contacts. Contacts are represented by two parallel lines:



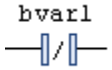
Contacts pass on condition "ON" (TRUE) or "OFF" (FALSE) from left to right.

A **boolean variable** is assigned to each contact. If this variable is TRUE, the condition is passed from left to right and finally to a coil in the right part of the network, otherwise the right connection receives the value FALSE.

Multiple contacts can be connected **in series** as well as **in parallel**. In case of two parallel contacts only one of them must transmit the value TRUE in order to get the parallel branch transmit the value TRUE. In case of contacts connected **in series** all contacts must transmit the condition TRUE in order to get the last contact transmit the TRUE condition.

So the contact arrangement corresponds to either an electric parallel or a series circuit.

A contact can also be **negated**, recognizable by the slash in the contact symbol:



A negated contact passes on the incoming condition (TRUE or FALSE) only if the assigned boolean variable is FALSE. Notice that the ToolBox directly provides negated contact elements.

A contact can be **inserted** in a LD network via one of the commands Insert Contact or Insert Contact (right) resp. Insert Contact Parallel (above) or Insert Contact Parallel (below) which by default are part of the FBD/LD/IL menu. Alternatively the element can be inserted via drag&drop from the ToolBox or from another position within the editor.

FBD, IL

If you are currently working in FBD or IL view, the command will not be available, but contacts and coils inserted in a LD network will be represented by corresponding FBD elements resp. IL instructions.

Coil

This is a **LD element.**

On the right side of a LD network there can be any number of so-called coils which are represented by parentheses:

They can only be arranged **in parallel**. A coil transmits the value of the connections from left to right and copies it to an appropriate **Boolean variable**. At the entry line the value ON (TRUE) or the value OFF (FALSE) can be present.

Contacts and coils can also be **negated**, recognizable by the slash in the coil symbol:

In this case the negated value of the incoming signal will be copied to the appropriate Boolean variable, thus: a negated contact only will connect through if the appropriate Boolean value is FALSE.

A coil can be **inserted** in a network via command Insert Assignment which per default is part of the FBD/LD menu Alternatively the element can be inserted via drag&drop from the ToolBox ("Ladder elements") or from another position within the editor. See also: Set and Reset Coils.

FBD, IL

If you are currently working in FBD or IL view, the command will not be available, but contacts and coils inserted in a LD network will be represented by corresponding FBD elements resp. IL instructions.

Set/Reset in FBD/LD/IL

FBD and LD

A boolean output in FBD or correspondingly a LD coil can be "set" or "reset". To change between the set states use the respective command Set/Reset from the context menu when the output is selected. The output resp. coil will be marked by a S or a R.

Set: If value TRUE arrives at a set output resp. coil, this output/coil will get TRUE and keep TRUE. This value cannot be overwritten at this position as long as the application is running.

Reset: If value TRUE arrives at a reset output resp. coil, this output/coil will get FALSE and keep FALSE. This value cannot be overwritten at this position as long as the application is running.

Abb. 383 Set output in FBD, example

See also: Set and Reset Coils.

IL

In an instruction list the S and R operators are used to set or reset an operand.

Set/Reset Coil

Coils can also be defined as set or reset coils. One can recognize a set coil by the "S" in the coil symbol: **(S).** A set coil will never overwrite the value TRUE in the appropriate Boolean variable. That is, the variable once set to TRUE remains TRUE.

One can recognize a reset coil by the "R" in the coil symbol: **(R).** A reset coil will never overwrite the value FALSE in the appropriate Boolean variable: the variable once set to FALSE will remain FALSE.

In the LD editor Set coils and Reset coils can directly be inserted via drag&drop from the ToolBox, category 'Ladder elements': , .

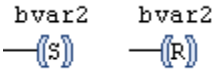


Abb. 384 Set coil, Reset coil, examples

13.8.8 FBD/LD/IL Editor in Online Mode

In online mode the FBD/LD/IL Editor provides views for monitoring and for writing and forcing the variables and expressions on the controller. Debugging functionality (breakpoints, stepping etc.) is available. See below.

- For information on how to open objects in online mode see User Interface in Online Mode.
- Notice that the editor window of an FBD, LD or IL object also includes the Declaration Editor in the upper part. See also in this context: Declaration Editor in online mode.

Monitoring

If the inline monitoring is not explicitly deactivated in the Options dialog, it will be supplemented in FBD or LD editor by small monitoring windows behind each variable resp. by an additional monitoring column showing the actual values (**Inline Monitoring**). This is even the case for not assigned function block inputs and outputs.

The inline monitoring window of a variable shows a little red triangle in the upper left corner, if the variable currently is forced, a blue one in the lower left corner, if the variable currently is prepared for writing or forcing. Example for a variable which is currently forced and prepared for releasing the force: MyCounter 40

SP_for_Win32.App.CallTest

Expression	Comment	Type	Value	Prepared value
Inst2		UpAndDown		
Enable		BOOL	TRUE	
Amplitude		INT	200	
GoHome		BOOL	FALSE	
Value		INT	41	
Up		BOOL	FALSE	
Down		BOOL	TRUE	
Counter		DINT	2159	
diValue		DINT	41	
X		BOOL	FALSE	
X		INT	0	
X2		INT	0	
Up		BOOL	TRUE	
Down		BOOL	FALSE	
MyCounter	Vars for third network	DINT	F 40	<Unforce and restore>
MyDownCounter		DINT	-2	
Err		BOOL	FALSE	
ErrCode		WORD	0	
stRotate	Vars for fifth network	STRING	'pcom'	'abc'
Ave	Vars for fifth network	DINT	38	

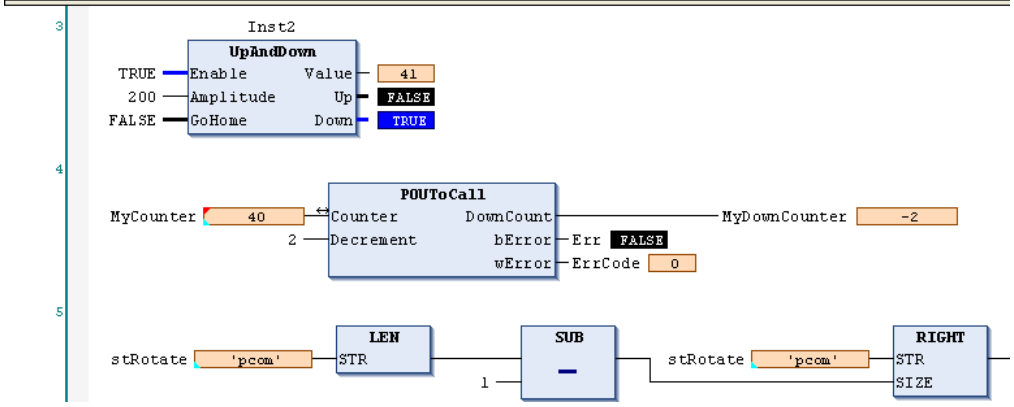


Abb. 385 Example - Online view of a FBD program

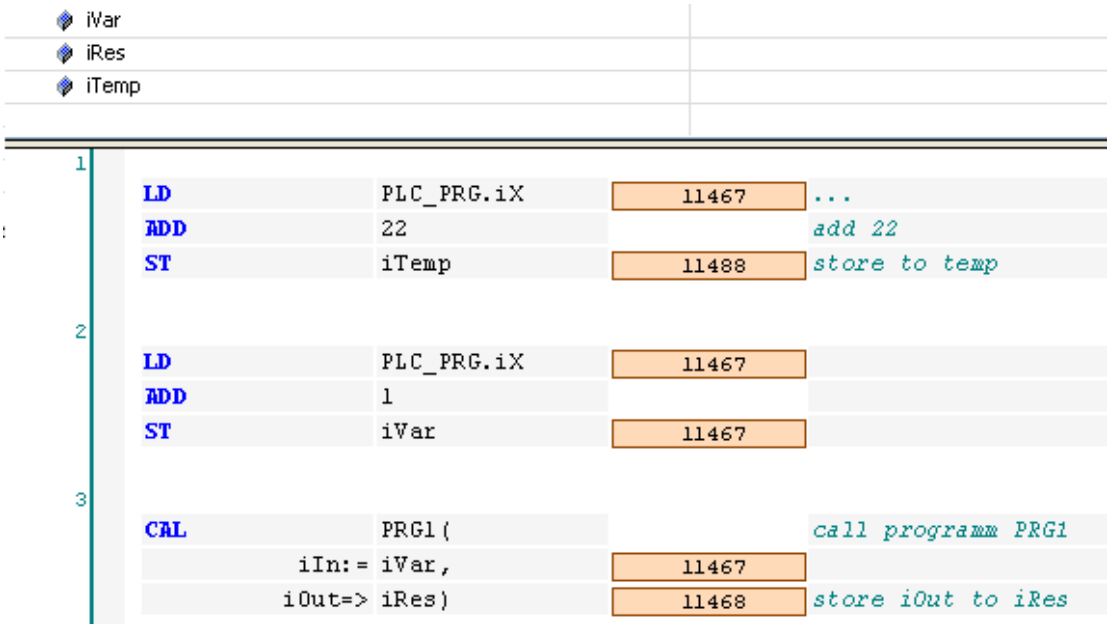


Abb. 386 Example - Online view of a IL program

In online view ladder networks have animated connections: Connections with value TRUE are displayed in bold blue, connections with value FALSE in bold black, whereas connections with no known value or with an analog value are displayed in standard outline (black and not fat). (Remark: the values of the connections are calculated from the monitoring values. It is no real powerflow.)

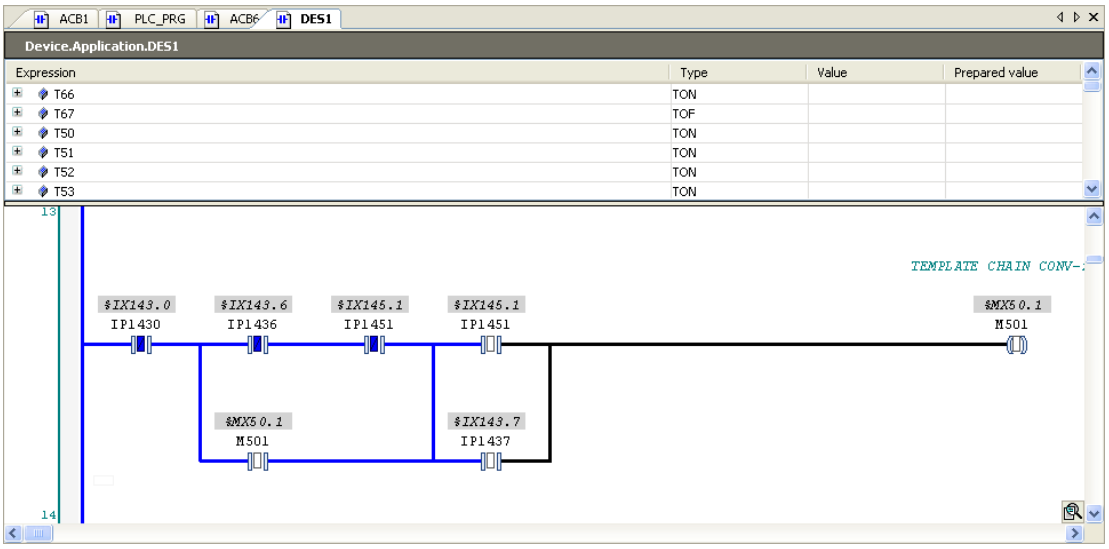


Abb. 387 Example - Online view of a LD program

Notice for the online view of a POU of type function block: In the implementation part no values will be viewed in the monitoring windows but "<Value of the expression>" and the inline monitoring fields in the implementation part will show three question marks each.

Forcing/Writing of variables

In online mode you can prepare a value for forcing or writing a variable either in the declaration editor or within the implementation part. In the implementation part at a mouseclick on the variable the following dialog will open:

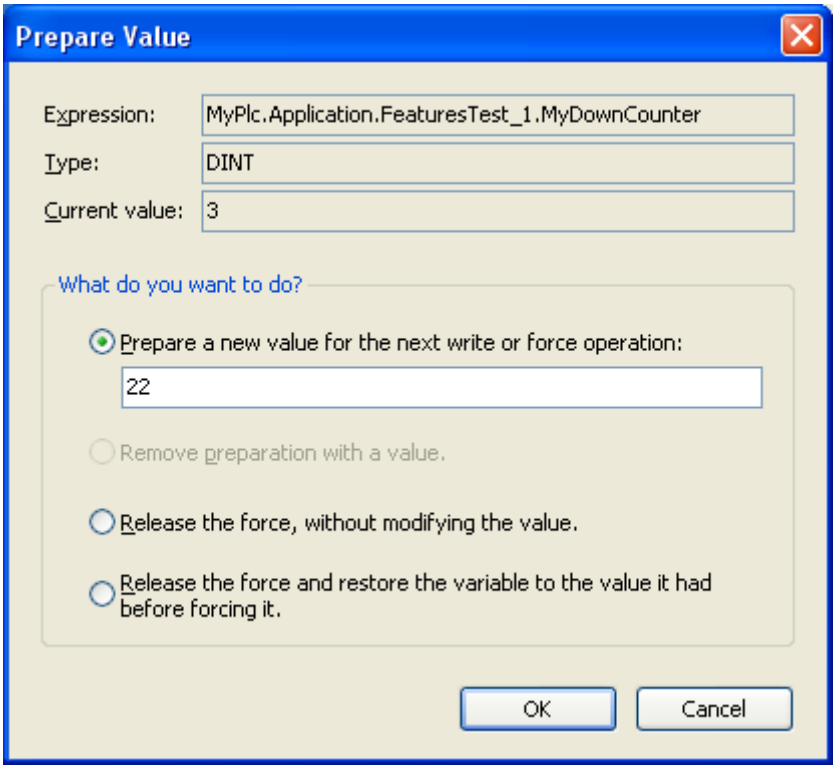


Abb. 388 Dialog - Prepare Value

You find the name of the variable completed by its path within the device tree ("Expression"), its type and current value. By activating the corresponding item you may choose wether you want to

- prepare a new value which must be entered in the edit field thereby
- remove a prepared value
- release the variable actually forced
- release the variable actually forced and reset it to the value it was assigned to just before forcing

The selected action will be carried out on executing the menu command **Force Values** (per default in the Online menu) or pressing **F7**.

Breakpoint resp. Halt positions:

Possible positions which can be defined for a breakpoint (halt position) for debugging purposes, basically are those positions at which values of variables can change (statements), at which the program flow branches out, or at which another POU is called. That is the following positions:

- on the network at a whole which effects that the breakpoint will be applied to the first possible position within the network
- on a box if this contains a statement; so not possible on operator boxes like for example ADD, DIV; regard however the Note inserted below
- on an assignment.
- at the end of a POU at the point of return to the caller; in online mode automatically an empty network will be displayed for this purpose, which instead of a network number is identified by "RET".

Note: Currently you cannot set a breakpoint directly on the first box of a network. If however a breakpoint is set on the complete network, the halt position will automatically be applied to the first box.

You might have a look at the selection list within the breakpoint dialog for all currently possible positions.

A network containing any active breakpoint position is marked with the breakpoint symbol (red filled circle) right to the network number and a red-shaded rectangle background for the first possible breakpoint position within the network. Deactivated breakpoint positions are indicated by a non-filled red circle resp. a surrounding non-filled red rectangle.

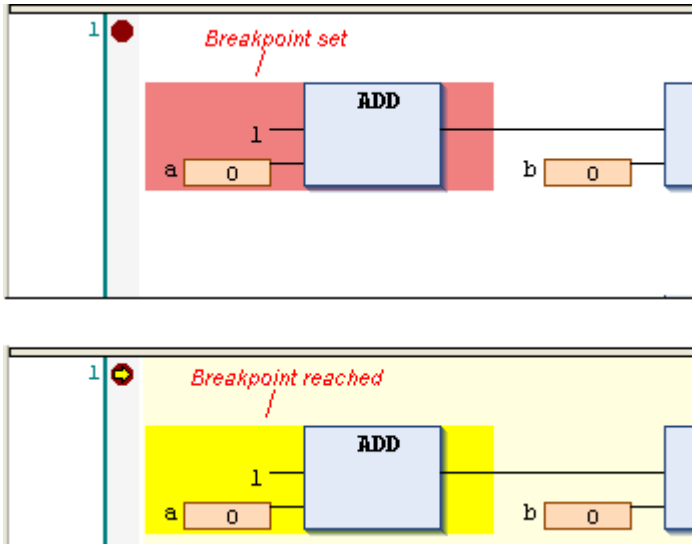


Abb. 389 Example - Breakpoint set resp. reached

As soon as a breakpoint position is reached during stepping or program processing, a yellow arrow will be displayed in the breakpoint symbol and the red shaded area will change to yellow.

The currently reached halt position is indicated by a yellow shadow and the subsequent, not yet reached one by a red shadow:

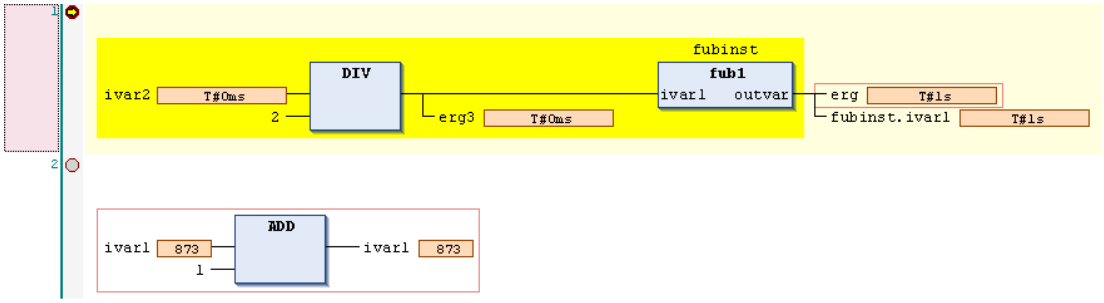


Abb. 390 Halt positions shown in an FBD example

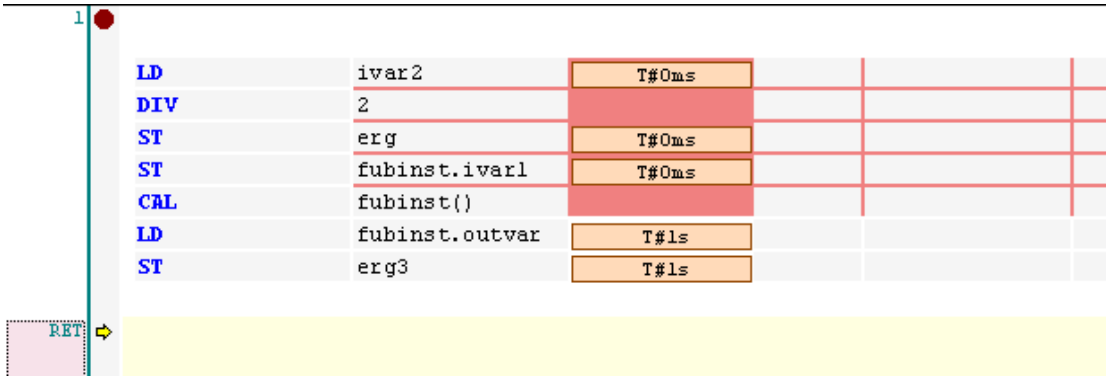


Abb. 391 Halt positions shown in an IL example

Note: Notice for breakpoints in methods: A breakpoint will be set automatically in all methods which might be called. So, if an interface-managed method is called, breakpoints will be set in all methods of function blocks implementing that interface and also in all derivative function blocks subscribing the method. If a method is called via a pointer on a function block, breakpoints will be set in the method of the function block and in all derivative function blocks which are subscribing the method.

13.9 Global Variables List Editor

The GVL Editor is a declaration editor for editing Global Variables Lists. It is working according to the options currently set for a text editor and in online mode also appearing like described for the declaration editor. The declaration must start with "VAR_GLOBAL" and end with "END_VAR". These keywords are provided automatically. In between enter valid declarations of global variables.

Abb. 392 GVL Editor

13.10 Library Manager Editor

The Library Manager is used for including and handling libraries in a project. The installation of libraries as well as the definition of library folders (repositories) is done by the Library Repository dialog which is also a component of the Library Manager and can be opened by the respective command in the menu bar (by default in the 'Tools' menu) or in the editor window.

The Library Manager editor is provided by the 'Library Manager Editor' plug-in and can be added to the project via command Add Object. It will be inserted as an object in the POUs view window or in the Devices view window assigned to a device or an application. At each of these possible positions only one Library Manager object can be inserted. The entry will be named as defined in the 'Add object' dialog.

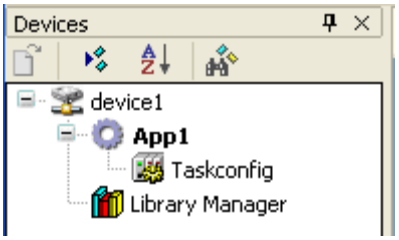


Abb. 393 Library Manager object in the POUs view window

The Library Manager can be opened in an editor window via the Edit object command or by a double-click on the object.

Compile messages concerning the Library Manager are displayed in a separate list in the Messages window.

13.10.1 Library Manager Editor window

The Library Manager editor can be opened in an editor view window via the Edit object command or by a double-click on the object.

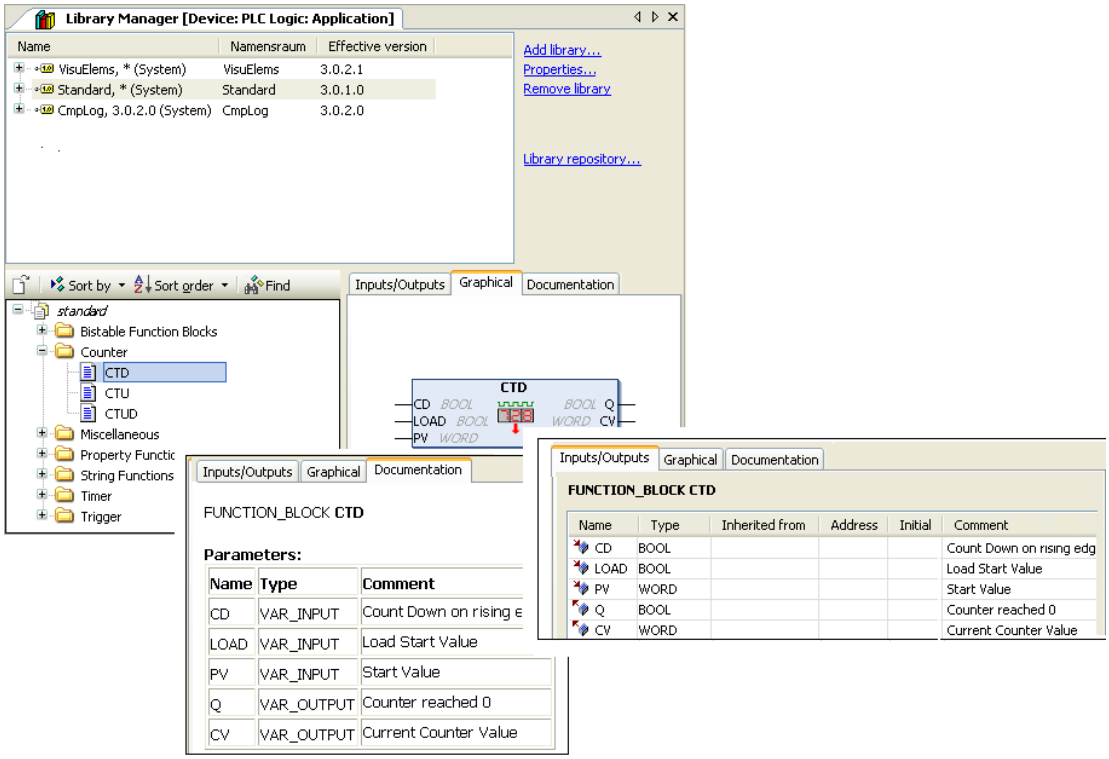


Abb. 394 Library Manager editor

See in the following on this page:

- - Structure of the editor window
- - The particular items of the editor window

For general information on the Library Management in »PLC Designer« please see: Library Management.

Structure of the editor window

The upper part of the dialog displays the libraries **currently included** in the project. The following information is provided:

Name: Title, Version and optionally Company name as it has been specified in the Summary dialog of the Project Information of the library during its creation.

Namespace: The default setting for the namespace of a library is "<library name>", **except** the library explicitly has got another namespace in its Project Information. The library namespace must be used as a prefix of the identifier in order to uniquely access a module which is plurally available in the project. The standard namespace of an included library can be modified for local use (within the project) in the 'Properties' dialog, see the corresponding command below. Please see also the help page on Library Management for information on library namespaces.

Effective version. "Effective version" is that library version which will be currently used due to the definition in the library properties.

Libraries which have been included in a project automatically by a plug-in, are displayed grey-colored, those which have been added manually (Add library...) are displayed in black color.

An icon before the library name indicates the type of the library:

	»PLC Designer« V3.x library (contains version information).
	The referenced file could not be found or is not a valid library (see the corresponding message in list 'Library Manager' in the Messages view). See command "Try to Reload Library'.

If a library has **dependencies** on other libraries (referenced libraries), those automatically will also get included - if available - and will be displayed with icon in a subtree of the entry. A subtree can be opened or closed by the respective plus- or minus-sign. As an example see library "VisuInputs" in figure 'Library Manager dialog' (above).

In the lower left part of the editor the **modules** of the currently selected library are displayed. The usual buttons for sorting, finding etc. are available for handling the modules tree.

In the lower right part the following tabs are available:

Documentation: The components of the module currently selected in the left part get displayed in a table, showing the variable Name, Data type and the Comment which might have been added to the component declaration when creating the library. Thus inserting such comments is an easy way to automatically provide the user with module documentation.

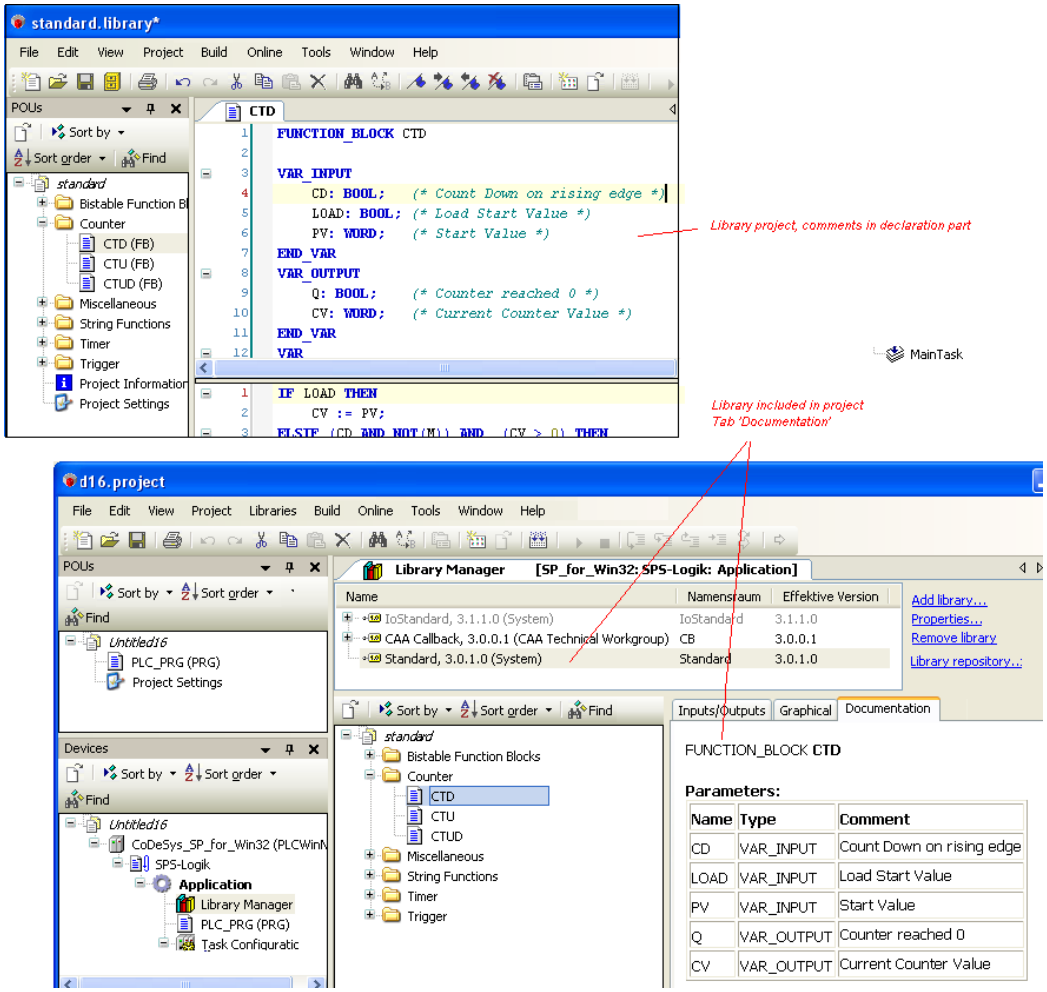


Abb. 395 Example, commented library modules

Inputs/Outputs tab on the right side the components of the currently selected library module are listed in a table with variable Name, (data) Type, Address, Initial value and Comment, as defined in the library.

Graphical tab the IEC implementation of the module is viewed graphically.

The buttons and commands in the editor window

The following commands, which are provided in the window when one or several libraries are selected, correspond to those of the 'Libraries' menu, which by default is available in the menu bar as long as the Library Manager editor is active:

Add library... for including a library into the current project; precondition: the library must be installed on the system

Properties. for settings on the namespace and - with notice that the library will later be available as a "referenced library" in another project - for settings on version handling, visibility and accessing.

Remove library: The library currently selected in the libraries list will be removed from the project.

Library Repository.... for defining library locations and for installing or uninstalling libraries.

The user will be notified if trying to insert a library which is already available in the project.

13.10.2 Libraries menu

When the Library Manager is active, the menu bar by default provides the 'Libraries' menu, containing the commands for the Library Manager dialog.

Notice also the Library Repository... command, which is by default in the Tools menu.

13.11 Network Variables List Editor

The NVL Editor is a declaration editor for editing Network Variables Lists. It is working according to the options currently set for a text editor and in online mode also appearing like described for the declaration editor. The declaration must start with "VAR_GLOBAL" and end with "END_VAR". These keywords are provided automatically. In between enter valid declarations of global variables.

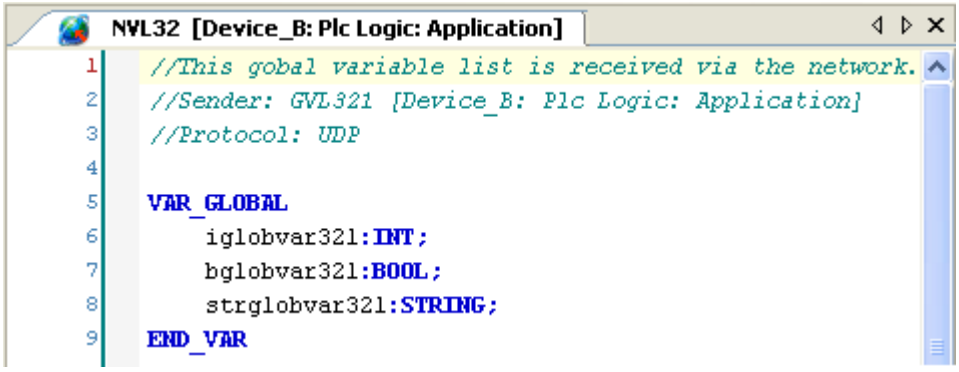


Abb. 396 NVL Editor

13.12 Recipe Manager Editor

Note: This functionality is only available if supported by the currently used feature set.

The 'Recipe Manager' plug-in provides the functionality for handling user-defined lists of project variables, named "**Recipe Definitions**" and definite value sets for these variables within a recipe definition, named "**Recipes**". Recipes can be used for setting and watching control parameters on the PLC. For this purpose they can be read from and written to the PLC and be loaded to and saved from files. These interactions are possible via visualization elements which must have been configured appropriately.

If the recipe manager is located on another PLC than the application being concerned by the recipes, the data server will be used to read/write the recipes. Reading and writing is done synchronously, that is, all variables contained in a recipe are updated at the same time. By calling g_RecipeManager.LastError after reading/writing you may check if the transmission has been carried out successfully (g_RecipeManager.LastError=0 in this case).

The Recipe Manager is a resource object to be added via command 'Add Object' in the Devices view below an application.

One or several **Recipe Definition** objects can be assigned to a Recipe Manager object, also by command 'Add Object'. Recipe definitions including the particular recipes then can be edited in a separate editor window. See the Recipe Definition help page for further information, also on how to handle the recipes in online mode.

The Recipe Manager dialog provides the following tabs:

Storage: Per default the recipes will be stored automatically to files according to the following settings:

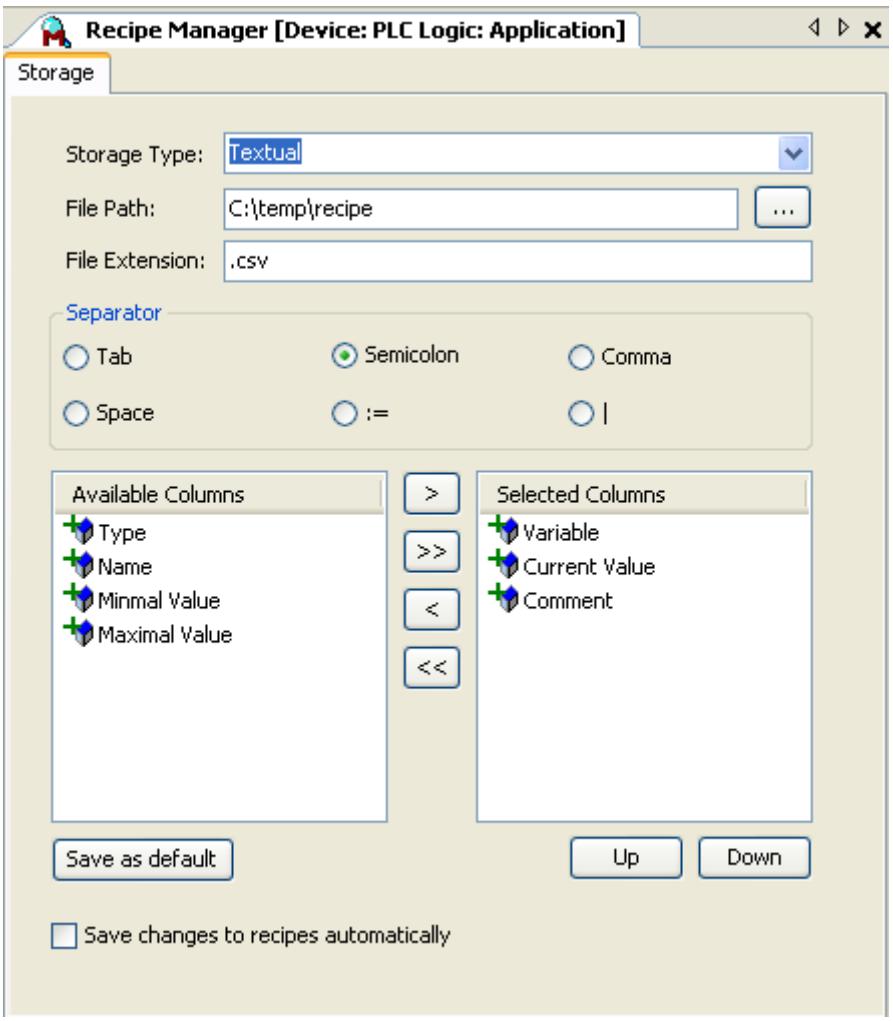


Abb. 397 Recipe Manager - Storage

Select the **Storage Type** (Textual or Binary), specify the storage location by indicating the **File Path** and define the **File Extension** of the recipe to be stored.

In case of textual storage the columns selected for storage will get separated by a **Separator**: select one of the six options proposed.

The bottom left part of the dialog shows all columns of the recipe definition represented by the respective header (**Available Columns**). The right part contains all **Selected Columns** of the recipe definition, that is, the columns that shall get stored. At least the column containing the "Current Value" will be contained in the right part and may not be deselected.

The other columns may be shifted to the right or to the left by selecting the respective entry and clicking on [>] or [<]. You may also shift all entries from one side to the other at once by using [>>] or [<<].

The buttons Up and Down may be used to adjust the order of the selected columns, which represents the order the columns will get stored within the storage file.

For each recipe a file **<recipe name>.<recipe definition>.<file extension>** will be created in the specified folder. This file(s) will be reload to the Recipe Manager at each restart of the application. As long as option **Save changes to recipes automatically** is activated (default), the storage file(s) will be updated at any modification of a recipe during run time. If this is not desired, the option must be deactivated.

If you click on **Save as default** the settings made within this dialog will from now on be used as default settings for each further recipe manager inserted.

13.12.1 Recipe Definition

The Recipe Manager handles one or several recipe definitions. A "**recipe definition**" contains a list of variables and one or several "**recipes**" (value sets) for these variables. By using different recipes you can for example assign another set of values to a set of variables on the PLC in one stroke.

Recipe Definition

A Recipe Definition must be added as a further object below the Recipe Manager in the Devices tree.

When a recipe definition object is selected, the corresponding editor window can be opened by a mouse-click on the entry or by command 'Edit Object'.

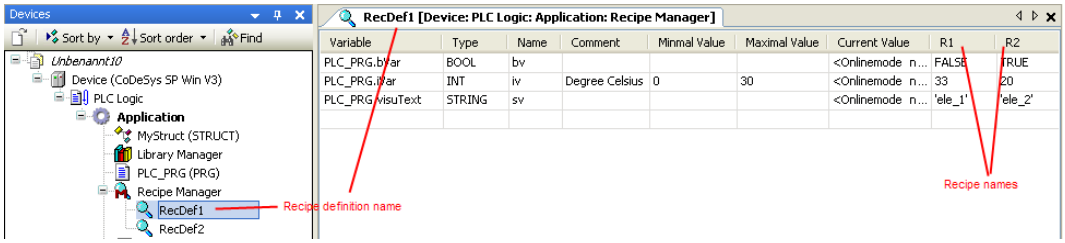


Abb. 398 Recipe Definition, editor window

The editor window will be titled with the name of the recipe definition.

In a table you can enter several project variables for which you want to define one or several recipes. Always one empty line will be available at the end of the table for adding a further variable.

In the **Variable** field enter the name of a project variable, for example "plc_prg.ivar". The **Type** will be filled automatically. Optionally you can define a symbolic **Name**.

The column **Comment** is thought for adding additional information as for example the unit of the value recorded in the variable.

The **Minimal Value** and **Maximal Value** which should be permissible for being written on this variable can be specified optionally.

The **Current Value** will be monitored in online mode.

An variable (line) can be removed from the table by Del (key resp. command) when one of its cells is selected. Multiple selection of lines is possible by keepint the Strg key pressed while selecting cells. The selected line(s) can be copied by Copy&Paste. Paste inserts the copied line(s) above the currently selected line. In doing so, recipe values will be inserted in the matching recipe column, if available.

In online mode the existing recipe definition can be changed either by a appropriate configured visualization element or by using the appropriate methods of the function block "RecipeManCommands" of the Recipe_Management.library. Therefore the following methods can be taken:

- **ReadRecipe**: The current variable values are taken into the recipe.
- **WriteRecipe**: The recipe is written into the variables.
- **SaveRecipe**: The recipe is stored into a standard recipe file.
- **LoadRecipe**: The recipe is loaded from a standard recipe file.
- **CreateRecipe**: A new recipe in the recipe definition is created.
- **DeleteRecipe**: An existing recipe from a recipe definition is deleted.

If a new download is executed all recipes of the project will be saved again. All other recipes (dynamic generated recipes) remain unchanged.

For each recipe within the recipe definition there is an own column, titled with the **recipe name**:

Recipes

A recipe can be **added** or **removed** offline or online. In offline mode use the commands 'Add a new Recipe' resp. 'Remove recipes' within the Recipe Manager editor. In online mode either configure an input on an appropriately configured visualization element, or use the appropriate methods of function block "RecipeManCommands" of the Recipe_Management.library.

When adding a recipe, a further column will be added behind the right-most column, titled with the name of the recipe (see picture above). The fields of a recipe column can be filled with appropriate values. Thus for the same set of variables different sets of values can be prepared in the particular recipes.

Using recipes in online mode:

The recipes can be handled (create, read, written, saved, loaded, delete) by using the methods of the function block **RecipeManCommands**, provided by the library **Recipe_Management.libray**, in the application code, or via inputs on **visualization** elements.

Note: Concerning the storage of recipes in files, which are reloaded at a restart of the application, see the Recipe Manager dialog.

- For a description of the RecipeManCommands methods please see the documentation within the library.
- For the appropriate "input" configuration of a visualization element see the help page on the visualization element properties (category "Input", "execute command").

The following actions on recipes are possible:

Create recipe (=Add a new Recipe)	A new recipe will be created in the specified recipe definition.
Read recipe	The current values of the variables of the specified recipe definition will be read from the controller and be "written" to the specified recipe. This means that the values will be stored implicitly (in a file on the PLC) and also immediately will be monitored in the recipe definition table in the »PLC Designer« Recipe Manager. In other words: The recipe managed in the Recipe Manager gets updated with the actual values from the PLC.
Write recipe	The values of the given recipe, as visible in the Recipe Manager, will be written to the variables on the PLC.
Save recipe	The values of the specified recipe will be written to a file with extension ***.txtrecipe**, the name of which you have to define. For this purpose the standard dialog for saving a file in the local file system will open. ATTENTION: The implicitly used recipe files, necessary as a buffer for reading and writing of the recipe values, may not overwritten. This means that the name for the new recipe file must be different from "<recipe name>.<recipe definition name>.txtrecipe" ! The values of the given recipe will be written to a file, which automatically gets named like the recipe.
Load recipe	The recipe which has been stored in a file (see above "Save recipe") can be reloaded from this file. The standard dialog for browsing for a file will open for this purpose. The filter automatically is set to extension "*.txtrecipe". After reloading the file, the recipe values will accordingly be updated in the »PLC Designer« Recipe Manager.
Delete recipe (=Remove recipes)	The specified recipe will be removed from the recipe definition.
Change recipe	The value of the project variables can be changed. So with a following write recipe action the appropriate project variables are written with the new values.

13.12.2 RecipeManCommands

CreateRecipe: This method creates a new recipe in the specified recipe definition and afterwards reads the current plc values into the new recipe. At the end the new recipe would be stored in the default file.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ALREADY_EXIST, ERR_FAILED, ERR_OK

CreateRecipeNoSave: This method creates a new recipe in the specified recipe definition and afterwards reads the current plc values into the new recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ALREADY_EXIST, ERR_FAILED, ERR_OK

DeleteRecipe: This method deletes a recipe from a recipe definition.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

DeleteRecipeFile: This method deletes the standard recipe file from a recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_OK

LoadAndWriteRecipe: This method loads a recipe from the standard recipe file and afterwards writes the recipe into the plc variables.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadFromAndWriteRecipe: This method loads a recipe from the specified recipe file and afterwards writes the recipe into the plc variables.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

FileName - Name of the file.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadRecipe: This method loads a recipe from the standard recipe file.

The standard recipe file name is <recipe>.<recipedefinition>.<recipeextension>.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

ReadAndSaveRecipe: This method reads the current plc values into the recipe and afterwards stores the recipe into the standard recipe file.

The content of an existing file would be overridden.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadAndSaveRecipeAs: This method reads the current plc values into the recipe and afterwards stores the recipe into the specified recipe file.

The content of an existing file would be overridden.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

FileName - Name of the file.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

SaveRecipe: This method stores the recipe into the standard recipe file. The content of an existing file would be overridden.

The standard recipe file name is <recipe>.<recipedefinition>.<recipeextension>.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadRecipe: This method reads the current plc values into the recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

WriteRecipe: This method writes the recipe into the plc variables.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

ReloadRecipes: This method reloads the list of recipes from the file system.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK

GetRecipeCount: This method returns the recipe count of the corresponding recipe definition.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

Return values:

-1 : if the recipe definition is not found

GetRecipeNames: This method returns the recipe names from the corresponding recipe definition.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

pStrings - The strings were the recipe values should be stored.

iSize - The size of the string array.

iStartIndex - The start index. Can be used for a scrolling function.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK

Example:

We have 50 recipes. We want to create a table which shows 10 recipe names at a time.

We define a Stringarray strArr: ARRAY[0..9] OF STRING;

Corresponding to the iStartIndex we can get the recipe names from a specific area.

iStartIndex := 0; The names 0..9 are returned.

iStartIndex := 20; The names 20..29 are returned.

In this example:

iSize := 10;

GetRecipeValues: This method returns the recipe values from the corresponding recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

pStrings - The strings were the recipe values should be stored.

iSize - The size of the string array.

iStartIndex - The start index. Can be used for a scrolling function.

iStringLength - The string length of the strings in the array.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

Example:

We have 50 recipe values. We want to create a table which shows 10 recipe values at a time.

We define a Stringarray strArr: ARRAY[0..9] OF STRING;

Corresponding to the iStartIndex we can get the recipe values from a specific area.

iStartIndex := 0; The values 0..9 are returned.

iStartIndex := 20; The values 20..29 are returned.

In this example:

iStringLength := 80;

iSize := 10;

GetRecipeValues: This method returns the recipe variable names from the corresponding recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

pStrings - The strings were the recipe values should be stored.

iSize - The size of the string array.

iStartIndex - The start index. Can be used for a scrolling function.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

Example:

We have 50 recipe values. We want to create a table which shows 10 recipe variable names at a time.

We define a Stringarray strArr: ARRAY[0..9] OF STRING;

Corresponding to the iStartIndex we can get the recipe variable names from a specific area.

iStartIndex := 0; The names 0..9 are returned.

iStartIndex := 20; The names 20..29 are returned.

In this example:

iSize := 10;

SetRecipeValues: This method sets the recipe values into the corresponding recipe.

Parameters:

RecipeDefinitionName - Name of the recipe definition.

RecipeName - Name of the recipe.

pStrings - The strings where the recipe values should be stored.

iSize - The size of the string array.

iStartIndex - The start index. Can be used for a scrolling function.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

Example:

We have 50 recipe values. We want to create a table which shows 10 recipe values at a time.

We define a Stringarray strArr: ARRAY[0..9] OF STRING;

Corresponding to the iStartIndex we can get the recipe values from a specific area.

iStartIndex := 0; The values 0..9 are returned.

iStartIndex := 20; The values 20..29 are returned.

In this example:

iStringLength := 80;

iSize := 10;

GetLastError: This method returns the last error of the previous operations.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_OK

ResetLastError: This method resets the last error.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_OK

SetRecipeManager: This method sets the recipe manager the function block works with.

Return values:

ERR_NO_RECIPE_MANAGER_SET, ERR_OK.

13.12.3 Recipe Manager in online mode

not yet implemented

13.13 SFC Editor

The SFC Editor, provided by the 'SFC Editor' plug-in, is available for programming objects in the IEC 61131-3 programming language Sequential Function Chart (SFC). The language is to be chosen when adding a new POU object to the project via the 'Add Object' command.

The SFC Editor is a **graphical** editor. General settings concerning behaviour and display are done in the SFC editor **options** dialog.

The SFC editor is available in the lower part of the window which opens when you edit a SFC POU object. In its upper part this window contains the Declaration Editor.

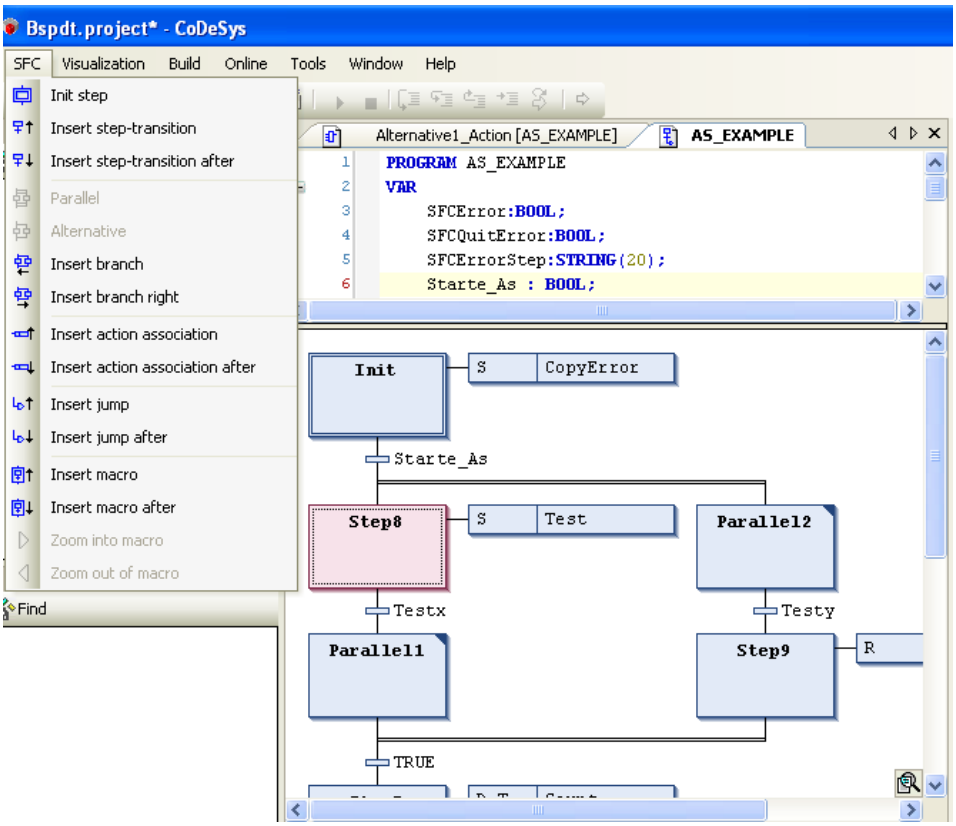


Abb. 399 SFC Editor

The **elements** used in a SFC diagram by default are available in the "SFC" menu, which by default is available as soon as the SFC Editor is active. They are to be arranged in a sequence resp. in parallel sequences of steps which are connected by transitions. See also: Working in the SFC Editor.

The **properties** of steps can be edited in a separate Properties box. Inter alia there the minimum and maximum time of activity can be defined for each step.

Implicit variables can be accessed for controlling the processing of a SFC (for example step status, timeout analyzation, reset etc.).

The **commands** for working in the SFC Editor can be called from the context menu or from the "SFC" menu which by default is available as soon as the SFC Editor is active.

Comparison to the »PLC Designer« V2.3 SFC Editor:

- Basically the same functionality is available, »PLC Designer« V2.3 SFC projects can be imported.
- Editing is made more comfortable by the fact that each particular element can be selected and arranged individually. During editing the syntax of the SFC not necessarily must be matched. Syntax errors will not be checked until a precompile run (build).
- There is only one step type, combining the two types "IEC steps" and "3S-steps", which are used in V2.3. Actions always must be provided as POUs. The actions always are assigned via the step element properties.
- Macros can be used for structuring purposes.

13.13.1 SFC - Sequential Function Chart

The Sequential Function Chart (SFC) is a graphically oriented language which allows to describe the chronological order of particular actions within a program. These **actions** are available as separate programming objects, written in any available programming language. In a SFC they get assigned to "**step**" elements and the sequence of processing is controlled by **transition** elements.

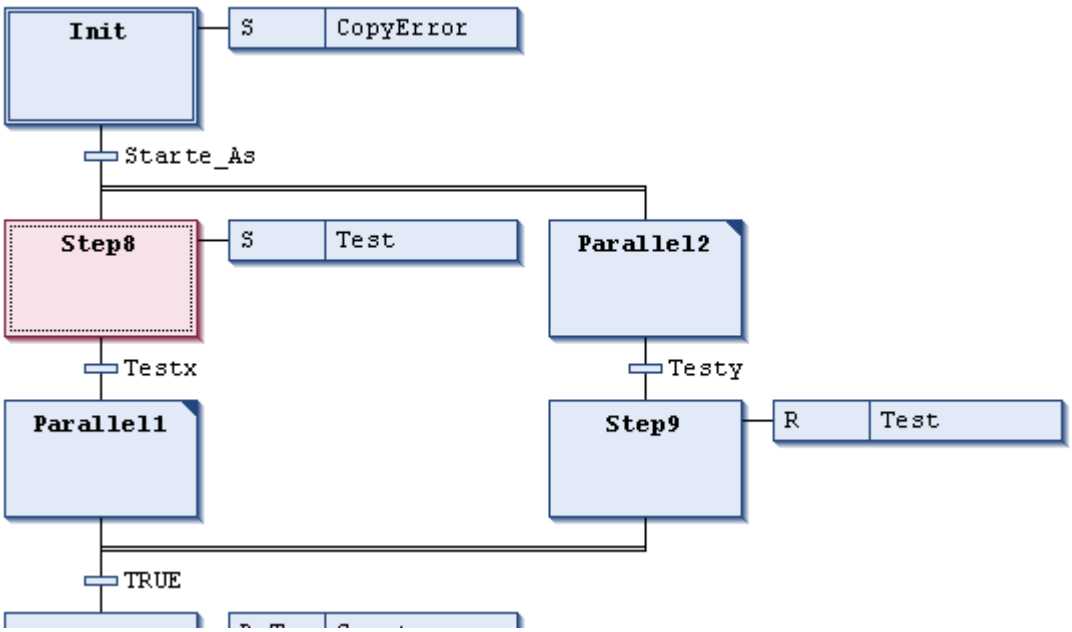


Abb. 400 Example for a sequence of steps in a SFC module

Note: In contrast to »PLC Designer« V2.3 there is only one type of steps combining the functionality of IEC-steps and "3S steps" as used in »PLC Designer« V2.3

13.13.2 Cursor Positions in SFC

A possible cursor position in a SFC diagram in the SFC Editor is indicated per default by a **grey shadow** when moving with the cursor over the elements.

There are two categories of cursor positions: Texts and element bodies. See the possible positions indicated by a grey shaded area as shown in the following pictures:

1. Texts

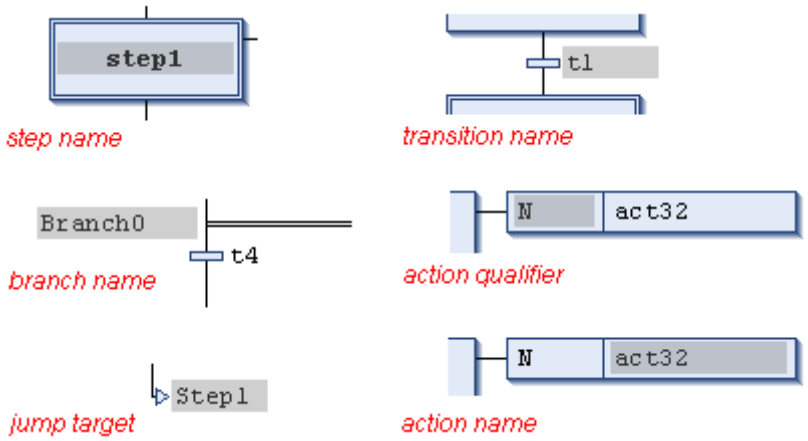


Abb. 401 Possible cursor positions, texts

When you click on a text cursor position, the string will get editable:

Abb. 402 Select action name for editing

2. Element bodies

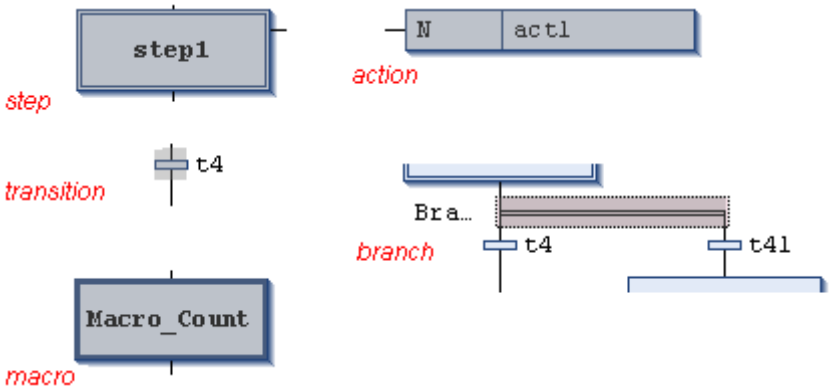


Abb. 403 Possible cursor positions, element bodies

When you click on a shadowed area, the **element will get** selected. It gets a dotted frame and is displayed red-shaded :

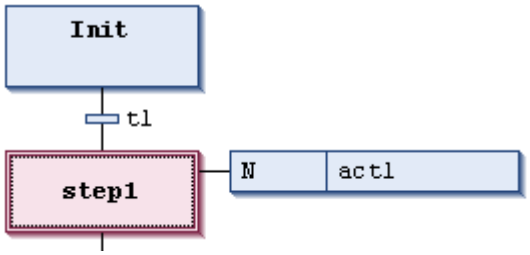


Abb. 404 Selected step element

13.13.3 Working in the SFC Editor

By default a new SFC POU contains an init step and a subsequent transition. For how to add further elements, how to arrange and edit the elements see the following information:

Navigating: Jumping to the next resp. previous element in the chart is possible by using the arrow keys.

Insert elements

The particular SFC elements can be inserted via the respective commands which by default are available in the "SFC" menu. . A double-click on an already inserted step, transition or action element, which does not yet reference a project object, will open a dialog for assigning one.

Select elements

An element resp. text field might be selected by a mouse-click on a possible cursor position. The selection might also always be given to an adjacent element by using the arrow keys. The element will change color to red.

In contrast to **»PLC Designer« V2.3** steps and transitions can be selected and thus also moved (cut, copy, paste) or deleted separately!

Multiple selection is possible by the following:

- Keep the <Shift>-key pressed and subsequently click on the particular elements to be selected.
- Press the left mouse-key and draw a rectangle (dotted line) around the elements to be selected.
- Use command Select All, by default from the Edit menu.

Edit texts

By a mouse-click on a text-cursor-position at once the edit field opens, where you can edit the text. If a text area has been selected via the arrow keys, the edit field must be opened explicitly by using the <Space>-bar.

Edit associated actions

A doubleclick on an step (entry, active or exit) or transition action association opens the associated action in the corresponding editor. For example perform a doubleclick on the transition element or on the triangle indicating an exit action in a step element.

Cut, Copy, Paste elements

Select the element(s) and use command cut, copy resp. paste (by default from the Edit menu) or the corresponding keys.

Notice the following behaviour which is different to **»PLC Designer« V2.3 :**

- When you paste one or several cut or copied element(s), the content of the clipboard will be inserted before the currently selected position. If nothing is currently selected, the element(s) will be appended at the end of the currently loaded chart.
- If you paste a branch while the currently selected element is also a branch, the pasted branch elements will be inserted left to the existing ones.
- If you paste an action (list) at a currently selected step, the actions will be added at the beginning of the action list of the step resp. an action list for the step will be created.
- Incompatible elements when cutting/copying: If an associated action (list) and additionally an element, which is not the step to which the action (list) belongs, are selected, a message box will appear: "The current selection contains incompatible elements. No data will be filed to the clipboard." The selection will not be stored and you cannot paste or copy it somewhere else!
- Incompatible elements when pasting: If you try to paste an action (list) while the currently selected element is not a step or another association, an error box will appear: "The current clipboard content can not be pasted at the current selection." If you try to paste an element like a step, branch or transition when currently an associated action (list) is selected the same message box will appear.

Delete elements:

Select the element(s) and use command Delete or the <Del>-key. Notice the following:

- Deleting a step also deletes the associated action list.
- Deleting the init step automatically sets the following step to be the initial one, that is option 'Initial step' will be activated in the properties of this step.
- Deleting the horizontal line preceding a branched area will delete all branches.
- Deleting all particular elements of a branch will delete the branch.

13.13.4 SFC Element Properties

The properties of a SFC element can be viewed and edited in the "Element properties" window. This window can be opened via command 'Element Properties', which by default is part of the 'View' menu.

It depends on the **currently selected element** which properties are displayed The properties are grouped and the particular group sections can be opened or closed by using the plus- resp. minus signs.

Notice that in the 'View' tab of the SFC Editor options you can configure whether the particular types of properties should be **displayed next to an element** in the SFC chart.

See in the following all possible properties:

Common:

Name	Element name, by default "<element><running number>" , for example step name "Step0", "Step1", branch name "branch0" etc.
Comment	Element comment, text string, for example "Reset the counter". Linebreaks can be inserted via <Ctrl>+<Enter>.
Symbol	For each SFC element implicitly a flag is created, named like the element. Here you can specify whether this flag variable should be exported to the symbol configuration and how the symbol then should be accessible in the PLC. Perform a double-click on the value field, resp. select the value field and use the space-key in order to open a selection list from which you can choose one of the following access options: None: The symbol will be exported to the symbol configuration, but it won't not be accessible in the PLC: Read: The symbol will be exported to the symbol configuration and it will be readable in the PLC. Write: The symbol will be exported to the symbol configuration and it will be written in the PLC. Read/Write: Combination of Read and Write. By default nothing is entered here, which means, that the symbol not at all is exported to the symbol configuration.

Specific:

Initial step	This option always is activated in the properties of the current initial step (init step). By default it is activated for the first step in a SFC and deactivated for other steps. Notice that if you activate this option for another step, you **must deactivate** it in the previous init step in order to avoid a compiler error. In this context also notice command 'Init Step'.
Times:	**Note**: Notice the possibility to detect timeouts in steps by the SFCError flag.
Minimal active	Minimum length of time the processing of this step should take; permissible values: time according to IEC-syntax (for example t#8s) or TIME variable; default: t#0s.
Maximal active	Maximum length of time the processing of this step should take; permissible values: time according to IEC-syntax (for example t#8s) or TIME variable; default: t#0s
Actions:	Define here the actions to be performed when the step is active. Notice the description of the sequence of processing for details.
Step entry	This action will be executed after the step has got active.
Step active	This action will be executed when the step is active and possible entry actions have been already processed.
Step exit	This action will be executed in the subsequent cycle after a step has been deactivated.

Note: Notice the possibility of getting information on step/action status resp. timeouts etc. via the appropriate implicit variables and SFC flags.

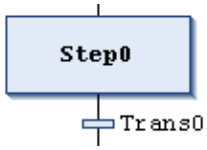
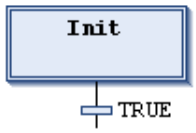
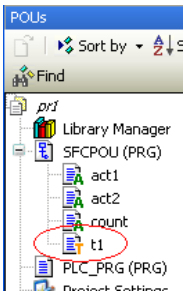
13.13.5 SFC Elements / Toolbox

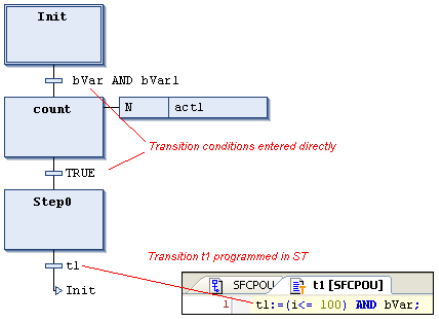
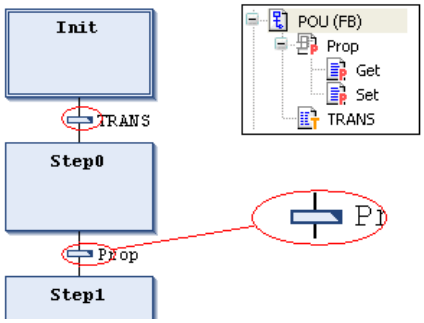
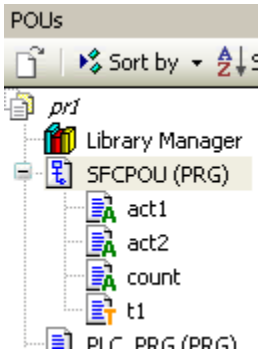
The graphic elements usable for programming in the SFC editor window currently can be inserted by using the insert commands, by default available in the SFC menu.

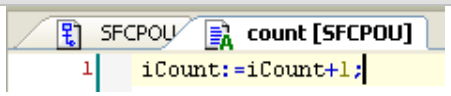
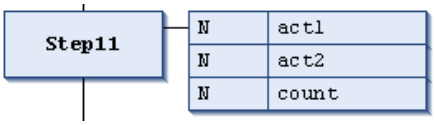
The SFC Toolbox currently is not yet implemented.

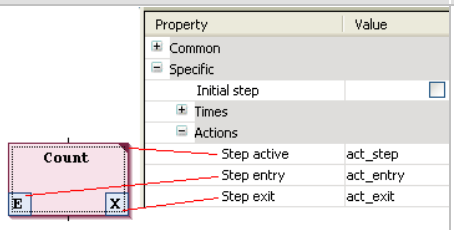
See also the help page on: Working in the SFC Editor.

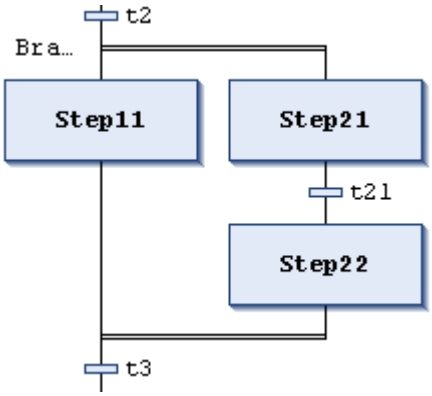
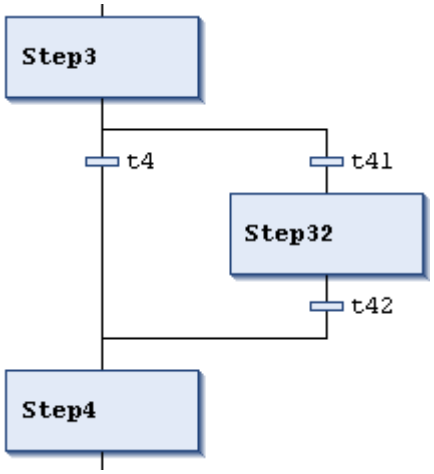
The following elements are available and described in the following table: Step, Transition, Action, Branch (Alternative), Branch (Parallel), Jump, Macro

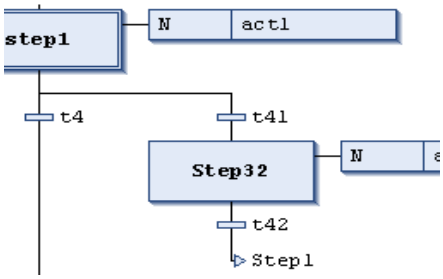
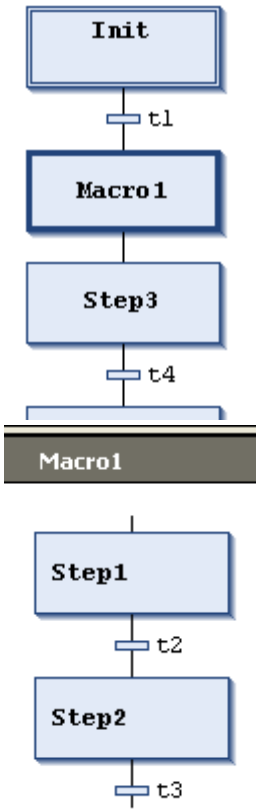
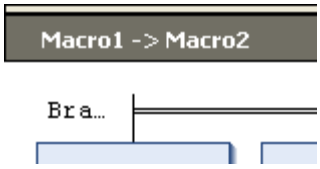
Symbol	Element name / Example	Description
	Step-Transition step and subsequent transition: 	A **step** is represented by a box containing the step name and being connected to the preceding and subsequent transitions by a line. The **step name** can be edited inline. **Note:** Step names must be unique in the scope of the parent POU. Notice this especially when using actions programmed in SFC.
	initial step and subsequent transition: 	The box frame of the **init step** is double-lined. Notice that each step - by command 'Init step' or by activating the respective step property - can be **transformed** to an **init step** , that is to that step, which will be executed first when the IL-POU is called. Each step is defined by the step properties. **Note:** In contrast to »PLC Designer« V2.3 there is only one type of steps, the IEC conforming steps. The actions to be performed when the step is active (processed) are to be associated (see below, action). Steps and transitions are basically **inserted** in combination via command 'Insert step-transition (after)'.
	 *Abb. 405 Transition in POUs tree*	A **transition** is represented by a small rectangle box connected to the preceding and subsequent steps by a line. It provides the condition on which the following step will get active (as soon as the condition is TRUE). By default automatically a transition "**trans<n>**" is inserted, whereby n is a running number. This default name should be modified: A valid name is either - the name of a **transition object** ( ) available in the **POUs tree** (this allows **multiple use** of transitions; see for example "t1" in the left column)

Symbol	Element name / Example	Description
	Examples of transition   *Abb. 406 Transition or property object indicated by a triangle:*	or - a valid **conditional expression**." Regard that transitions which consist of a transition or a property object are indicated by a small **triangle** in the upper right corner of the rectangle. A transition **condition** must have the value TRUE or FALSE. Thus it can consist of either a boolean variable, a boolean address, a boolean constant, or a series of instructions having a boolean result. But a transition may not contain programs, function blocks or assignments! **In contrast to »PLC Designer« V2.3...** ... a transition condition is handled like a method call; It be entered according to the Syntax: <transition name>:=<transition condition>; (for example" trans1:= a=100") or just <transition condition>; (for example "a=100") See also t1in the left column. In **online mode** the subsequent step can only get active if the preceding transition has become TRUE.
	Action  *Abb. 407 Actions in POUs tree*	An action can contain a series of **instructions** written in one of the valid programming languages. It is assigned to a step and **in online mode** it will be processed according to the defined sequence of processing. Each action to be used in SFC steps must be available as a valid **POU** within the SFC POU resp. the project ( ). The 'Add Object' command is available for adding an action POU below a SFC POU. **Note:** Step names must be unique in the scope of the parent POU. An action may not contain a step having the same name like the step to which it is assigned to.

Symbol	Element name / Example	Description
	 *Abb. 408 Example of an action written in ST*  *Abb. 409 IEC conforming step action list associated to a step* Each action box in the first column shows the qualifier and in the second the action name.	There are the following types of actions: **1. IEC conforming step action ("IEC action")**: This is an action according to standard IEC1131-3 which will be processed according to its qualifier when the step has got active and a second time when it has got deactivated. In case of assigning multiple actions to a step (**action list**) the actions will be executed from up to down. • Different qualifiers can be used for IEC step actions in contrast to a "normal" step action. • A further difference to the "normal" step actions is that each IEC step action is provided with a **control** flag, which allows to make sure that - even if the action is called also by another step - the action will get executed always only once at a time. This is not guaranteed with the normal step actions. • An IEC step action is represented by a **bipartite box**, connected to the right of a step via a connection line. In the left part it shows the action **qualifier**, in the right part the **action name**. Both can be edited inline. • IEC step actions get **associated** to a step via the 'Insert action association (after)' command. One or multiple actions can be associated to a step. The position of the new action depends on the current cursor position and the command. The actions must be available in the project and be inserted with a unique action name (for example plc_prg.a1).

Symbol	Element name / Example	Description
	 *Abb. 410 IEC-extending step actions* The "Step active", "Step Entry" and "Step Exit" actions are defined in the step properties.	**2. IEC-extending step actions:** These are actions extending the IEC standard, previously known as "3S step actions": The actions must be available as objects below the SFC object. The action names must be unique. - **"step entry" action**: This type of step action will be processed as soon as the step has become active and before the "step active" action. The action is associated to a step via an entry in the "step entry" field of the step properties. It is represented by an "E" in the lower left corner of the respective step box. - **"step active" action**: This type of step action will be processed when the step has become active and after a possible "step entry" action of this step has been processed. However in contrast to an IEC step action (see above) it is not executed once more when it gets deactivated and it cannot get assigned qualifiers. The action is associated to a step via an entry in the "step active" field of the step properties. It is represented by a small triangle in the upper right corner of the respective step box. - **"step exit" action**: An exit action will be executed once when the step has got deactivated. Notice however that this execution will not be done in the same, but at the beginning of the subsequent cycle. The action is associated to a step via an entry in the "step exit" field of the step properties. It is represented by an "X" in the lower right corner of the respective step box.
		A sequential function chart can diverge, that is the processing line can be branched into two or several further lines ("**branches**"). Parallel branches will be processed parallel (both at a time), in case of alternative branches only one will be processed depending on the preceding transition condition. Each branching within a chart is preceded by a horizontal double (parallel) or simple (alternative) line and also terminated by such a line or by a jump.

Symbol	Element name / Example	Description
	Parallel branch 	A **parallel branch** must begin and end with a step. Parallel branches can contain alternative branches or other parallel branches. The horizontal lines before and after the branched area are **double-lines**. Processing **in online mode**: If the preceding transition (t2 in the example shown on the left) is TRUE, the first steps of all parallel branches will become active (Step11 and Step21). The particular branches will be processed parallel to one another before the subsequent transition (t3) will be noticeed. A parallel branch is **inserted** via command 'Insert branch (right)' when a step is currently selected. Notice that parallel and alternative branches can be **transformed** to each other by the commands Parallel resp. Alternative. This might be useful during programming. Automatically a **branch label** is added at the horizontal line preceding the branching which is named "Branch<n> whereby n is a running number starting with "0". This label can be specified when defining a jump target.
	Alternative branch 	An **alternative branch** must begin and end with a transition. Alternative branches can contain parallel branches and other alternative branches. The horizontal lines before and after the branched area are **simple lines**. If the step which precedes the alternative beginning line is active, then the first transition of each alternative branch will be evaluated from left to right. The first transition from the left whose transition condition has value TRUE, will be opened and the following steps will be activated. Alternative branches are **inserted** via command 'Insert branch (right)' when a transition is currently selected. The horizontal lines before and after the branched area are **simple lines**. Notice that parallel and alternative branches can be **transformed** to each other by the commands Parallel resp. Alternative. This might be useful during programming.

Symbol	Element name / Example	Description
	Jump 	A **jump** is represented by a vertical connection line plus a horizontal arrow and the name of the jump target. A jump defines the next step to be processed as soon as the preceding transition is TRUE. Jumps might be needed because the processing lines must not cross or lead upward. Besides the default jump at the end of the chart a jump may only be used at the end of a branch. It gets **inserted** via command 'Insert jump (after)' when the last transition of the branch is selected. The **target** of the jump is specified by the associated text string which can be edited inline. It can be a step name or the label of a parallel branch.
	Macro Main SFC editor view: Macro editor view for Macro1: 	A **macro** is represented by a bold-framed box containing the macro name. It includes a part of the SFC chart, which thus is not directly visible in the main editor view. The process flow is not influenced by using macros, it is just a way to "hide" some parts of the program, for example in order to simplify the display. A macro box is **inserted** by command 'Insert macro (after)'. The macro name can be edited inline. To open the **macro editor**, perform a double-click on the macro box or use command 'Zoom in to macro'. You can edit here just as in the main editor view and enter the desired section of the SFC chart. To get out use 'Zoom out of macro'. The title line of the macro editor always shows the path of the macro within the current SFC. Example: 

13.13.6 Qualifier

In order to configure in which way the actions should be associated to the IEC steps, some qualifiers are available, which are to be inserted in the qualifier field of an action element.

These qualifiers are handled by the SFCActionControl function block of the **IecSfc.library**, which automatically is included in a project by the SFC plug-ins.

The available qualifiers:

N	Non-stored	The action is active as long as the step is active.
R	overriding Reset	The action gets deactivated.
S	Set (Stored)	The action will be started when the step becomes active and will be continued after the step is deactivated, until the action gets reset.
L	time Limited	The action will be started when the step becomes active and it will continue until the step goes inactive or a set time has passed.
D	time Delayed	A delay timer will be started when the step becomes active. If the step is still active after the time delay, the action will start and continue until it gets deactivated.
P	Pulse	The action will be started when the step becomes active/deactive and will be executed once.
SD	Stored and time Delayed	The action will be started after the set time delay and it will continue until it gets reset.
DS	Delayed and Stored	If the step is still active after the specified time delay, the action will start and it will continue until it gets reset.
SL	Stored and time limited	The action will be started when the step becomes active and it will continue for the specified time or until a reset.

The qualifiers L, D, SD, DS and SL need a time value in the TIME constant format.

Note: When an IEC action has been deactivated it will be executed once more. This means that each action at least is executed twice.

13.13.7 Implicit variables - SFC flags

Each SFC step and IEC action provides implicitly generated variables for watching the status of steps and IEC actions during runtime. Also variables can be defined for watching and controlling the execution of a SFC (timeouts, reset, tip mode). These variables also might be generated implicitly by the SFC object.

Basically for each step and each IEC action an implicit variable is generated. A structure instance, named like the element, for example "step1" for a step with step name "step1". Notice the possibility, to define in the element properties, whether for this flag a symbol definition should be exported to the symbol configuration and how this symbol should be accessible in the PLC.

The data types for those implicit variables are defined in library **IecSFC.library**. This library will automatically be included in the project as soon as an SFC object is added.

Step and Action status and Step time

Basically for each step and each IEC-action an implicit structure variable of type **SFCStepType** resp. **SFCActionType** is created. The structure components (flags) describe the status of a step resp. action or the currently processed time of an active step.

The syntax for the implicitly done variable declaration is:

<stepname> : SFCStepType; resp. _<actionname> : SFCActionType;

Note: In contrast to V2.3 implicit variables for actions always are preceded by an underscore.

The following boolean flags for step or action states are available:

For steps:

<stepname>.x shows the current activation status.

<stepname>._x: shows the activation status for the next cycle.

If <stepname>.x = TRUE, the step will be executed in the current cycle.

If <stepname>._x = TRUE and <stepname>.x = FALSE, the step will be executed in the following cycle, that is <stepname>._x gets copied to <stepname>.x at the beginning of a cycle.

For actions:

_<actionname>.x is TRUE, if the action is executed

_<actionname>._x is TRUE, if the action is active

Symbol generation:

In the element properties of a step or an action you can define, whether for the step or action name flag a **symbol** definition should be added to a possibly created and downloaded symbol application. For this purpose make an entry for the desired access right in column "Symbol" of the element properties view.

Note: The flags described above might be used to force a certain status value for a step, that is for setting an step active, but be aware that this will effect uncontrolled states within the SFC !

Time via TIME variables:

The flag "t" gives the current time span which has passed since the step had got active; this is only for steps, no matter whether there is a minimum time configured in the step attributes or not (see also below: SFCError)

For steps: **<stepname>.t** (<stepname>._t not usable for external purposes)

For actions: the implicit time variables are not used.

Control of SFC execution (timeouts, reset, tip mode)

Some implicitly available variables, also named SFC flags, (see table below) can be used to control the operation of an SFC, for example for indicating time overflows or enabling tip mode for switching transitions.

In order to be able to access these flags and to get them work, they **must be declared and activated**. This **is to be done in the** SFC Settings dialog which is a subdialog of the object properties dialog. Manual declaration, as it was needed in V2.3, is only necessary to enable write access from another POU (see below, Accessing flags). In this case however regard the following: If you declare the flag **globally**, you must deactivate the "Declare" option in the SFC Settings dialog in order not to get a implicitly declared local flag, which then would be used instead of the global one ! Keep in mind, that the SFC Settings for a SFC POU initially are determined by the definitions currently set in the SFC opfions dialog.

Notice that a declaration of a flag variable solely done via the SFC Settings dialog will **only be visible in the online view** of the SFC POU !

Example of the use of SFCError

There is a SFC POU named "sfc1" containing a step "s2" which has time limits defined in its step attributes. See the attributes displayed below in figure "Online view of SFC sfc1".

If for any reason the step s2 stays active for a longer time than allowed by its time properties (time overflow), a SFC flag will be set which might be read by the application.

In order to allow this access, declare the flag in the SFC Settings dialog. For this purpose select sfc1 in the Devices resp. POUs window and choose command 'Properties' from the context menu. Open subdialog 'SFC Settings' and there, on tab 'Flags', set a checkmark each in column 'Declare' and 'Use' for flag "SFCError". (Solely declaring would make the variable visible in the online view of the sfc1 declaration part, but it would be without function.)

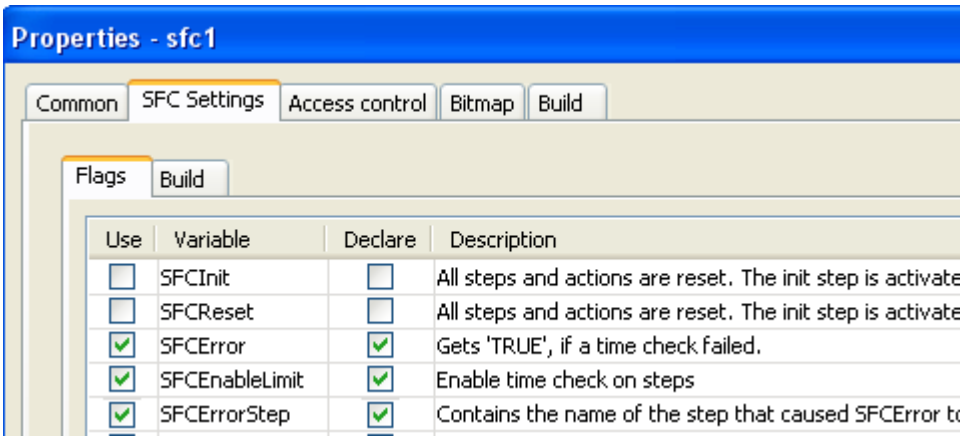


Abb. 411 SFC Settings

Now you can read SFCError within the SFC, for example in an action, via "SFCError", or from another POU via "sfc1.SFCError".

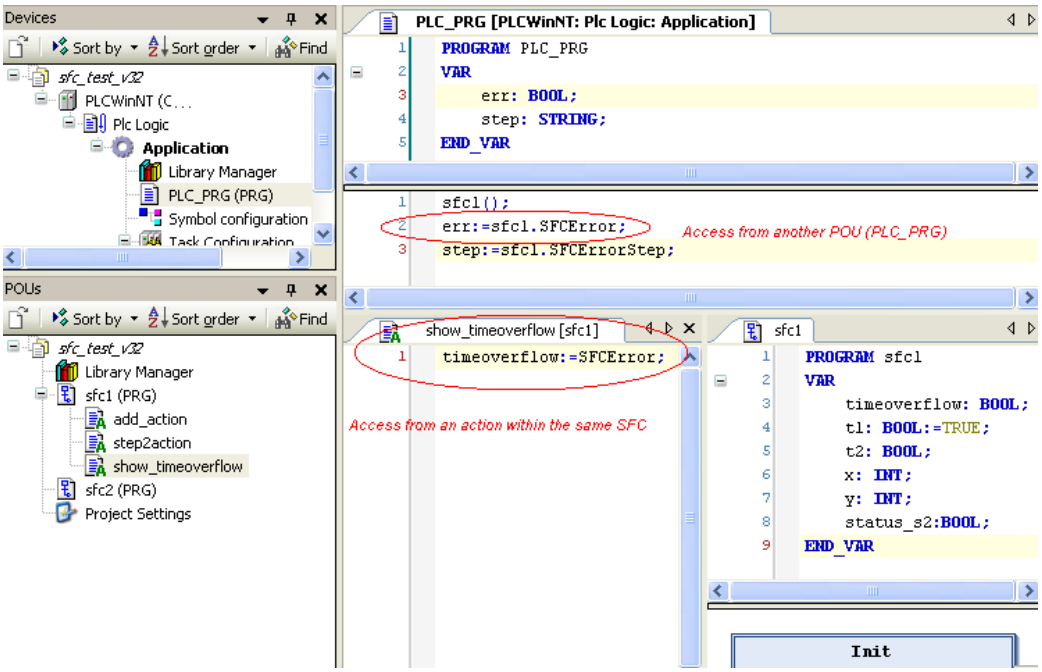


Abb. 412 Accessing SFCError

SFCError will get TRUE as soon as a timeout within sfc2 occurs.

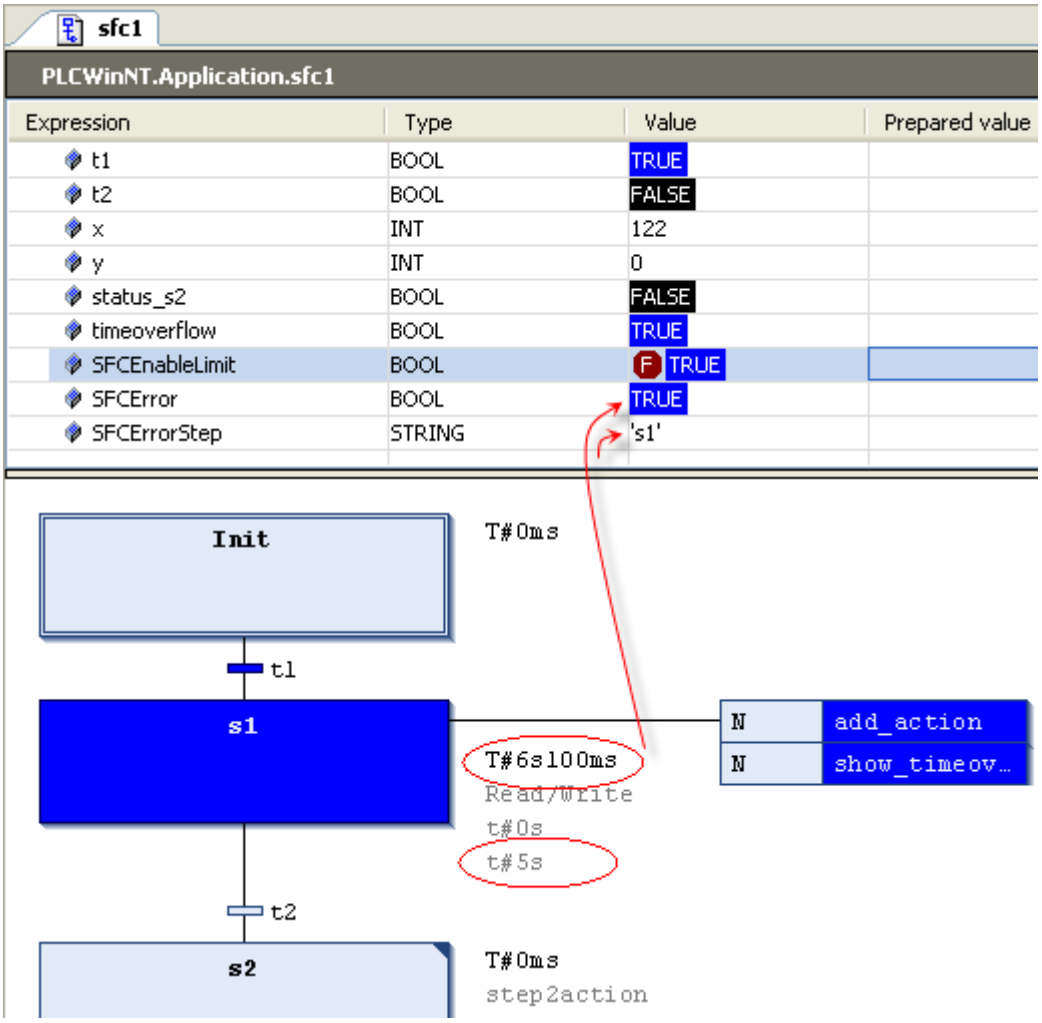


Abb. 413 Online view of SFC sfc1

The following implicit variables (flags) can be used. For this purpose they must be declared and activated in the SFC Settings:

SFCInit: BOOL;	If this variable gets TRUE, the sequential function chart will be set back to the Init step. All steps and actions and other SFC flags will be reset (initialization). The Init step will remain active, but not be executed as long as the variable is TRUE. SFCInit must be set back to FALSE in order to get back to normal processing.
SFCReset: BOOL;	This variable behaves similarly to SFCInit. Unlike the latter however, further processing takes place after the initialization of the Init step. Thus in this case for example a reset to FALSE of the SFCReset flag could be done in the Init step.
SFCError: BOOL;	As soon as any timeout occurs at one of the steps in the SFC, this variable will get TRUE. Precondition: SFCEnableLimit must be TRUE. Notice that any further timeout can not be registered before a reset of SFCError. SFCError must be defined, if you want to use the other time-controlling flags (SFCErrorStep, SFCErrorPOU, SFCQuitError).
SFCEnableLimit: BOOL;	This variable can be used for the explicit activation (TRUE) and deactivation (FALS) of the time control in steps via SFCError. This means, that If this variable is declared and activated (SFC Settings) then it must be set TRUE in order to get SFCError working. Otherwise any timeouts of the steps will not be registered. The usage might be reasonable during start-ups or at manual operation. If the variable is not defined, SFCError will work automatically. Of course as a precondition SFCError must be defined!
SFCErrorStep: STRING;	This variable stores the name of a step at which a timeout was registered by SFCError. timeout. Precondition: SFCError must be defined!
SFCErrorPOU: STRING;	This variable stores the name of the SFC POU in which a timeout has occurred. Precondition: SFCError must be defined!
SFCQuitError: BOOL;	As long as this variable is TRUE, the execution of the SFC diagram is stopped and variable SFCError will be reset. As soon as the variable has been reset to FALSE, all current time states in the active steps will be reset. Precondition: SFCError must be defined!
SFCPause: BOOL;	As long as this variable is TRUE, the execution of the SFC diagram is stopped.
SFCTrans: BOOL;	This variable gets TRUE, as soon as a transition is actuated.
SFCCurrentStep: STRING;	This variable stores the name of the currently active step, independently of the time monitoring. In case of simultaneous sequences the name of the outer right step will be registered.
SFCTip, **SFCTipMode**: BOOL;	This variables allow using the inching mode within the current chart. When this mode has been switched on by SFCTipMode=TRUE, you can only skip to the next step by setting SFCTip=TRUE (rising edge). As long as SFCTipMode is set to FALSE, it is possible to skip by the transitions.

A timeout has been detected in step 's1' in SFC object 'POU' by flag SFCError.

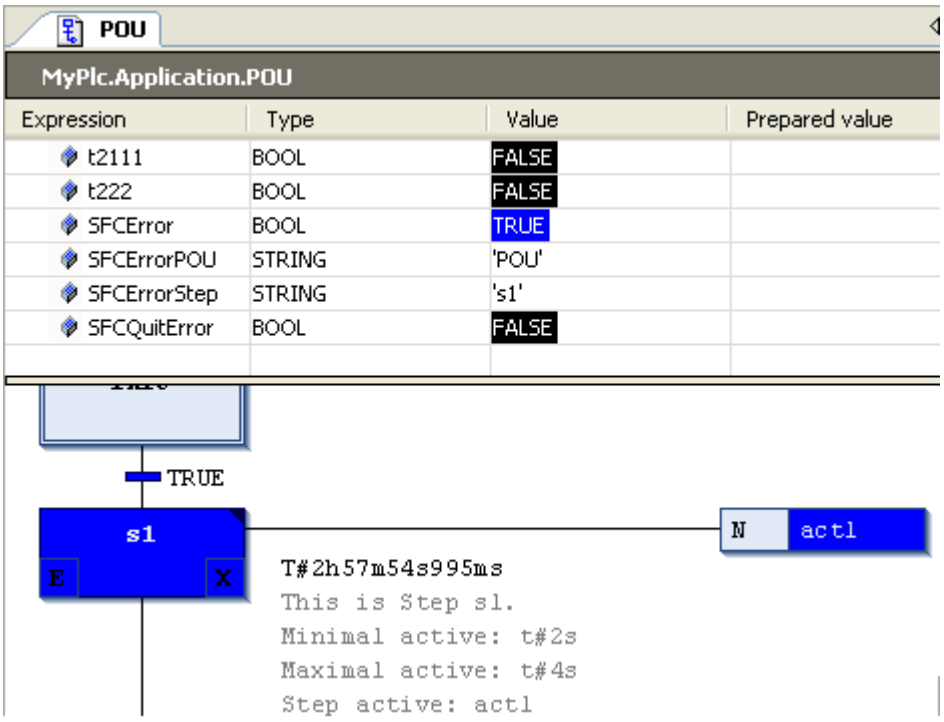


Abb. 414 Example of some SFC error flags in online mode of the editor

Accessing flags:

For enabling access on the flags for the control of SFC execution (timeouts, reset, tip mode), the flag variables must be declared and activated as described above (Control of SFC execution).

Syntax for accessing:

... from an action or transition within the SFC POU: **<stepname>.<flag>** resp. **_<actionname>.<flag>**
Examples: "status:=step1._x;" "checkerror:=SFCerror;"

... from another POU: **<SFC POU>.<stepname>.<flag>** resp. **<SFC POU>_<actionname>.<flag>**
Examples: "status:=SFC_prog.step1._x;" checkerror:=SFC_prog.SFCerror;"

Regard that in case of **write access** from another POU the implicit variable additionally must be declared explicitly as a **VAR_INPUT** variable of the SFC POU or **globally** e.g. in a GVL.

Example:

Local declaration:

```
PROGRAM SFC_prog
VAR_INPUT
  SFCinit:BOOL;
END_VAR
```

or Global declaration in a GVL:

```
VAR_GLOBAL
SFCinit:BOOL;
END_VAR
```

Accessing the flag in PLC_PRG:

```
PROGRAM PLC_PRG
VAR
  setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;  //write access on SFCinit in SFC_prog
```

13.13.8 Sequence of processing in SFC

In online mode the particular action types will be processed according a defined sequence, see the table below.

First note the following use of terms:

- **Active step**: A step, whose step action is being executed, is called "active". In Online mode active steps are filled with blue color.
- **Initial step**: In the first cycle after a SFC POU has been called, the initial step automatically gets active and the associated step action is executed.
- **IEC actions** are executed at least twice: The first time when they have got active, the second time - in the following cycle - when they have been deactivated.
- **Alternative Branches**: If the step preceding the horizontal start line of alternative branches is active, then the first transition of each particular branch will be evaluated from left to right. The first transition from the left whose transition condition has value TRUE will be searched and the respective branch will be executed, that is the subsequent step within this branch will get active.
- **Parallel Branches**: If the double-line at the beginning line of parallel branches is active and the preceding transition condition has the value TRUE, then in all parallel branches each the first step will get active. The branches now will be processed parallel to one another. The step subsequent to the double-line at the end of the branching will become active when all previous steps are active and the transition condition after the double-line has the value TRUE.

Processing order of elements in a sequence (matching the processing order as used in »PLC Designer« V2.3):

1. Reset	All action control flags of the IEC actions get **re-set** (not however the flags of IEC actions that are called within actions!).
2. Step Exit Actions	All steps are checked in the order which they assume in the sequence chart (top to bottom and left to right) to determine whether the requirement for execution of the **step exit action** is provided, and - if that is the case - this will be executed. An exit action will be executed, if the step is going to get deactivated, that is if its entry and step actions - if existing - have been executed during the last cycle, and if the transition for the following step is TRUE.
3. Step Entry Actions	All steps are tested in the order which they assume in the sequence to determine whether the requirement for execution of the **step entry action** is provided and - if that is the case - this will be executed. An entry action will be executed, if the step-preceding transition condition is TRUE and thus the step has been activated.
4. Timeout check, Step Active Actions	For all steps , the following is done in the order which they assume in the sequence: - (not yet implemented) If applicable, the elapsed time is copied into the corresponding implicit step status variable <stepname>.t - (not yet implemented) If applicable, any timeout is tested and the SFC error flags are serviced as required. - For non-IEC steps, the corresponding **step active action** is now executed.
5. IEC Actions	**IEC actions** that are used in the sequence are executed in alphabetical order. This is done in two passes through the list of actions. In the first pass, all the IEC actions that are deactivated in the current cycle are executed. In the second pass, all the IEC actions that are active in the current cycle are executed.
6. Transition check, Activating next steps	Transitions are evaluated: If the step in the current cycle was active and the following transition returns TRUE (and if applicable the minimum active time has already elapsed), then the following step is **activated**.

Note:
The following must be noticeed concerning implementation of actions:
It can come about that an action is carried out several times in one cycle because it is associated with multiple sequences. (For example, an SFC could have two IEC actions A and B, which are both implemented in SFC, and which both call IEC action C; then in IEC actions A and B can both be active in the same cycle and furthermore in both actions IEC action C can be active; then C would be called twice).
If the same IEC action is used simultaneously in different levels of an SFC, this could lead to undesired effects due to the processing sequence described above. For this reason, an error message is issued in this case. It can possibly arise during processing of projects created with older versions of the programming system.

Note: Notice the possibility of using implicit variables for controlling the status of steps and actions resp. the execution of the chart.

13.13.9 SFC Editor in online mode

In online mode the SFC-Editor provides views for monitoring and for writing and forcing the variables and expressions on the controller. Debugging functionality (breakpoints, stepping etc.) is not yet available. See below.

- Notice that the editor window of an SFC object also includes the Declaration Editor in the upper part. In case of having declared implicit variables (SFC flags) via the SFC Settings dialog, those will be added here, but will not be viewed in the offline mode of the declaration editor.
- Also please notice the sequence of processing of the elements of a sequential function chart.
- See the object properties resp. the SFC Editor options and SFC defaults for settings concerning compilation resp. online display of the SFC elements and their attributes.
- Consider the possible use of flags for watching and controlling the processing of an SFC.

Monitoring

Active steps are displayed filled blue-colored. The display of step attributes depends on the currently set SFC Editor options.

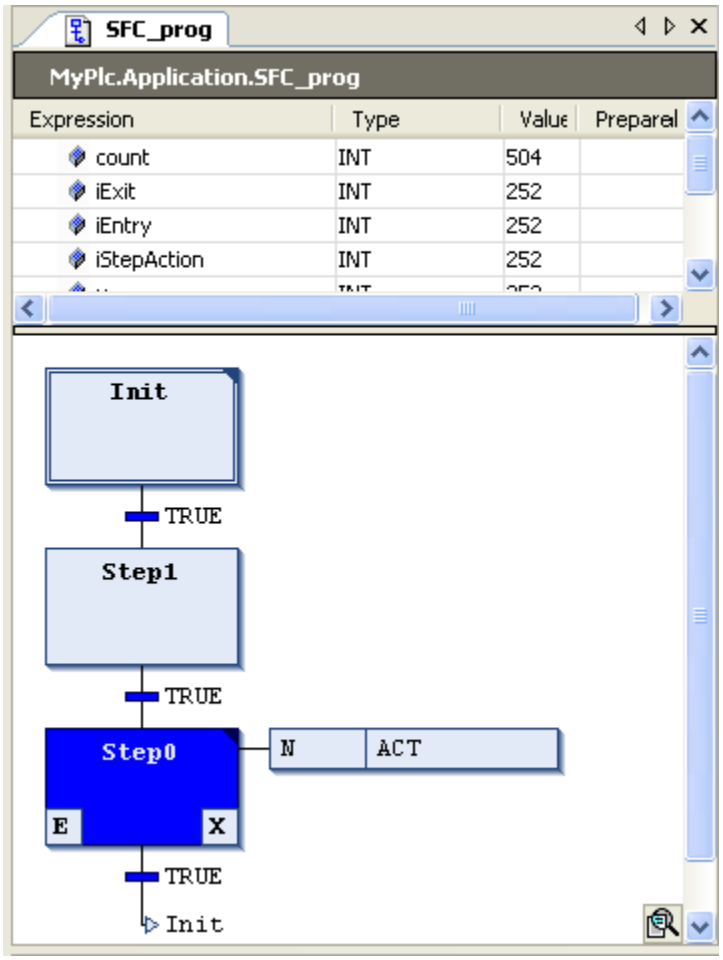


Abb. 415 Example: Online view of a program object SFC_prog

Breakpoint positions in SFC Editor: (not yet implemented)

The possible breakpoint positions basically are those positions in a POU at which values of variables can change or at which the program flow branches out resp. another POU is called.

Note: Notice for breakpoints in methods: A breakpoint will be set automatically in all methods which might be called. So, if an interface-managed method is called, breakpoints will be set in all methods of function blocks implementing that interface and also in all derivative function blocks subscribing the method. If a method is called via a pointer on a function block, breakpoints will be set in the method of the function block and in all derivative function blocks which are subscribing the method.

13.14 ST Editor

The ST-Editor, provided by the 'ST Editor' plug-in, is used to create programming objects in the IEC programming language Structured Text (ST) resp. Extended Structured Text which provides some extensions to the IEC 61131-3 standard.

The ST-Editor is a **text editor** and thus the corresponding text editor settings in the Options and Customize dialogs can be used to configure behaviour, appearance and menus. There you can define the default settings for highlight coloring, line numbers, tabs, indenting and many more.

not yet implemented: The usual Windows functions are available and even those of the IntelliMouse can be used if the corresponding driver has been installed.

Notice that **block selection** is possible by pressing <Alt> while selecting the desired text area with the mouse.

The editor will be available in the lower part of a window which also includes the Declaration Editor in the upper part.

Notice that in case of **syntactic errors** during editing the corresponding messages will be displayed in the "Precompile" Messages window. An update of this window is done each time you re-set the focus to the editor window (for example put cursor in another window and then back to the editor window).

13.14.1 ST-Editor in Online Mode

In online mode the Structured Text Editor (ST-Editor) provides views for monitoring and for writing and forcing the variables and expressions on the controller. Debugging functionality (breakpoints, stepping etc.) is available. See below.

Monitoring

If the inline monitoring is not explicitly de-activated in the Options dialog, small monitoring windows will be displayed behind each variable showing the actual value (**Inline Monitoring**).

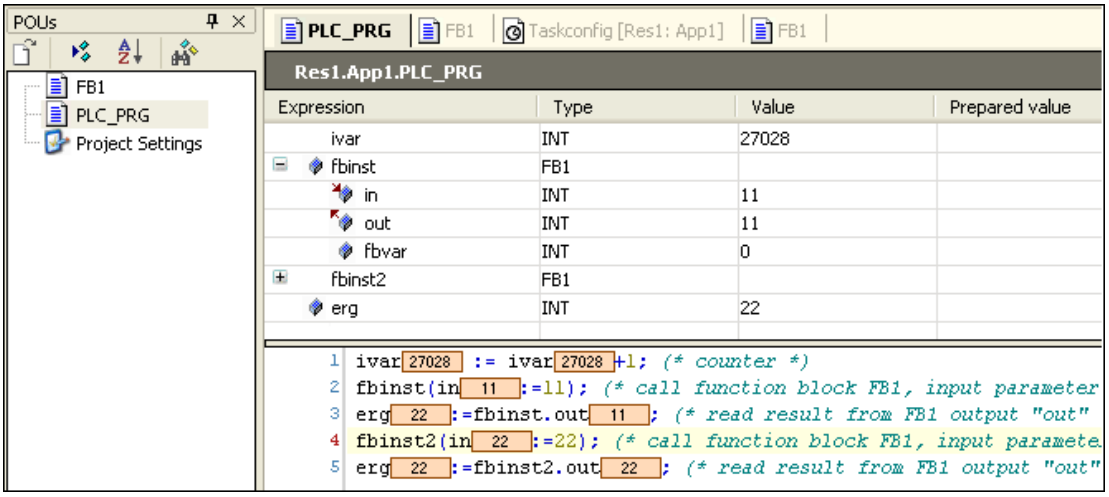


Abb. 416 Example: Online view of a program object PLC_PRG

Online view of a function block POU:

No values will be viewed. Instead the term "<Value of the expression>" will be displayed in column 'Value' and the inline monitoring fields in the implementation part will show three question marks each.

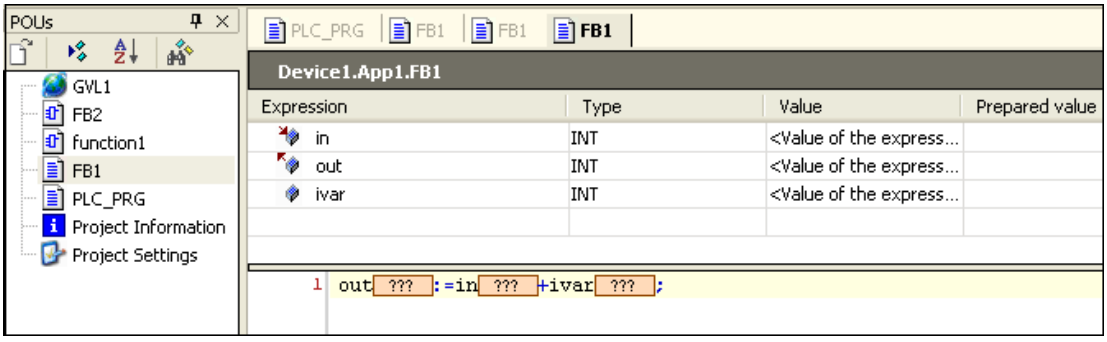


Abb. 417 Example: Online view of a function block FB1

Forcing of variables

In addition to the possibility to enter a prepared value for a variable within the declaration of any editor the ST-Editor provides to click on the monitoring box of a variable within the implementation part (in online mode), whereon you may enter the prepared value in the rising dialogue:

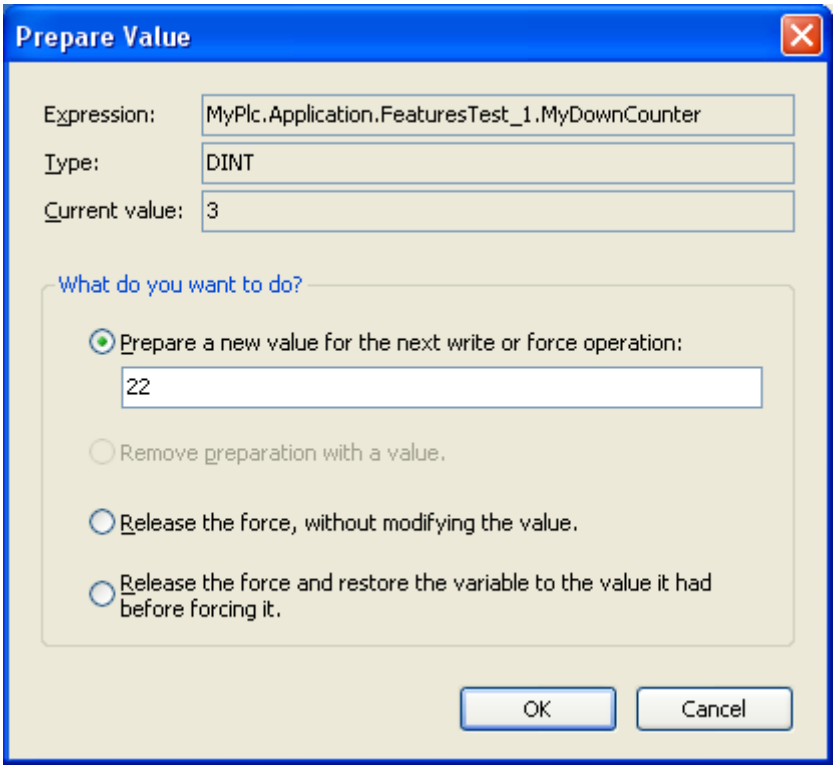


Abb. 418 Dialog - Prepare Value

You find the name of the variable completed by its path within the device tree ("Expression"), its type and current value. By activating the corresponding item you may choose wether you want to

- prepare a new value which has to be entered in the edit field thereby
- remove a prepared value
- release the variable actually forced
- release the variable actually forced and reset it to the value it was assigned to just before forcing

The selected action will be carried out on executing the menu command **Force Values** (item Online) or pressing **F7**.

Breakpoint positions in ST-Editor:

The user can set a breakpoint basically at those positions in a POU at which values of variables can change or at which the program flow branches out resp. another POU is called. In the following descriptions "{BP}" indicates a possible breakpoint position:

Assignment: At the beginning of the line. Notice: Assignments as expressions define no further breakpoint positions within a line.

FOR-Loop: 1. before the initialization of the counter, 2. before the test of the counter. 3. before a statement

{BP} FOR i := 12 TO {BP} x {BP} BY 1 DO

{BP} [statement1]

...

{BP} [statementn-2]

END_FOR

WHILE-Loop: 1. before the test of the condition. 2. before a statement

{BP} WHILE i < 12 DO

{BP} [statement1]

...

{BP} [statementn-1]

END_WHILE

REPEAT-Loop: before the test of the condition.

REPEAT

{BP} [statement1]

...

{BP} [statementn-1]

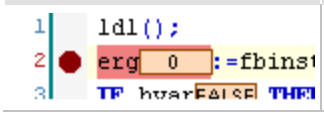
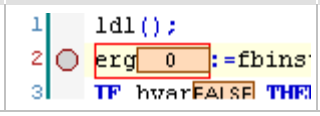
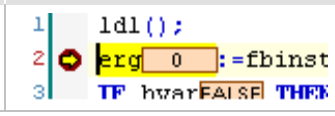
{BP} UNTIL i >= 12

END_REPEAT

Call of a program or a function block: At the beginning of the line.

At the end of a POU. When stepping through, this position also will be reached after a RETURN instruction.

Breakpoint display in ST:

Breakpoint in online mode:	Disabled breakpoint:	Program stop at breakpoint:
		

Note: The following must be noticeed for breakpoints in methods: A breakpoint will be set automatically in all methods which might be called. So, if an interface-managed method is called, breakpoints will be set in all methods of function blocks implementing that interface and also in all derivative function blocks subscribing the method. If a method is called via a pointer on a function block, breakpoints will be set in the method of the function block and in all derivative function blocks which are subscribing the method.

13.14.2 Structured Text ST / ExST

Structured Text is a textual high-level programming language, similar to PASCAL or C. The program code is composed of expressions and instructions. In contrast to IL (Instruction List) numerous constructions can be used for programming loops, thus allowing the development of complex algorithms.

Example:

```
IF value < 7 THEN
 WHILE value < 8 DO
  value:=value+1;
 END_WHILE;
END_IF;
```

Extended Structured Text (ExST) is a »PLC Designer«-specific extension to the IEC 61131-3 standard for Structured Text (ST). Examples: Assignment as expression, Set-/Reset-Operators.

Expressions

An expression is a construction which after its evaluation returns a value This value is used in instructions.

Expressions are composed of operators, operands and/or assignments. An operand can be a constant, a variable, a function call or another expression.

Examples:

33	(* constant *)
ivar	(* variable *)
fct(a,b,c)	(* function call *)
a AND b	(* expression *)
(x*y) / z	(* expression *)
real_var2 := int_var;	(* assignment, see below *)

Valuation of expressions

The evaluation of expression takes place by means of processing the operators according to certain binding rules. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed.

Operators with equal binding strength are processed from left to right.

Below you find a table of the ST operators in the order of their binding strength:

Operation	Symbol	Binding strength
Put in parentheses	(expression)	Strongest binding
Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negate Building of complements	- NOT	
Multiply Divide Modulo	* / MOD	
Add Subtract	+ -	
Compare	<,>,<=,>=	
Equal to Not equal to	= <>	
Boolean AND	AND	
Boolean XOR	XOR	
Boolean OR	OR	Weakest binding

Assignment as expression

As extension to the IEC 61131-3 standard (ExST), »PLC Designer« allows assignments to be used as an expression.

Examples:

int_var1 := int_var2 := int_var3 + 9;	(* int_var1
real_var1 := real_var2 := int_var;	(* correct assignments, real_var1 and real_var2 will get the value of int_var *)
int_var := real_var1 := int_var;	(* this will lead to an error message because of type mismatch real-int ! *)

IF b := (i = 1) THEN
i := i + 1;
END_IF	

Instructions

Instructions tell what to do with the given expressions. The following instructions can be used in ST:

Instruction type	Example
Assignment (see also: Assignment operators)	A:=B; CV := CV + 1; C:=SIN(X);
Calling a function block and use of the FB output	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
RETURN	RETURN;

IF	D:=B*B;
	IF D<0.0 THEN
	C:=A;
	ELSIF D=0.0 THEN
	C:=B;
	ELSE
	C:=D;
	END_IF;

CASE	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;

Instruction type	Example

FOR	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I;
	EXIT;
	END_IF;
	END_FOR;

WHILE	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
REPEAT	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
EXIT	EXIT;
CONTINUE	CONTINUE;
JMP	label: i:=i+1; JMP label;
Empty instruction	;

Assignment operator

On the left side of an assignment there is an operand (variable, address) to which he value of the expression on the right side is assigned by the assignment operator **:=**

See also the MOVE operator which does the same.

Example:

```
Var1 := Var2 * 10;
```

After completion of this line Var1 has the tenfold value of Var2.

Notice the following features which are new in comparison to »PLC Designer« V2.3:

Further assignment operators, which are not part of the 61131-3 standard (**ExST**):

Set operator '**S=**': The value will be "set", that is if once set to TRUE will remain TRUE.

Example:

```
a S= b;
```

a gets the value of b; if once set to TRUE it will remain true, even if b gets FALSE again.

Reset operator 'R=': The value will be "reset", that is if once set to FALSE, it will remain FALSE,

Example:

```
a R= b;
```

a gets set to FALSE when b = TRUE.

Note: Notice the behaviour in case of a multiple assignment: All set and reset assignments refer to the last member of the assignment.
Example:
a S= b R= fun1(par1,par2);
In this case b gets the reset output value of fun1, BUT: a does **not** get the set value of b, but gets the set output value of fun1 !

Assignment as expression, extension to the IEC 61131-3 standard (ExST):

Notice that an assignment can be used as an expression.

Calling function blocks in ST

A function block (abbreviated by "FB" in the following) is called in Structured Text according to the following syntax:

Syntax:

<name of FB instance>(FB input variable:=<value or address>|, <further FB input variable:=<value or address>|...further FB input variables);

Example:

In the following example a timer function block (TON) is called with assignments for the parameters IN and PT.

Then result variable Q is assigned to variable A. The timer FB is instantiated by "TMR:TON;"

The result variable, as in IL, is addressed according to syntax <FB instance name>.<FB variable>:

```
TMR(IN := %IX5, PT := 300);
A:=TMR.Q
```

RETURN instruction

The RETURN instruction can be used to leave a POU, for example depending on a condition.

Syntax:

RETURN;

Example:

```
IF b=TRUE THEN
RETURN;
END_IF;
a:=a+1;
```

If b is TRUE, instruction a:=a+1; will not be executed, the POU will be left immediately.

IF instruction

With the IF instruction you can check a condition and, depending upon this condition, execute instructions.

Syntax:

IF <Boolean_expression1> **THEN**

<IF_instructions>

{ELSIF <Boolean_expression2> **THEN**

<ELSIF_instructions1>

.

.

ELSIF <Boolean_expression n> **THEN**

<ELSIF_instructions n-1>

ELSE

<ELSE_instructions>**}**

END_IF;

The part in braces {} is optional.

If the <Boolean_expression1> returns TRUE, then only the <IF_Instructions> are executed and none of the other instructions.

Otherwise the Boolean expressions, beginning with <Boolean_expression2>, are evaluated one after the other until one of the expressions returns TRUE. Then only those instructions after this Boolean expression and before the next ELSE or ELSIF are evaluated.

If none of the Boolean expressions produce TRUE, then only the <ELSE_instructions> are evaluated.

Example:

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

Here the heating is turned on when the temperature sinks below 17 degrees. Otherwise it remains off.

CASE instruction

With the CASE instructions one can combine several conditioned instructions with the same condition variable in one construct.

Syntax:

CASE <Var1> **OF**

<Value1>**:** <Instruction 1>

<Value2>**:** <Instruction 2>

<Value3, Value4, Value5>: <Instruction 3>

<Value6 .. Value10>: <Instruction 4>

...

<Value n>**:** <Instruction n>

ELSE <ELSE instruction>

END_CASE;

A CASE instruction is processed according to the following model:

- If the variable in <Var1> has the value <Value i>, then the instruction <Instruction i> will be executed.
- If <Var 1> has none of the indicated values, then the <ELSE Instruction> will be executed.
- If the same instruction is to be executed for several values of the variables, then one can write these values one after the other separated by commas and thus condition the common execution.
- If the same instruction is to be executed for a value range of a variable, one can write the initial value and the end value separated by two dots. So you can condition the common condition.

Example:

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
 BOOL3 := FALSE;
2: BOOL2 := FALSE;
 BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
 BOOL3:= TRUE;
ELSE
 BOOL1 := NOT BOOL1;
 BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR loop

With the FOR loop one can program repeated processes.

Syntax:

INT_Var :INT;

FOR <INT_Var> := <INIT_VALUE> **TO** <END_VALUE> **{BY** <Step size>**} DO**

<Instructions>

END_FOR;

The part in braces {} is optional.

The <Instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before executing the <Instructions> so that the <instructions> are never executed if <INIT_VALUE> is greater than <END_VALUE>.

When <Instructions> are executed, <INT_Var> is increased by <Step size>. The step size can have any integer value. If it is missing, then it is set to 1. The loop must also end since <INT_Var> only becomes greater.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Let us assume that the default setting for Var1 is "1". Then it will have the value 32 after the FOR loop.

Note: If <END_VALUE> is equal to the limit value of counter <INT_VAR>, for example if Counter - used in the example shown above - is of type SINT and if <END_VALUE> is 127, then you will get an endless loop. So <END_VALUE> must not be equal to the limit value of the counter !

Extension to the IEC 61131-3 standard (**ExST**):

The CONTINUE instruction can be used within a FOR loop.

WHILE loop

The WHILE loop can be used like the FOR loop with the difference that the break-off condition can be any Boolean expression. This means you indicate a condition which, when it is fulfilled, the loop will be executed.

Syntax:

WHILE <Boolean expression> DO

<Instructions>

END_WHILE;

The <Instructions> are repeatedly executed as long as the <Boolean_expression> returns TRUE. If the <Boolean_expression> is already FALSE at the first evaluation, then the <Instructions> are never executed. If <Boolean_expression> never assumes the value FALSE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example, by counting up or down one counter.

Example:

```
WHILE Counter<>0 DO
 Var1 := Var1*2;
 Counter := Counter-1;
END_WHILE
```

The WHILE and REPEAT loops are, in a certain sense, more powerful than the FOR loop since one doesn't need to know the number of cycles before executing the loop. In some cases one will, therefore, only be able to work with these two loop types. If, however, the number of the loop cycles is clear, then a FOR loop is preferable since it allows no endless loops.

Extension to the IEC 61131-3 standard (**ExST**):

The CONTINUE instruction can be used within a WHILE loop.

REPEAT loop

The REPEAT loop is different from the WHILE loop because the break-off condition is checked only after the loop has been executed. This means that the loop will run through at least once, regardless of the wording of the break-off condition.

Syntax:

REPEAT

<Instructions>

UNTIL <Boolean expression>

END_REPEAT;

The <Instructions> are carried out until the <Boolean expression> returns TRUE.

If <Boolean expression> is produced already at the first TRUE evaluation, then <Instructions> are executed only once. If <Boolean_expression> never assumes the value TRUE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example by counting up or down one counter.

Example:

```
REPEAT
 Var1 := Var1*2;
 Counter := Counter-1;
UNTIL
 Counter=0
END_REPEAT;
```

Extension to the IEC 61131-3 standard (**ExST**):

The CONTINUE instruction can be used within a REPEAT loop.

CONTINUE instruction

As an extension to the IEC 61131-3 standard (**ExST**) the CONTINUE instruction is supported within FOR, WHILE and REPEAT-loops.

CONTINUE makes the execution proceed with the next loop-cycle.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
CONTINUE; (* to avoid division by zero *)
END_IF
Var1:=Var1/INT1; (* only executed, if INT1 is not "0" *)
END_FOR;
Erg:=Var1;
```

EXIT instruction

If the EXIT instruction appears in a FOR, WHILE, or REPEAT loop, then the innermost loop is ended, regardless of the break-off condition.

JMP instruction

The JMP instruction can be used for an unconditional jump to a code line marked by a jump label.

Syntax:

<label>:

JMP <label>;

The <**label**> is an arbirary, but unambiguous identifier that is placed at the beginning of a program line. The instruction JMP has to be followed by the indication of the jump destination that has to equal a predefined label. When arriving the JMP instruction a flyback to the program line that is provided with the indicated label will be effected.

Note: The programmer has to avoid the creation of endless loops, for example by subjecting the jump to an IF condition.

Example:

```
i:=0;
label1: i:=i+1;
(*instructions*)
IF (i<10) JMP label1;
```

As long as the variable i being initialized with 0 has a value less than 10, the conditional jump instruction of the example above will effect a repeated flyback to the program line provided with label label1 and therefor it will effect a repeated processing of the instructions comprised between the label and the JMP instruction. Since these instructions include also the increment of the variable i, we can be sure that the jump condition will be infringed (at the 9th check) and program flow will be proceeded.

This functionality may also be achieved by using a WHILE or REPEAT loop in the example. Generally the use of jump instructions can and should be avoided, because they reduce the readability of the code.

Comments in ST

There are two possibilities to write comments in a Structured Text object.

1. Start the comment with „(*" and close it with „*)". This allows comments which run over several lines. Example: "(*This is a comment.*)"

2. Single line comments as an extension to the IEC 61131-3 standard: "//" denotes the start of a comment that ends with the end of the line. Example: "// This is a comment."

The comments can be placed everywhere within the declaration or implementation part of the ST-Editor.

Nested comments: Comments can be placed within other comments.

Example:

```
(*
a:=inst.out; (* to be checked *)
b:=b+1;
*)
```

In this example the comment that begins with the first bracket is not closed by the bracket following "checked," but only by the last bracket.

13.15 Symbol Configuration Editor

The symbol configuration functionality allows to create symbol descriptions, via which project variables can be accessed from external, for example by an OPC Server.

The configuration of symbols for an application can be done in the **Symbol Configuration Editor** view, which opens on a double-click on the symbol configuration object entry in the device tree.

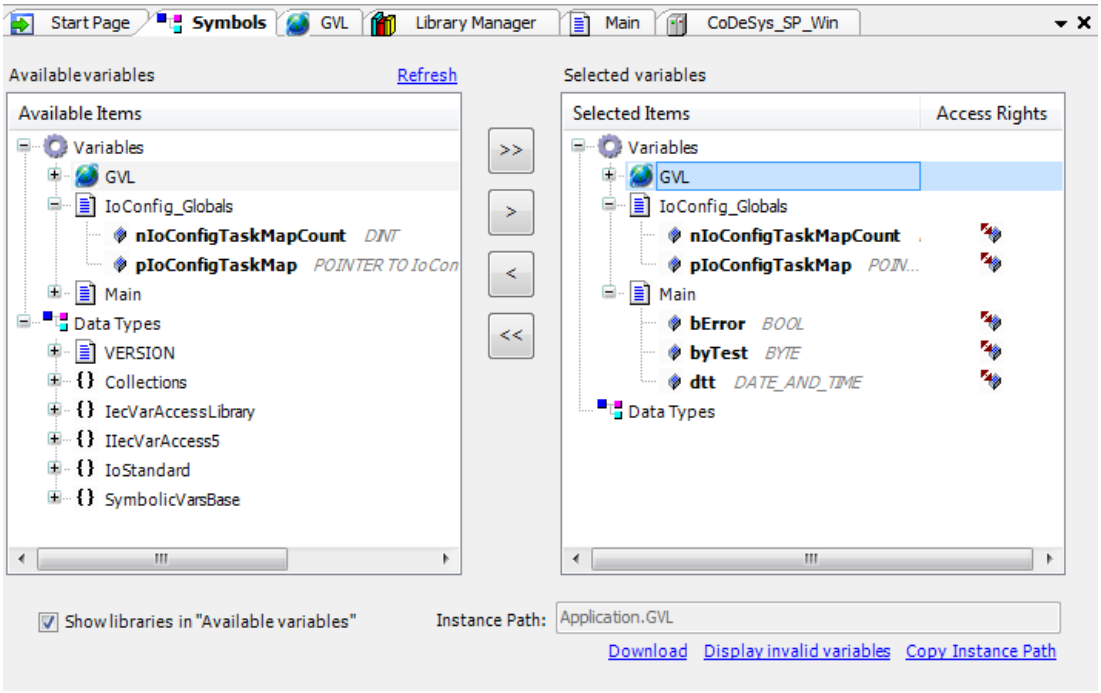


Abb. 419 Symbol Configuration Editor

In the left part of the editor window the currently **Available variables** of the application are shown in a tree structured by a "Variables" and a "DataTypes" node. Indented each below these nodes find the POU, GVL and data type names and again below finally the names, data types and access right of the particular variables (items). Via the **Refresh** button the tree can be updated with the currently valid variables. With the compiler option **Link Always** a not compiled object can be reinterpreted and downloaded to the PLC. Then all variables declared in this object will be available. In addition the pragma {attribute 'linkalways') can be used to make not compiled variables available in the symbol configuration.

In the right part of the window the currently **Selected variables** are displayed also in a tree structure. For those items each a symbol definition will be created and exported to the symbol file. The **Display invalid variables** button will indicate by a red writing all variables which are currently not valid in the application, for example because their declaration has been removed.

Use the **single-arrow buttons** in the center of the editor window to add a currently selected item (or node) from the left tree to the right tree resp. to remove one from there. The **double-arrow buttons** always will take the complete list, in this case nothing must be selected explicitly .

The access right for a selected item can be modified in the right window by a mouse-click on the symbol in the **Access Rights** column. Each mouse-click will switch the symbol within the following definitions: read+write, write-only, read-only.

The checkbox **Show libraries in "Available variables"** defines if variables of libraries are also displayed.

The **Instance Path** of a currently selected item is displayed in the same-named field in the bottom part of the dialog. It can be copied to the clipboard by the **Copy Instance Path** button, which might be useful for inserting a long path to any text edit field.

If the symbol configuration has been modified in online mode, the modified application can be downloaded via button **Download**.

13.16 Task Editor

13.16.1 Task Configuration

The Task Configuration defines one or several tasks for controlling the processing of an application program. Thus it is an essential resource object for an application and must be available in the Devices window.

In contrast to »PLC Designer« 2.3 the tree structure of the Task Configuration is not viewed and handled within the editor window, but is **integrated in the resources tree in the** Devices **window**. It is an obligate resource object for each application and can be added via the Add Object command.

At the topmost position of a task configuration tree there is the entry "Task configuration". Below there are the currently defined tasks, each represented by the task name. The POU calls of the particular tasks are not displayed in the task configuration tree.

The task tree can be edited (tasks can be added, copied, pasted or removed) by the appropriate commands usable for the devices tree. For example for adding a new task use command 'Add Object' .

The particular tasks can be configured in the **Task Editor** dialogs which additionally provide a monitoring view in online mode. The options available for a task configuration are target-specific.

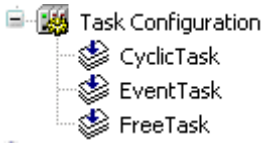


Abb. 420 Task Configuration in Devices Tree below an Application

A **Task** is a time unit in the processing of an IEC program. It is defined by a name, a priority and by a type determining which condition will trigger the start of the task. This condition can be defined by a time (cyclic, freewheeling) or by an internal or external event which will trigger the task; for example the rising edge of a global project variable or an interrupt event of the controller.

For each task you can specify a series of program POUs that will be started by the task. If the task is executed in the present cycle, these programs will be processed for the length of one cycle.

The combination of priority and condition will determine in which chronological order the tasks will be executed.

For each task you can configure a watch dog (time control); the possible settings depend on the target system.

Additionally there is the possibility to link System events (for example Start, Stop, Reset) directly with the execution of a project POU.

In online mode the task processing can be monitored.

13.16.2 Task Editor, Usage

Task Editor

The 'TaskConfig Editor' plug-in provides the editor for handling the Task Configuration, which is needed as a resource object for an application.

By a double-click on a Task configuration object in the Devices view window resp. by opening this object via command 'Edit object' the editor window will open.

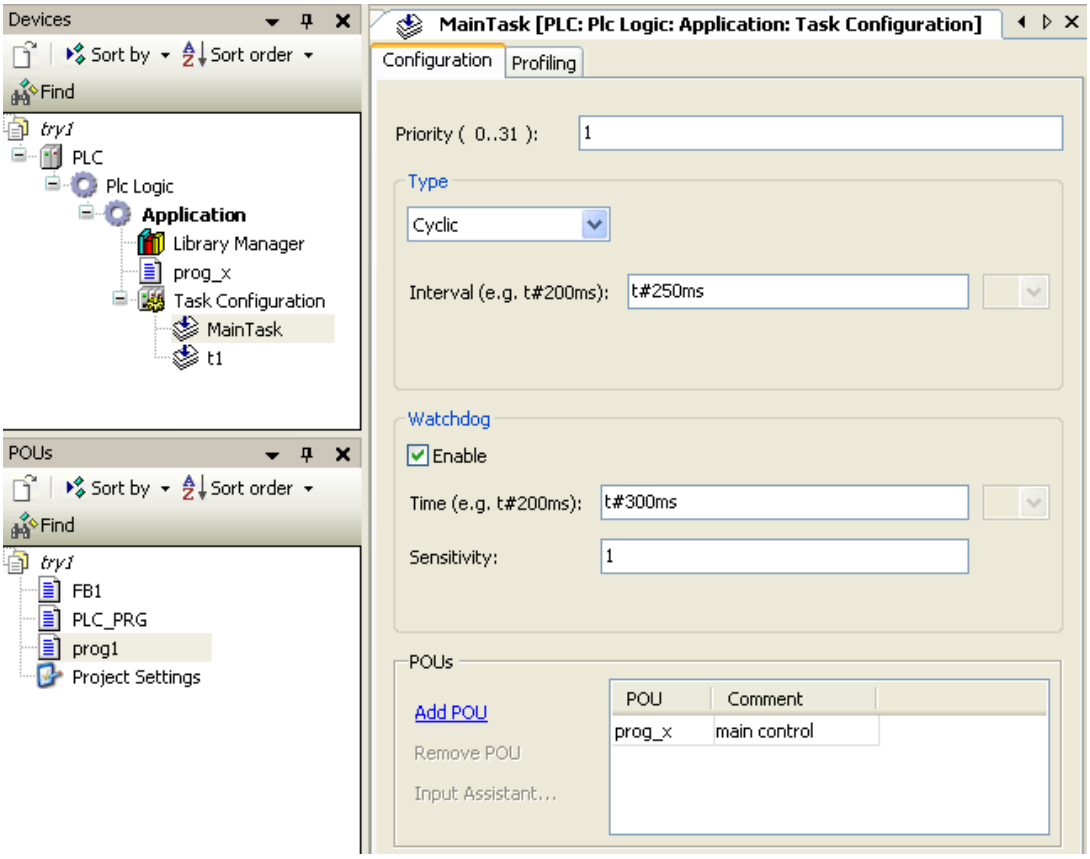


Abb. 421 Task Editor, Dialog for configuration of a task

In contrast to »PLC Designer« 2.3 the tree structure of the Task Configuration is integrated in the resources tree in the **Devices** view window. There the Task Configuration object itself as well as particular tasks can be inserted via the Add Object command. When a Task Configuration object is added, automatically also one task "Task" will be inserted and the Configuration dialog will be opened.

The task configuration resp. a particular task can be conf0igured in the following dialogs of the Task Editor, which will be opened in a tabbed window on a double-click on the respective entry in the devices tree:

- Properties dialog for information on basic task configuration settings.
- Configuration dialog for doing the configuration of a particular task, optionally extended by a Parameter dialog for specific task parameters.
- System events dialog for linking POU calls to system events.

It depends on the currently used device (target) which options are available in the configuration dialogs. NOT YET IMPLEMENTED:If the standard descriptions are extended by customer specific definitions, then those will be displayed in an additional dialog 'Parameter'.

Note: Please do not use the same string function (see standard.library) in several tasks, because this may cause program faults by overwriting.

Properties dialog

When the top entry in the task configuration tree is selected, the **Properties** dialog will be opened in the task editor window.

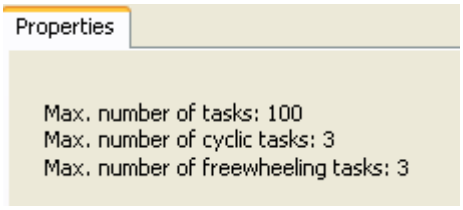


Abb. 422 Task configuration Properties dialog, example

Information on the current task configuration as provided by the target will be displayed, for example the maximum allowed numbers of tasks per task type.

Configuration dialog

When inserting a task ('Add Object' command) in the Task Configuration in the Devices view, the Task Editor dialog for setting the task properties will be opened:

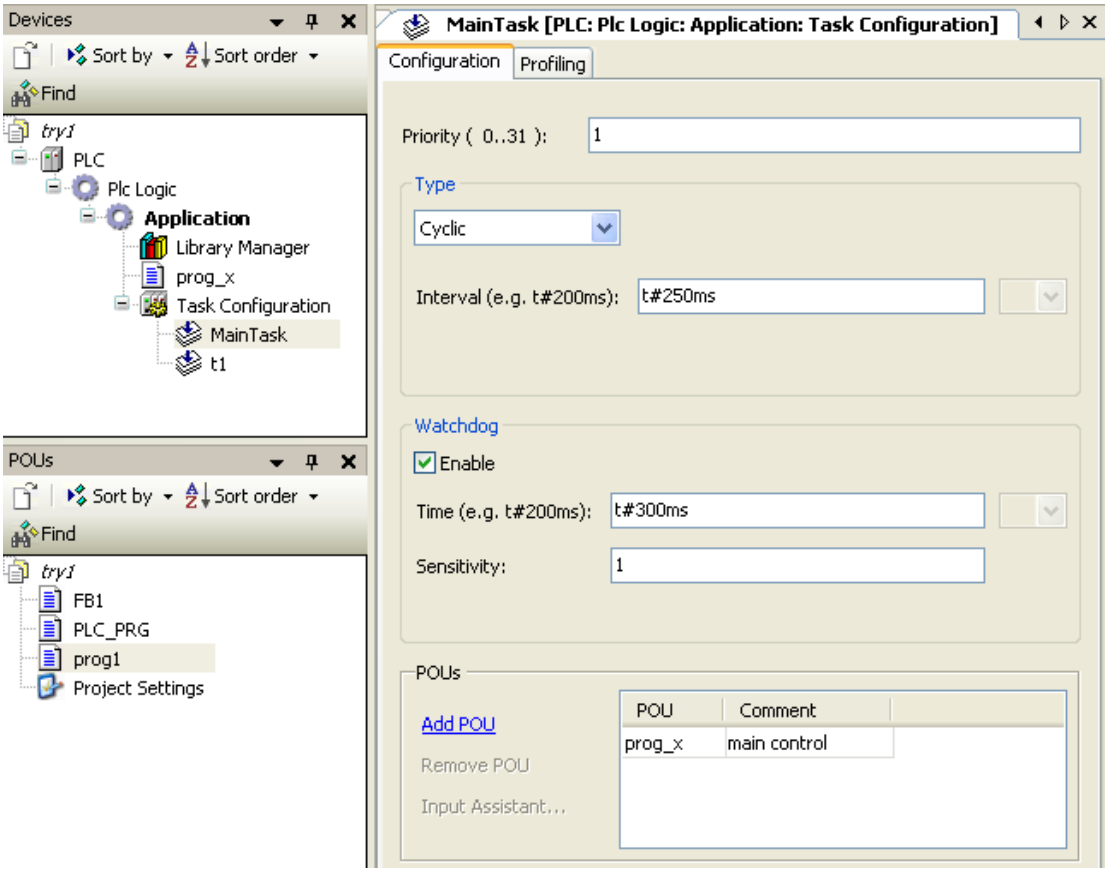


Abb. 423 Configuration dialog for a task

Insert the desired attributes:

Note: The task name can be modified by editing the respective entry in the devices tree.

Priority (0-31): (a number between 0 and 31; 0 is the highest priority, 31 is the lowest),

Type: The selection list offers the following task types:

Cyclic: The task will be processed cyclic according to the time definition given in the field 'Interval' (see below).

Freewheeling: The task will be processed as soon as the program is started and at the end of one run will be automatically restarted in a continuous loop. There is no cycle time defined.

Status: The task will be started if the variable defined in the Event field is true.

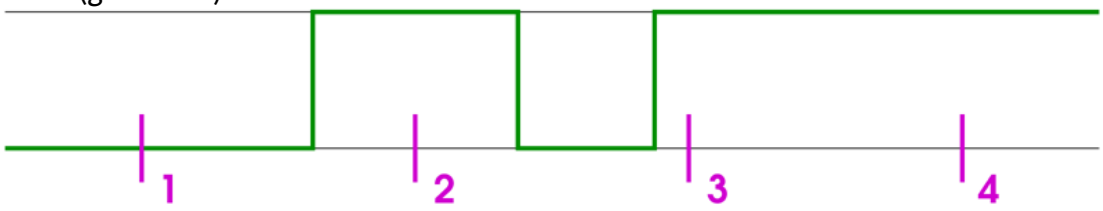
Event: The task will be started as soon as the variable defined in the Event field gets a rising edge.

External event: The task will be started as soon as the system event, which is defined in the **Event** field, occurs. It depends on the target, which events will be supported and offered in the selection list. (Not to be mixed up with system events.

Difference between Status and Event:

The specified event being TRUE fulfills the start condition of a status driven task, whereas an event driven task requires the change of the event from FALSE to TRUE. If the sampling rate of the task scheduler is too low, rising edges of the event may be left undetected.

The following example illustrates the resulting behaviour of the task in reaction to an event (green line):



At sampling points 1-4 (magenta) tasks of different types show different reaction:

Behaviour at point:	1	2	3	4
Status	no start	start	start	start
Event	no start	start	no start	no start

Obligatory Entries depending on task choice:

Interval (obligatory for Type 'cyclic' resp. 'triggered by external event' if the event requires a time entry): the period of time, after which the task should be restarted. If you enter a number, then you can choose the desired unit in the selection box behind the edit field: milliseconds [ms] or microseconds [µs]. Inputs in [ms]-format will be shown in the TIME format (for example "t#200ms") as soon as the window gets repainted; but you also can directly enter the value in TIME format. Inputs in [ms] will be displayed as a pure number (for example "300").

Event (obligatory for Type 'event' or 'triggered by external event'): a global variable which will trigger the start of the task as soon as a rising edge is detected. Use button **...** or the input assistant <F2> to get a list of all available global event variables.

Note: If the event that is driving a task stems from an entry, there must be at least one task which is not driven by events. Otherwise the I/Os will never get updated and the task will never get started.

NOT YET IMPLEMENTED: Possibly the target system defines Singleton Events. These are events, which only allow to start one single task. Whether such an event starts several tasks will be checked during compilation of the project. The check notices the data address of the event variable, not on the name. For example: If the target system defines %MX1.1 and %IB4 as Singleton-Events, using the following variables as event variables will produce two detected errors (a and b as well as c and d each have the same address).

```
VAR_GLOBAL
a AT %MX1.1: BOOL;
b AT %MX1.1: BOOL;
c AT %MB4: BOOL;
d AT %MD1: BOOL;
END_VAR
```

Watchdog

For each task a time control (watchdog) can be configured. If the target system supports an "extended" watchdog configuration, then upper and lower limits as well as a default for the watchdog time and a time definition in percent might be predefined by the device description.

Enable: When this option is activated (☑), the watchdog is enabled. This means that the task will be terminated in error status ("exception"), if the currently set watchdog **Time** (see below) gets exceeded, whereby there is taken account of the currently set **Sensitivity** (see below). If option "Update IO while in stop" is enabled in the PLC Settings dialog, the outputs will be set to the predefined default values.

The following cases are possible:

1.Contiguous time overruns; the following is true:

Sensitivity	Exception in cycle...
0, 1, 2	1
3	2
...	...
n	n-1

2. Single time overrun: Exception if the cycle time for the current cycle is greater than (Time * Sensitivity). Example: Time=t#10ms, Sensitivity=5 -> Exception as soon as the task (once) runs longer than 50ms. This serves to detect endless loops in the first cycle.

Time (e.g. t#200ms): Watchdog time; description see above: Enable. Depending on the target system the time has to be entered as percentage of the task interval. In this case the unit selection box will be greyed and show "%".

Sensitivity: Number; description see above: Enable;

Note: In the initialisation phase directly after a restart of the PLC application, single PLC cycles with an extended runtime occur. In order to ensure that the watchdog does not respond when this happens, the following parameterisation should be chosen:
Time : double the cycle time of the task to be monitored
Sensitivity : 3

Note that a watchdog may be **disabled for particular SPS cycles** by use of the functions provided by the library CmpIecTask.library; this may be useful for SPS cycles requiring more time as usual due to initialization processes.

After declaring an appropriate variable for the handle of the task (of type RTS_IEC_HANDLE),

```
hIecTask : RTS_IEC_HANDLE;
```

the disabling (and succeeding reenabling) can be handled by employing the interface functions in the following manner:

```
hIecTask := IecTaskGetCurrent(0);
IecTaskDisableWatchdog(hIecTask);
... // Code that is protected against watchdog
IecTaskEnableWatchdog(hIecTask);
```

POUs:

The POUs which are currently controlled by the task are listed here in a table with POU name and optionally a comment. Left to the table there are commands for editing:

- In order to define a new POU open the Input Assistant dialog via command **Add POU**. There choose one of the programs available in the project.
- In order to replace a program call by another one, select the entry in the table, open the Input Assistant via command **Input Assistant** and choose another program.
- In order to delete a call, select it in the table and use command **Remove**.
- Command **Open POU** opens the currently selected program in the corresponding editor.

The sequence of the listed POU calls from up to down determines the **sequence of execution** in online mode. Via the commands **Move up** and **Move down** the currently selected entry can be shifted within the list.

NOT YET IMPLEMENTED: Manufacturer specific attributes:

Additionally to these standard attributes for the currently selected task manufacturer specific attributes might be displayed in a second tab "**Parameters**". This will be the case if it is defined in the target-specific description file for the task configuration.

Parameter dialog

This dialog is only available on a tab of the Task Editor dialog if defined by the target. It provides manufacturer specific attributes of the task (Name, Type, Value, Description).

System Events dialog

NOT YET IMPLEMENTED

13.16.3 Task Editor in online mode

Which task is being processed?

For the execution of the tasks defined in the Task Configuration the following rules apply:

- That task is executed, whose condition has been met; that is, if its specified time has expired, or after its condition (event) variable exhibits a rising edge.
- If several tasks have a valid requirement, then the task with the highest priority will be executed.
- If several tasks have valid conditions and equivalent priorities, then the task that has had the longest waiting time will be executed first.
- The processing of the program calls will be done according to their order (top down) in the task editor. If a program is called which with the same name is available assigned to the application in the device tree as well as in a library or project-globally in the POUs window, that one directly assigned to the application will be executed.

Monitor, Online view of the Task Editor

When the top node in the Task Configuration tree is selected, besides the Properties dialog on a further tab the 'Monitoring' dialog is available. In online mode it shows the status and some current statistics on the cycles and cycle times are displayed in a table view. The update interval for the values is the same as used for the monitoring of PLC values.

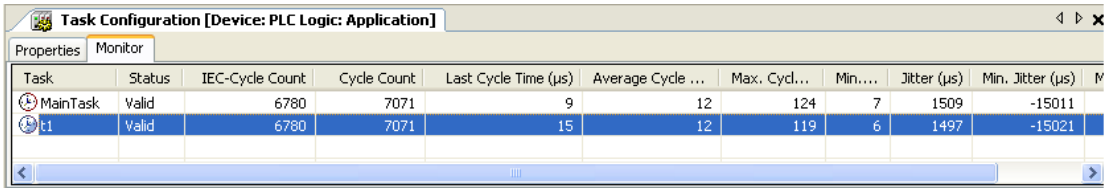


Abb. 424 Task Configuration, Monitoring

For each task the following information is displayed in a line:

Task	task name as defined in the task configuration
Status	possible entries: Not created: has not been started since last update; especially used for event tasks Created: task is known in the runtime system, but is not yet set up for operation. Valid: task is in normal operation Exception: task has got an exception
IEC-Cycle Count	number of run cycles since having started the application; "0" if the function is not supported by the target system
Cycle Count	number of already run cycles; (depending on the target system this can be equal to the IEC-Cycle Count, or bigger, if cycles are even counted when the application is not running)
Last Cycle Time (µs)	last measured runtime in µs
Average Cycle Time (µs)	average runtime of all cycles in µs
Max. Cycle Time (µs)	maximum measured runtime of all cycles in µs
Min. Cycle Time (µs)	minimum measured runtime of all cycles in µs
Jitter (µs)	last measured jitter* in µs
Min. Jitter (µs)	minimum measured jitter* in µs
Max. Jitter (µs)	maximum measured jitter* in µs

* Jitter: Time between when the task was started and when the operating system indicates that it is running.

When the cursor is placed on a task name field, the values can be reset to 0 for the respective task by the **Reset** command available in the context menu.

13.17 Trace Editor

The Trace functionality is provided with the 'Trace Editor' plug-in. It allows to record and read the progression of the values of variables on the PLC over a certain time. For this purpose the values of defined trace variables are steadily written to a »PLC Designer« buffer of a specified size and then can be viewed in the form of a graph along a time axis.

The configuration as well as the display of the trace sampling is done in the Trace Editor dialogs and views. Multiple variables can be traced and displayed at the same time, if desired in different views.

In order to start trace sampling according to the current trace configuration in online mode, you have to **download the configuration** to the runtime system. The graphs of the trace variables will be displayed in the Trace Editor window and you might store them to an external file which later can be reloaded to the editor.

Commands for accessing the configuration dialogs as well as commands for modifying the currently displayed section of the trace curves resp. graphs are available in the Editor window.

The readout of a trace may also be integrated within a visualization by use of the special visual element trace.

13.17.1 Trace Editor

See on this page:

- Overview
- Commands and zoom functionality
- Configuration of the trace
- Trace editor in online mode

Overview

The Trace Editor, provided by the 'Trace Editor' plug-in, is used to configure and to display "Traces". A "trace" in this context is a **sampling of the values of variables**, that is the online progression of the values read from the PLC over a certain time. Additionally a trigger can be set. If that had be done only the triggered values are recorded and displayed. For this purpose the values are sampled in a »PLC Designer« buffer and can be **visualized** in the Trace Editor window in the form of a graph in a coordinate system of defined appearance.

If exist the additional component 'Trace Manager' in the runtime system, then the trace configuration management and the data storage is done there. It is also possible to access device-specific traces with help of the trace manager like the record of current values. If there isn't available a trace manager, the functionality is downloaded with IEC-code. Then the menu commands **Online List...** and **Upload Trace** are not available.

Variables triggered simultaneously can be recorded in one trace object. There the variables can be displayed in a variable specific coordinate system and possibly in multi-channel-mode. Traces of variables with different trigger have to be recorded in their own trace object. You can create any number of trace objects.

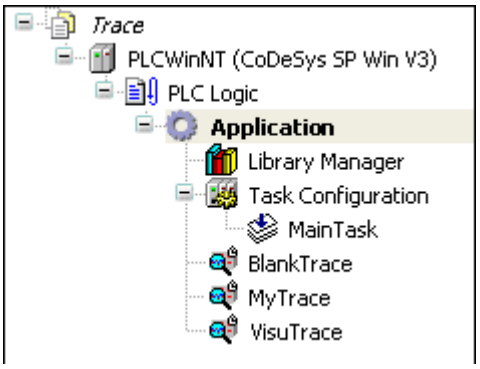


Abb. 425 Example: Project with traces

This picture shows an example of tracing in online mode. Four variables have been selected for display in the variables tree in the right part of the dialog. In the context menu of this part of the window see the commands available to add or delete variables, to set them visible, to change the appearance, or to reach the configuration dialog.

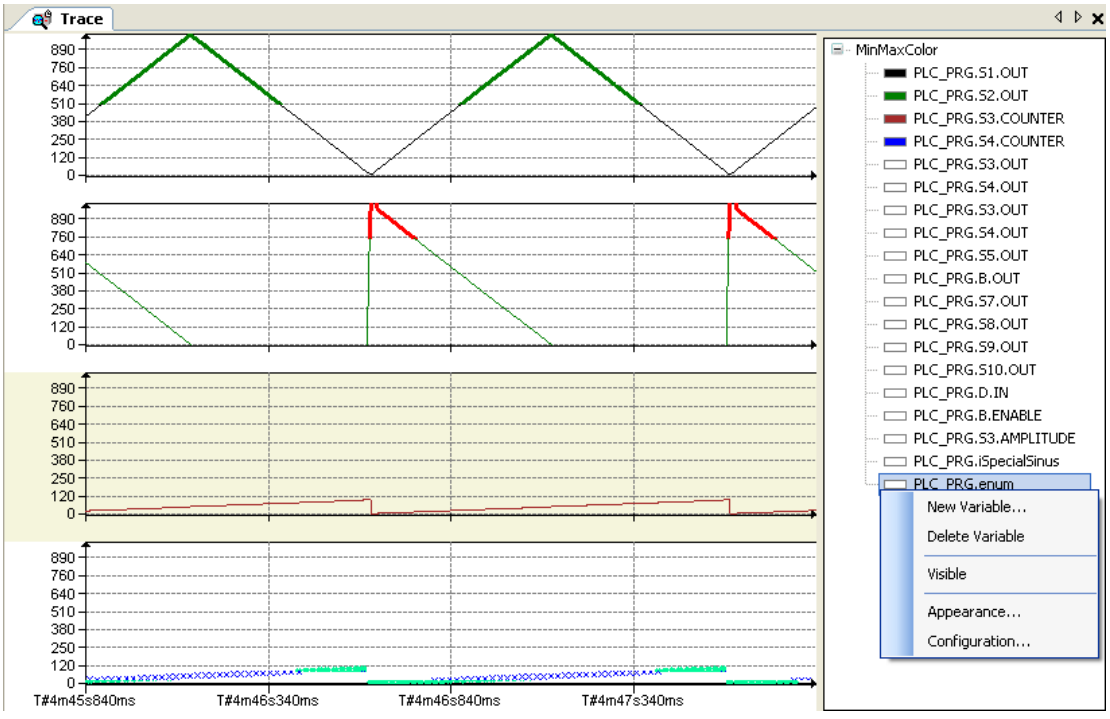


Abb. 426 Example: Trace editor window, showing one trace of four variables in online mode

Commands and zoom functionality

For "working" in the Trace Editor in offline resp. in online mode there are commands available in the Trace menu as well as in the trace window. The Trace menu provides commands for operating the trace graph. With focus on the diagram in the trace window the context menu provides the trace menu commands too. The trace configuration with trigger setting and the variable selection can only be done in the context menu of the right part of the trace window where the trace tree is visualized. Up to the selection in the trace tree different commands are enabled.

Notice that besides the zoom commands also a "scroll mouse" can be used to **zoom** the display of the coordinate system in the trace window.

Configuration of the trace

A "Trace" object can be **added** in the Devices view window by the 'Add Object' command and by 'Edit Object'. Primarily the main editor window, titled with "**<name of trace object>**", shows an empty area in the left part where later the trace graph will be displayed and the trace tree as configured in the Configuration... dialog in the right part.

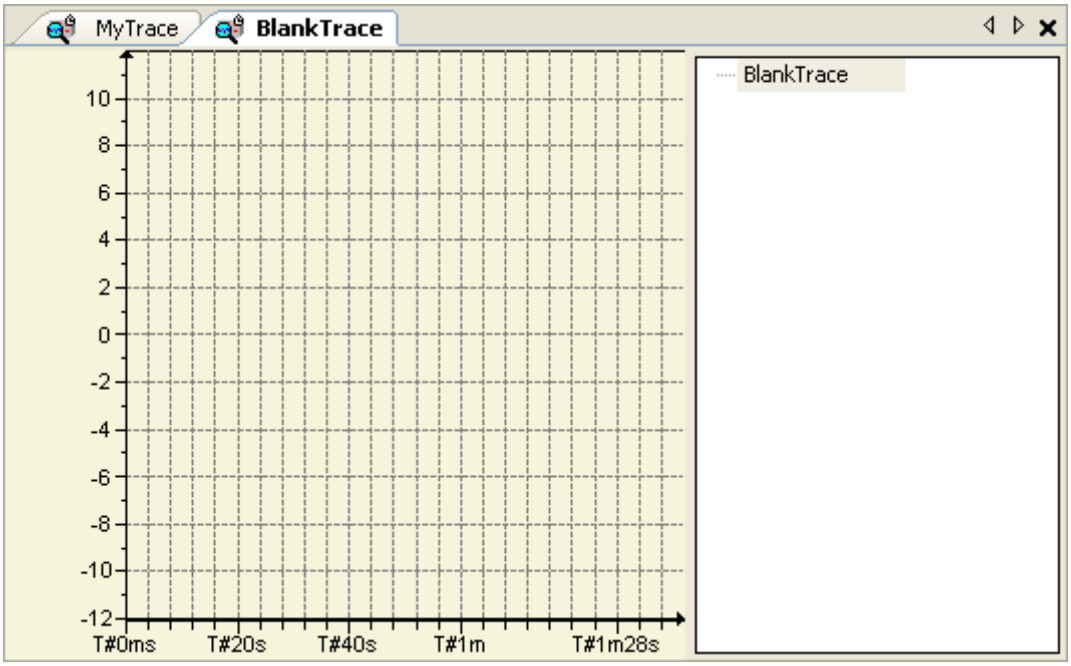


Abb. 427 Trace editor window, no configuration

A trace defines one or several variables to be traced. For this common settings can be defined concerning the start of the trace, sampling, trigger conditions, buffer size, responsible task etc.. Beside that the "appearance" (display parameters for the coordinate system where the curve will be displayed) of every trace variable has to be set.

To set up a trace use the context menu in the right part of the trace window ...

1. New Variable...: dialog to add a variable and set some display parameters (color, graph type etc.).
2. **Delete Variable:** to delete the selected variables.
3. **Visible:** to set a selected variable visible.
4. Appearance...: dialog to determine the appearance of the graph (This command is grayed as long as no configuration is loaded).
5. Configuration...: dialog for the definition of the trace (conditions, curve type etc., and once again appearance of the graphs) with its particular trace variables.

A trace configuration can be **stored to an external file *.trace** and be reloaded to the trace window.

downloaded to the target system.

Trace editor in online mode

To perform a trace record viewing the code generated for the trace must explicitly loaded to the target system with the menu command Download.

The Trace menu provides additional **commands** for working in the currently displayed trace graphs.

New Variable...

The Trace Configuration dialog with the Variable Settings dialog opens when using command **New Variable...** in the context menu of the trace tree window in the Trace Editor.

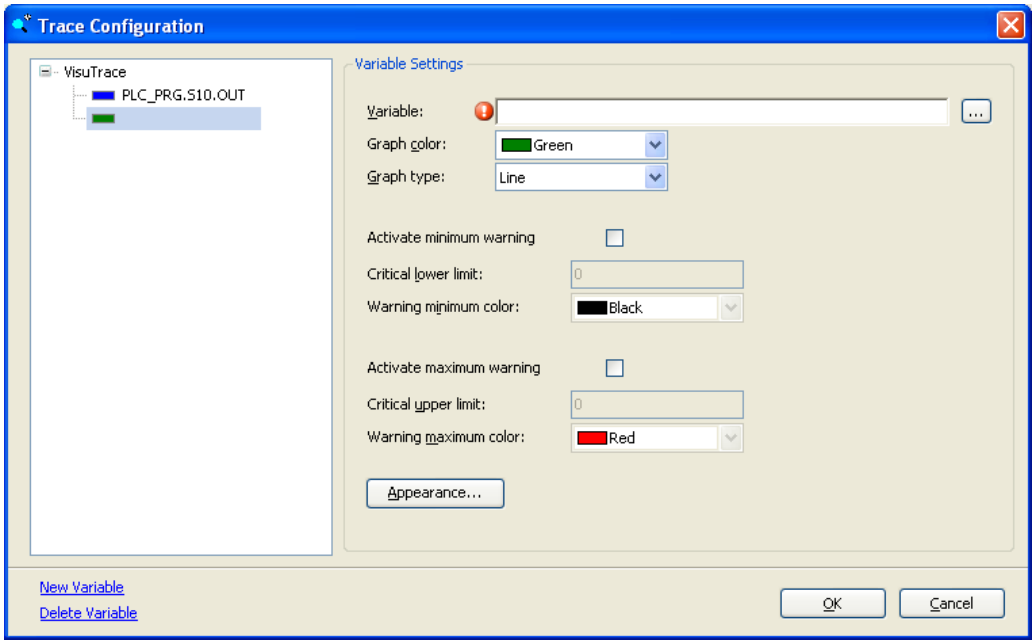


Abb. 428 Trace Configuration dialog

Variable: Enter here the name (path) of the variable to be traced. The input assistant ... can be used in order to get a valid entry. Check resp. edit the current configuration settings. The variable will be immediately entered in the record tree. To modify the variable settings later, just select the variable entry in the record tree again and use the Variable Settings dialog again.

Graph color: From the given color selection list choose a color in which the trace curve for the variable should be displayed.

Graph type: Specify here how the particular data points in the trace curve of the variable should be displayed. It is recommended to use the "Line" type for big volumes of data.

None	No graph will be displayed.
Line	The points are connected to a line.
Crosses	The points are indicated by crosses.
Spline	A spline graph created of the recorded values will be displayed.
Step	The points are connected in shape of a staircase: vertical line up to the y-value of the next point, from there horizontal line up to the x-value.
Dots	The points are indicated by dots.

Activate Minimum Warning: If this option is activated, the trace graph will be displayed in the color defined in 'Warning minimum color' as soon as the variable exceeds the value defined in 'Critical lower limit'.

Critical lower limit: See above, Activate Minimum Warning.

Warning minimum color:See above, Activate Minimum Warning.

Activate Maximum Warning: If this option is activated, the trace graph will be displayed in that color which is defined in 'Warning maximum color' as soon as the variable exceeds the value defined in 'Critical upper limit'.

Critical upper limit: See above, Activate Maximum Warning.

Warning maximum color: See above, Activate Maximum Warning.

Configuration...

See on this page:

- Record Settings
- Variable Settings

The **Trace Configuration dialog** opens on command **Configuration...** which is available in the context menu of the trace tree on the right part of the main Trace Editor window. By double klick on the trace name or the variable names the dialog opens, too. Here you configure which variables should be traced and which parameters should be set for the tracing and the defined variables.

Note: The settings done in the dialog Record Settings are valid for all variables in the trace graph.

The Trace Configuration is composed of one trace record configuration (Record Settings) and of the dedicated variable settings. The configured trace variables are visualized in the trace tree in the left part of the trace configuration dialog and is the same than the one in the main trace editor window.

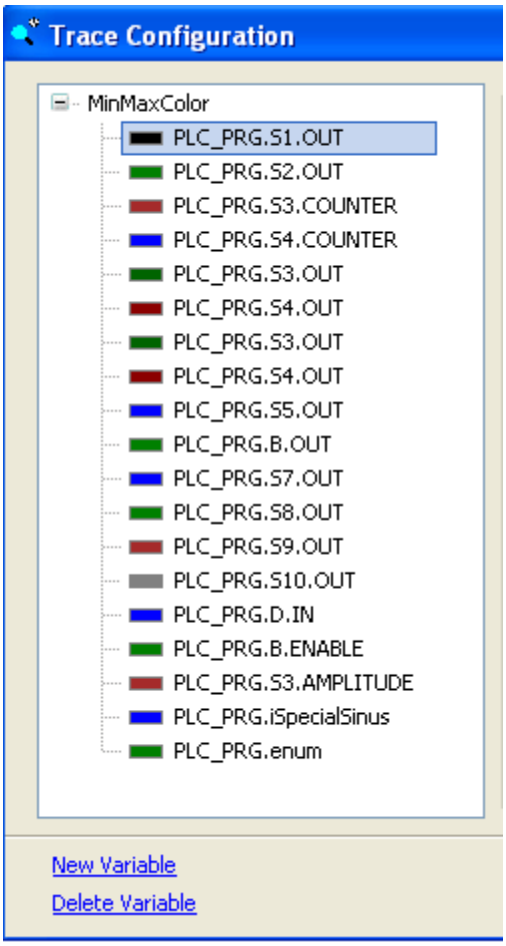


Abb. 429 Trace tree in the Trace Configuration dialog

See on top level the name of the configured trace and indented below the names of the tracing variables. For changing the record settings select the name of the trace on top level of the trace tree. Then the Record Settings dialog is opened.

For adding a variable to the trace or deleting one see the commands below the record tree: **New Variable**, **Delete Variable**.

For changing the variable settings select the name of the desired variable, then the **Variable Settings** dialog is opened. Also it is possible to select several items with **Shift + mouse click** or **Ctrl + mouse click** of the variable list. Then the changing in the Variable Settings dialog is effected on all of the selected items.

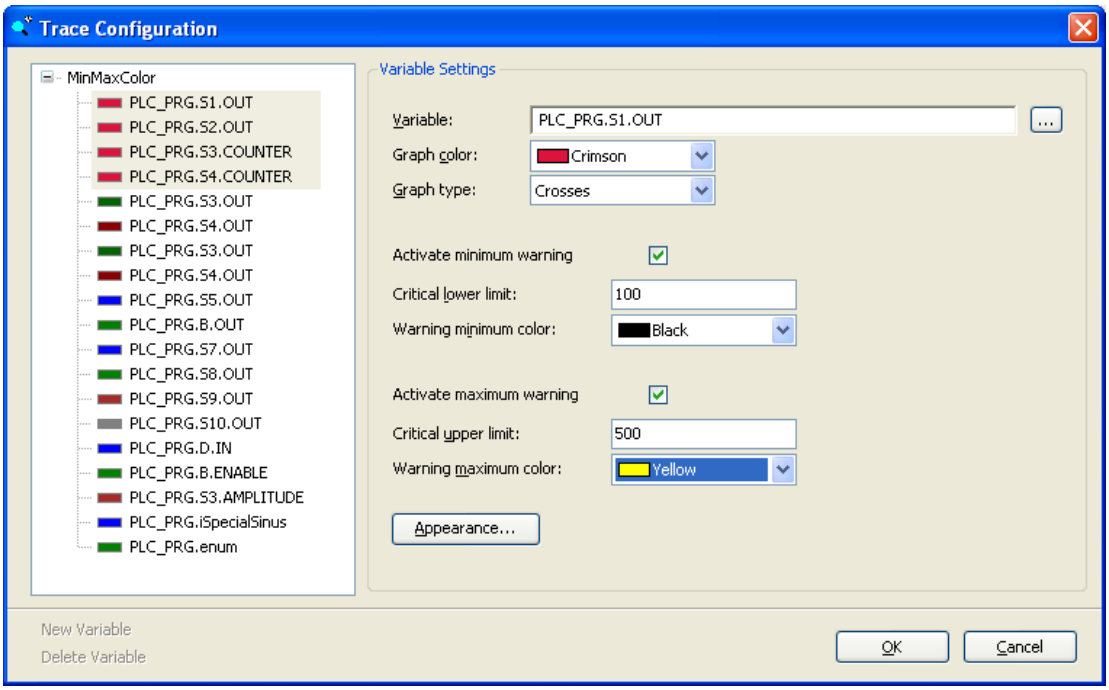


Abb. 430 Multi selection of variables

Record Settings

See in the following a description of the trace configuration dialog for record settings.

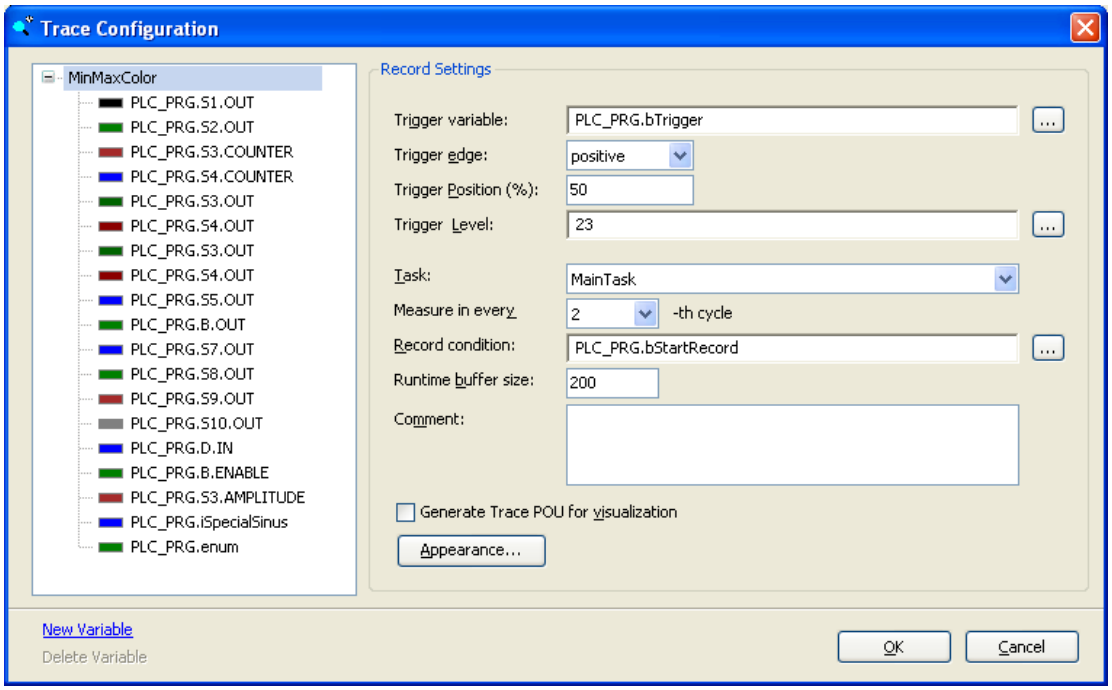


Abb. 431 Trace Configuration, Record Settings

Trigger variable: Optionally a trigger can be configured, which should determine the time span of tracing according to certain conditions. The following settings have to be made for this purpose:

Enter a boolean variable, an expression or an analogue variable. Also enumeration variables or property variables can be entered. When this variable has reached a defined value according to the type of 'Trigger edge' (see below), the tracing will be terminated after having additionally sampled that percentage of values which is defined in the 'Position' field (see below). This means that as soon as the trigger gets TRUE or reaches a certain value, the trace logging will be continued for a defined number of cycles.

By button ... the input assistant can be used to get an appropriate trigger variable.

Notice the possibility of controlling additionally the start of the tracing by the 'Record condition' (see below).

Trigger edge:

none	no trigger effect at all (Default)
positive	trigger event on rising edge of the boolean trigger variable or as soon as the value defined by 'Level' for an analogue trigger variable is reached by an ascending run
negative	trigger event on falling edge of the boolean trigger variable or as soon as the value defined by 'Level' for an analogue trigger variable is reached by an descending run
both	trigger event on the conditions described for 'positive' and 'negative' (see above)

Trigger Position (%): Enter here the percentage of the measured values for the trace variable to be recorded **after** the trigger event has occurred. So you can define, which percentage of the measured values of the trace variable should be recorded before and which after the trigger event. Example: If you enter "25", then in the trace curve 75 % of the measured values will be displayed before the triggering event and 25% after that. If you want to start the tracing exactly with the trigger event, enter 100%.

Trigger Level: If an analogue variable is used as trigger variable, define here the value of this variable at which the trigger event will be caused. Enter a value directly or a variable defining the value (also ENUM constants are allowed). Default: empty.

Task: From the list of available tasks select that one after the execution of which the trace variable should be read. Default: The first task in the Task Configuration tree.

Measure in every x-th cycle: Define the number of cycles which should be left out before the variable should be read again. By (Buffer Size * Measure in every x-th cycle * Task interval) the minimum guarded time span can be estimated.

Record condition: A boolean variable, a value or a boolean expression can be entered here. As soon as this condition gets TRUE, the trace sampling will be started. If nothing is entered here, the trace recording will start as soon as the Trace Configuration has been downloaded and the application is running.

Buffer size: Size of the buffer in Bytes which is allocated in the runtime system for the trace variables values of the current record. The content of this buffer will be displayed during trace monitoring. Default: 100.

Comment: A comment text might be entered here concerning the current record.

Generate Trace POU for visualization: Activating the associated checkbox induces the implicit build of the component <Tracename>_<Taskname>_VISU. Use this option, if you want to integrate the trace readout within a visualization.

Appearance: This button opens the **Edit Appearance** dialog, where you can set up the display of the trace window for the currently configured record (axes, colors and scroll behaviour).

Note: If you want to display the trace curve of a trace variable in a different appearances at a time, you must assign this variable to a further trace object having different appearances.

The following settings are done for the X axis and the Y axis of the trace graph and they are visible when the trace diagram is displayed in single channel view.

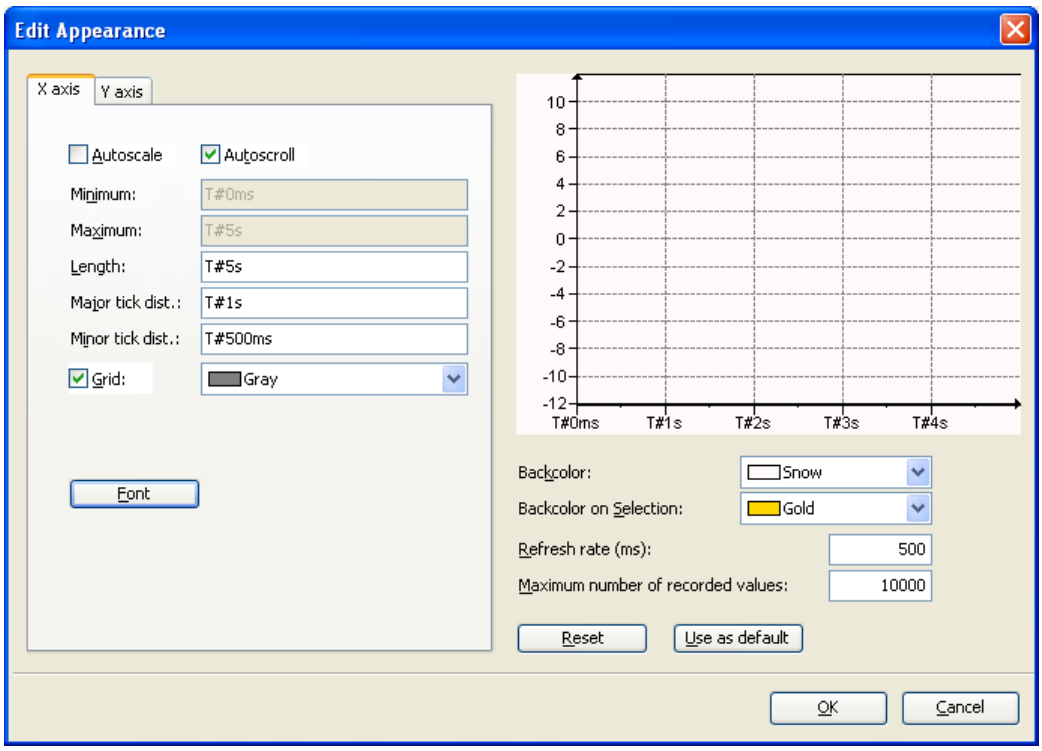


Abb. 432 Edit Appearance dialog

Autoscale: If this option is activated, the time span for tracing, that is the time span displayed on the x-axis of the trace window, will automatically be scaled according to the buffer size currently defined for the record in the runtime system.

If the option is not activated, the time span which is displayed in the trace window, can be defined by the following settings:

If option **Autoscroll** is not activated:

Minimum , **Maximum** : The times entered here (standard time format, for example "T#200s") define exactly the time span of the tracing to be displayed. This means, that the x-axis starts with the Minimum time and ends with the Maximum time. If another time span should be displayed, you have to scroll manually.

If option **Autoscroll** is activated (only possible, if option 'Autoscale' is not activated):

Length: The time span given here (standard time format, for example "T#2m") will be displayed, whereby the axis will be scrolled in a way that automatically the most recent variable values will be visible.

Major tick dist.: Value intervals indicated on the axis by long ticks. If Grid is activated they will be elongated by dotted lines. The entry must match the value type represented by the axis and must follow the IEC syntax; for example a time value for the x-axis: for example "T#2s" for 2 seconds.

Minor tick dist.: Value intervals indicated on the axis by short ticks (not elongated by a vertical dotted line in case Grid is active). Valid entries see 'Major tick dist.".

Grid: If this option is activated a grid will be displayed. The major ticks will be elongated by dotted lines for this purpose. The color of the grid lines can be chosen from the color selection list.

Description: If this option is activated, the text entered in the edit field will be displayed at the upper end of the y-axis. For the x-axis no description will be displayed here.

Font: This button opens the standard dialog for defining the font for the trace display.

Backcolor: From the given color selection list choose the background color for the coordinate system which should be used as long as this is not selected in the trace window.

Backcolor on Selection: From the given color selection list choose the background color for the coordinate system which should be used as long as this is selected in the trace window.

Refresh Rate (ms): Time interval in milliseconds, in which the update of the trace window in online mode will be done. Default: 500ms.

Maximum number or recorded values: Maximum number of values which can be written to the »PLC Designer« trace ring buffer. Default: 10000. When the buffer is full, the contained values will be overwritten starting at the "oldest" ones.

Reset: With this command the appearance is set to its default value.

Use as default: With this command the current appearance is set as default. They are used when a new trace or variable is configured.

Variable Settings

See in the following a description of the trace configuration dialog for variable settings.

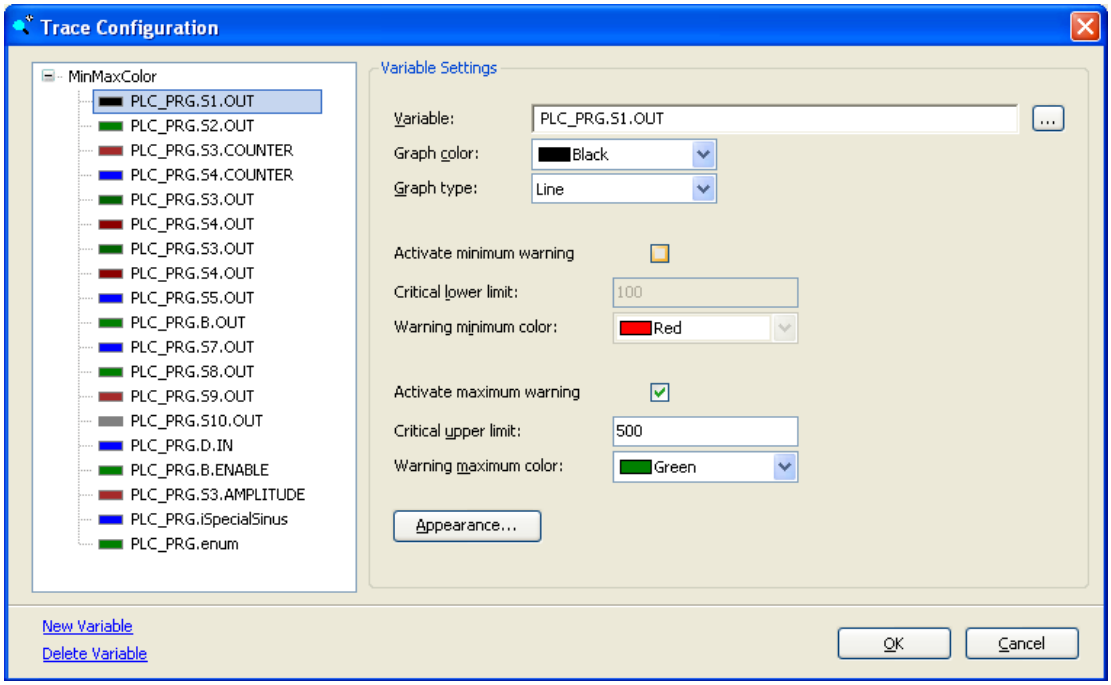


Abb. 433 Trace Configuration, Variable Settings

For the trace record you must define the variables to be "traced" and to be displayed in the record coordinate system. In order to add a variable to the record use button **New Variable** (bottom part of the window) in the left part of the Trace Configuration window. In the right part of the window now the Variable Settings dialog will appear and at the same time an empty variable entry will be added in the record tree. Define the variable:

Variable: Enter here the name (path) of the variable to be traced. Beside of data variables it is possible to trace the content of pointers, properties, references or array elements. The input assistant (...) can be used in order to get a valid entry. Check resp. edit the current configuration settings (see below). The variable will be immediately entered in the trace tree. To modify the variable settings later, just select the variable entry in the trace tree and use the Variable Settings dialog again.

Graph color: From the given color selection list choose a color in which the trace curve for the variable should be displayed.

Graph type: Specify here how the particular data points in the trace curve of the variable should be displayed. It is recommended to use the "Line" type for big volumes of data.

None:	No graph will be displayed.
Line:	The points are connected to a line.
Crosses:	The points are indicated by crosses.
Spline:	A spline graph created of the recorded values will be displayed.
Step:	The points are connected in shape of a staircase: vertical line up to the y-value of the next point, from there horizontal line up to the x-value.
Dots:	The points are indicated by dots.

Activate Minimum Warning: If this option is activated, the trace graph will be displayed in the color defined in 'Warning minimum color' as soon as the variable exceeds the value defined in 'Critical lower limit'.

Critical lower limit: See above, Activate Minimum Warning.

Warning minimum color:See above, Activate Minimum Warning.

Activate Maximum Warning: If this option is activated, the trace graph will be displayed in that color which is defined in 'Warning maximum color' as soon as the variable exceeds the value defined in 'Critical upper limit'.

Critical upper limit: See above, Activate Maximum Warning.

Warning maximum color: See above, Activate Maximum Warning.

Appearance: This button opens the **Appearance of the Y-axis** dialog, where you can set up the display of the trace window for the currently configured Y-axis (colors and scroll behaviour) for every variable in its own style. This settings are used when the trace diagram is displayed in **multi channel view**.

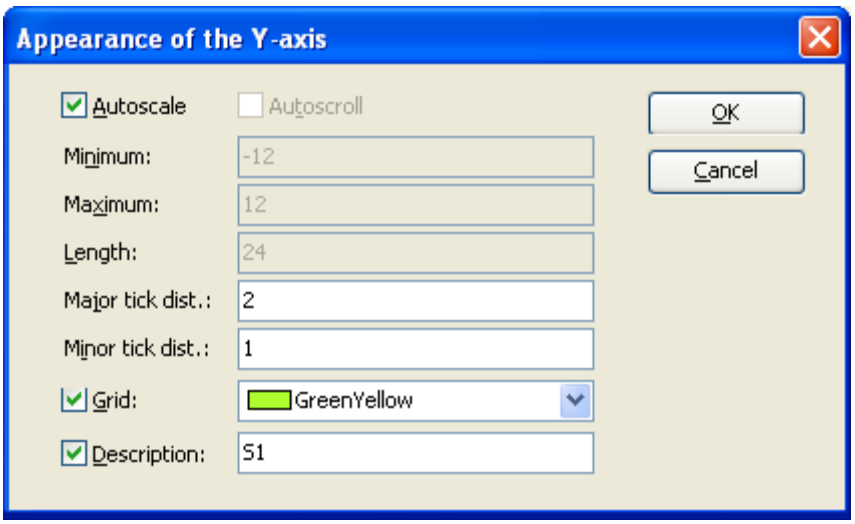


Abb. 434 Edit Appearance of the Y-axis dialog

Autoscale: If this option is activated, then the trace graph is visualized form the minimum Y-value to the maximum Y-value automatically.

If the option is not activated, then the range, which is displayed in the trace window, can be defined by the following settings:

If option **Autoscroll** is not activated:

Minimum, **Maximum**: The Y-values entered here define exactly the displaying range of the trace. This means, that the range of the Y-values starts with the Minimum value and ends with the Maximum value. If another range should be displayed, you have to scroll manually.

If option **Autoscroll** is activated (only possible, if option 'Autoscale' is not activated):

Length: The value given here will define the length of the displaying range, whereby the axis will be scrolled in a way that automatically the highest Y-values will be visible.

Major tick dist.: Value intervals indicated on the axis by long ticks. If Grid is activated they will be elongated by dotted lines. The entry must match the value type represented by the axis and must follow the IEC syntax; for example a time value for the x-axis: for example "T#2s" for 2 seconds.

Minor tick dist.: Value intervals indicated on the axis by short ticks (not elongated by a vertical dotted line in case Grid is active). Valid entries see 'Major tick dist.".

Grid: If this option is activated a grid will be displayed. The major ticks will be elongated by dotted lines for this purpose. The color of the grid lines can be chosen from the color selection list.

Description: If this option is activated, the text entered in the edit field will be displayed at the upper end of the y-axis. For the x-axis no description will be displayed.

Appearance...

You find this command in the context menu of the trace tree in the right part of the main window of the Trace-Editor. It opens the **Edit Appearance dialog** in the Trace Editor. The dialog is available, when the name of the trace on top level of the trace tree is selected. If a variable in the trace tree is selected, the **Appearance of the Y-axis dialog** opens.

The dialog is also available, when you are in the Configuration dialog of the trace.

In both dialogs you define how the trace data will be displayed in the trace diagram. That is you define the looking of the coordinate system (axes configuration, colors, grid etc.) in which the values' graphs of the variables will be displayed. The appearance settings of the record (**Edit Appearance dialog**) are used in the default single channel mode, but the appearance settings of the variables (**Appearance of the Y-axis dialog**) are visible in the multi channel mode.

Trace Editor in online mode

The "Trace" is running like an application on the current device. You get indicated an activated Trace sampling by the green color of the Trace object in the devices tree.

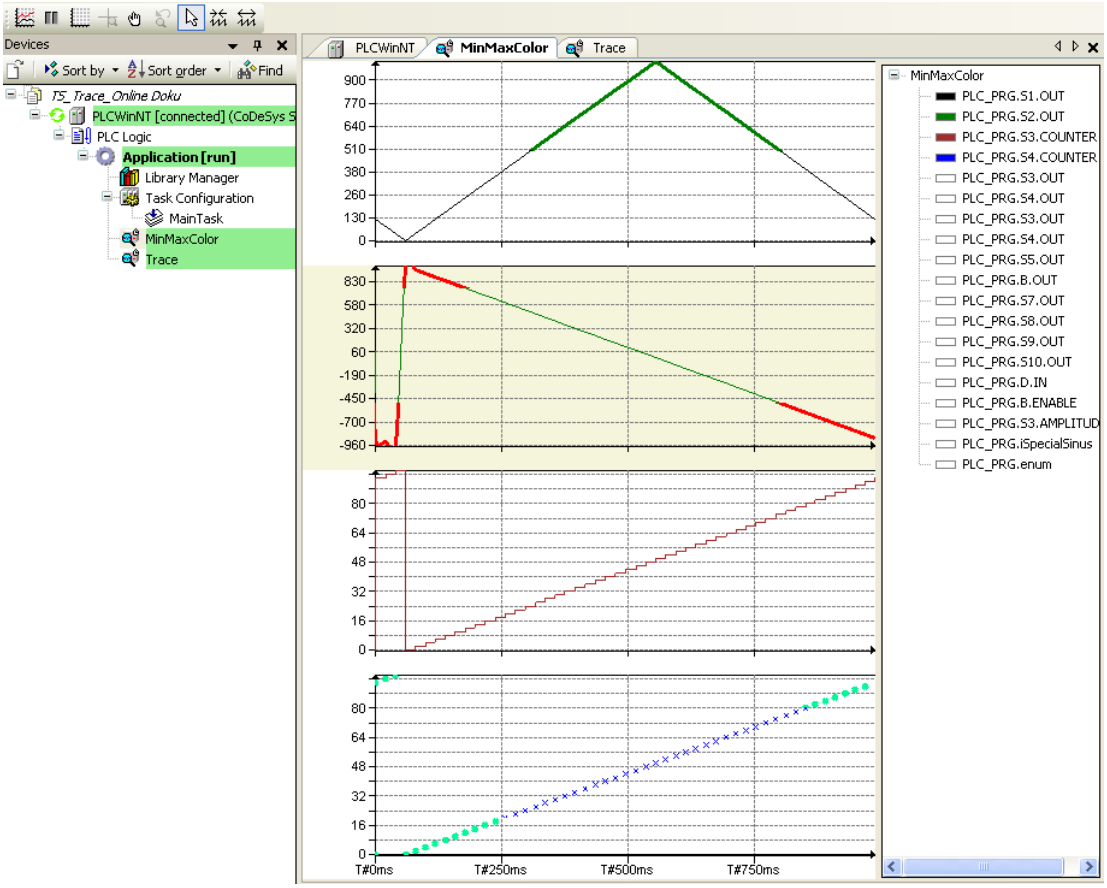


Abb. 435 Active Trace in Devices tree

Downloaded trace

For the first use and later after any changes in the trace configuration **you have explicitly to** download **the Trace "application"** in order to activate the tracing for the currently running application. After having changed the application the trace is terminated automatically and you have to re-download it. After having logged out and logged in from/to the application without having made any changes, the trace will continue without a re-download.

Configuration of trace graphs

The Configuration... dialog for the variables is also available in online mode and changes can be done online. You can double-click the name of the variable in the configuration tree in the right part of the trace editor or you can open the context menu there and select the configuration command.

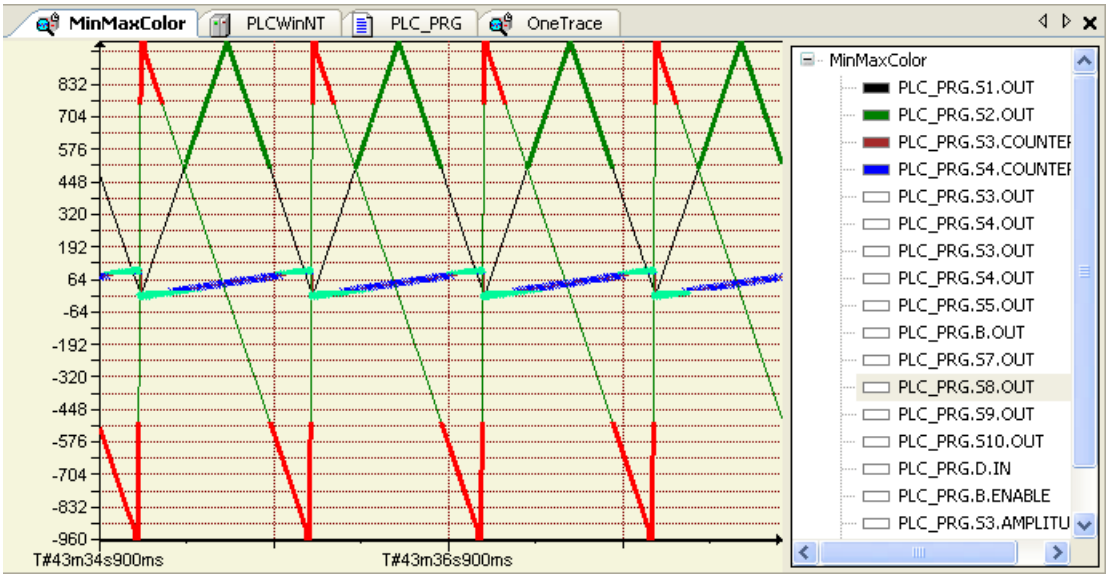


Abb. 436 Trace window in online mode, Example in single channel mode

Control the trace display

By default the commands of the Trace menu are available for controlling the currently viewed trace graph (**Start/Stop Trace**, **Reset Trigger**) resp. for modifying the currently displayed section of the trace curves (**Mouse Scrolling**, **Mouse Zooming**, etc.).

The value range of the trace graphs depends on the specific settings for the trace. However you can manipulate it manually by the scroll and zoom functions available in the Trace menu. This can be done with shortcuts, too. For vertical scrolling of the current view of the trace along the y-axis use the **Up arrow key** to move up and the **Down arrow key** to move down. For horizontal scrolling along the x-Axis use the **Left or right arrow key** while pressing the **Alt key**.

If the Multi Channel option is enabled, every graph has its own diagram and a selected graph can be scrolled as the only one alone with the shortcuts above. To scroll all the graphs simultaneously use the **Arrow keys** with an additional pressed **Ctrl key** to move the y-axis.

All the trace shortcuts listed in Shortcuts for trace.

Notice in this context also the zoom functionality of a "scroll mouse".

When the cursor is placed on the trace window the respective value of the x-axis is being showed in the status line.

Trace child application

If exist the additional component 'Trace Manager' in the runtime system, then the trace configuration is downloaded in its own child application. Then it is listed in the applications dialog of the PLC device editor and it is possible to scan and delete the child application there.

Syntax of a trace child application

<application name>.__<trace name>

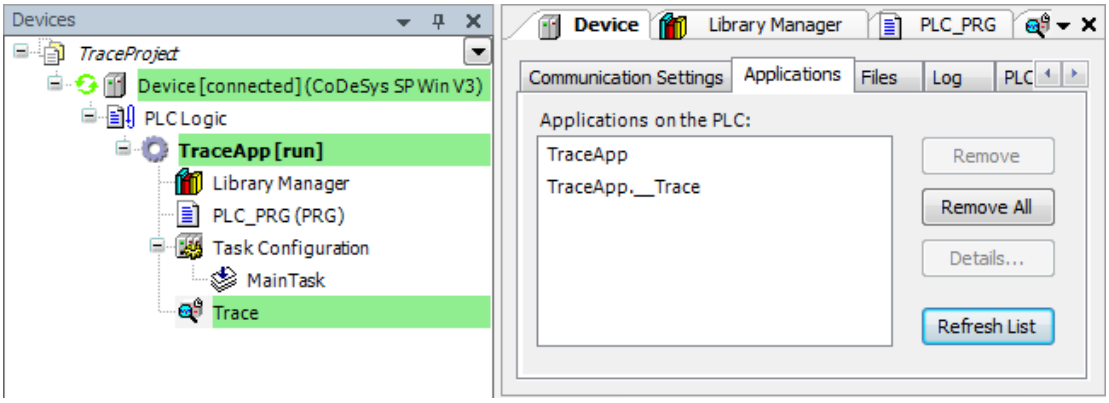


Abb. 437 Device editor, Application tab: trace application

13.18 Visualization Editor

The visualization editor - used to program a visualization object - is available via the 'Visualization Editor' plug-in. See Visualization for the preconditions and an overview on the Visualization in »PLC Designer« V3.x.

The visualization editor will be opened if you are going to edit a visualization object, which has been added to the POUs or Devices window by the 'Add Object' command.

The editor window will be titled " <visualization name>[<object path>]".

There are the following additional tools and editors, to be used in combination with the Visualization editor for creating a visualization:

1. The visualization ToolBox which contains all currently available visualization elements.
2. The Visual Element Properties editor, containing the properties of the visualization element which is currently selected in the editor window.

If at least one of the following editors is activated via the respective command (per default in the Visualization menu), the Visualization Editor will get bipartite and provides the respective tabs in the upper part:

1. The Interface Editor for defining placeholders. If the visualization is intended to be inserted in another visualization.
2. The Hotkeys Configuration Editor for assigning actions to keys resp. key combinations. Regard however that the device using the visualization must support the respective keys.
3. The Element List Editor, providing an overview on all elements of the current visualization and allowing selection, deletion and changing element position from back to front and vice versa.

Please note: For working with visualizations in the programming system the current settings in the Options dialog, category Visualization are valid. For example a grid can be applied to the editor window.

Note: In visualizations associated to a device there might be device-depending restrictions concerning the configuration possibilities (for example: list of available elements, fonts, colors, image formats, max. number of visualization elements in a visualization, max. number of visualizations below the device).

As soon as an element has been inserted in the visualization editor, it can be selected and modified concerning position and size by drag&drop mouse-actions and concerning alignment and order via the respective commands provided with the Visualization Editor.

Further configuration options depend on the type of the element. All element attributes are displayed in the Visual Element Properties view and can be edited there.

13.18.1 Inserting Visualization Elements, Toolbox

The **ToolBox**, which is to be used in combination with the Visualization Editor, provides all currently available visualization elements. All visualization elements are provided via libraries and the current selection depends on the currently active visualization profile and in case of device-associated visualizations by the device description.

The ToolBox window can be opened via the 'View' menu. It contains the three categories Visual, Complex Controls and Windows Controls, wherein the associated visualization elements are listed with names and icons. Via **drag & drop** the elements can be inserted in the currently opened visualization editor window.

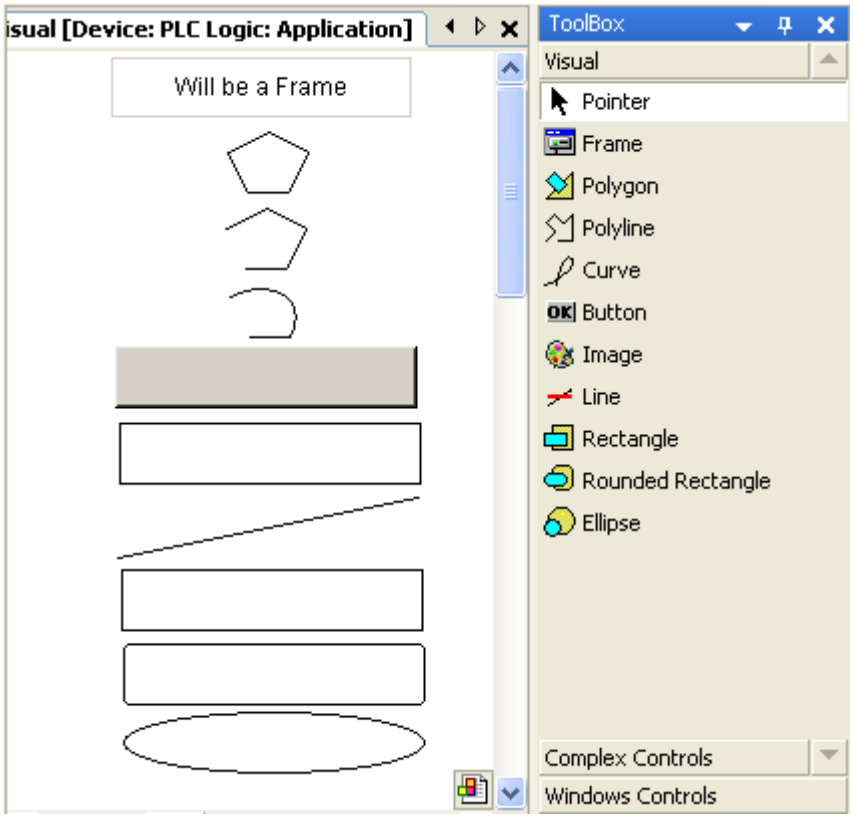


Abb. 438 Example -Toolbox category "Visual" and inserted elements

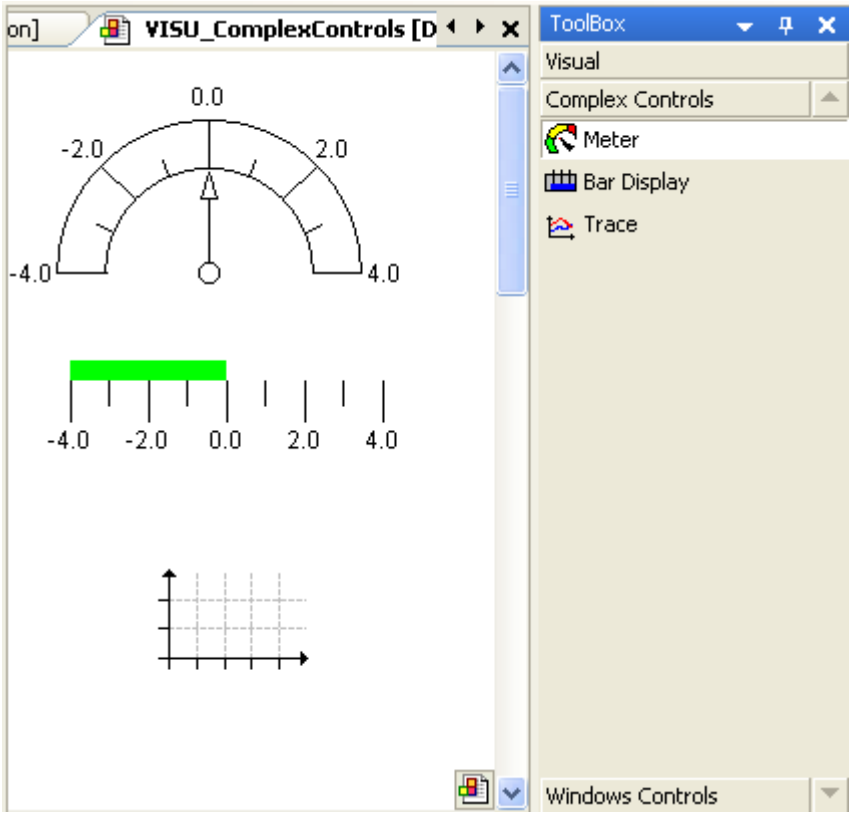


Abb. 439 Example -Toolbox category "Complex Controls" and inserted elements.

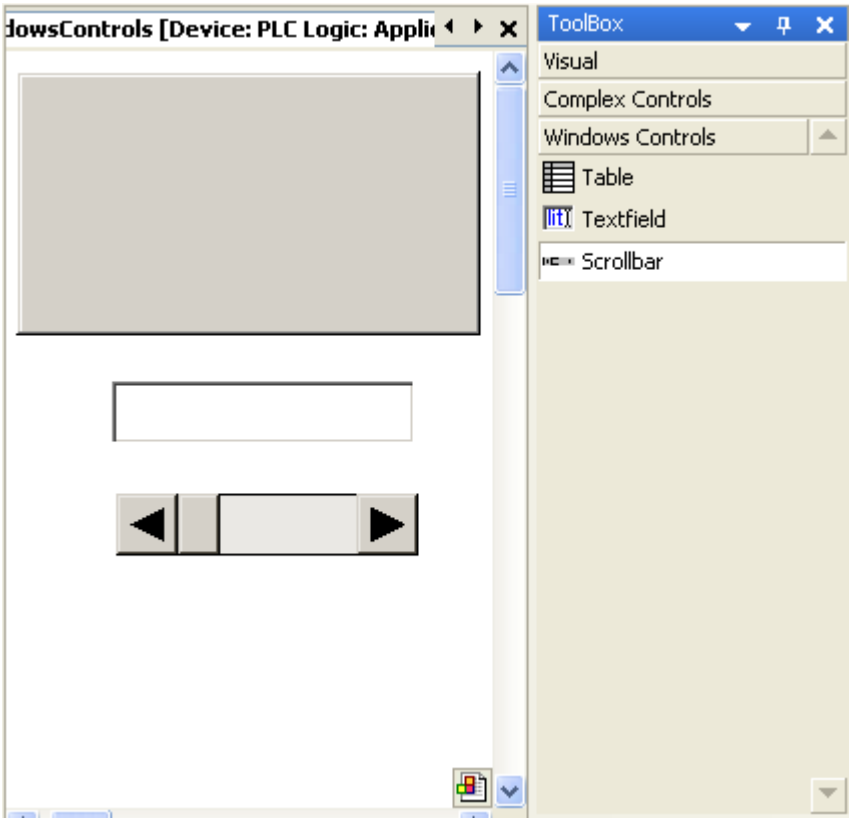


Abb. 440 Example -Toolbox category "Windows Controls" and inserted elements.

A plus-sign at the cursor during dragging indicates that the element will be brought into the editor window. After having been dropped, the element will be displayed in the visualization.

Another way to insert an element is via command Add Visualelement, by default in the 'Visualization' menu, which leads to a submenu providing the available elements like the toolbox does. If necessary use the Customization dialog to configure menus and commands regarding command category 'Visual Commands'.

For inserting other visualizations in a frame element of the current visualization use the Frame Selection command, by default also available in the Visualization menu.

After having inserted an element you can select it and change position, size, order and alignment directly within the visualization editor resp. configure further parameters in the element properties window.

Notice the standard library for visualization elements: VisuElems.library.

13.18.2 Selecting elements

A selected element is provided with small squares on the outline indicating the points which then can be selected via a further mouse-click to change position or size.

To select a visualization element move the mouse on an element. As soon as the cursor symbol converts to a hand-symbol, perform a mouse-click.

Multiple selection is possible by keeping the <Shift>-key pressed when selecting the particular elements or by drawing a rectangle around multiple elements with the left mouse-button. Also notice the respective "Visual commands" 'Select all' and 'Select none'.

Alternatively elements can be selected in the Element List. There also single subelements of a group can be selected.

Keyboard Usage:

When an element is selected, the selection can be shifted to the next element according to the insertion order (z-order) via the <Tabulator> key, via <Shift> + <Tabulator> will shift to the previous one.

By using the <Space> bar when an element is selected, a text input field gets opened. After having entered a string, that will be applied to the element (element property "text") as soon as having closed the edit field via the <Enter> key.

The properties of the currently selected element immediately will be displayed in the Visualization Element Properties window and can be edited there. If multiple elements are currently selected, the property modifications will be applied to all.

13.18.3 Position, Size, Alignment, Order

Changing the position of a visualization element:

Position

Either select the element and immediately move it - the mouse-button keeping pressed - to the desired position or instead of the mouse use the arrow keys. Or click again on an already selected element and move it by keeping the mouse-button pressed or by using the arrow keys. Another possibility is to edit the "Position" values in the Visual Element Properties window.

Size

Select the element and then perform a mouse-click on one of the little squares on the elements outline. The actual cursor symbol, a double-arrow in vertical, horizontal or diagonal orientation will indicate in which direction the square and thereby the respective parts of the outline can be moved to change the elements size. Another possibility is to edit the "Position" values in the Visual Element Properties window.

Order

To define where an element should be positioned on the Z-axis of the visualization area, that is in back- or foreground or somewhere in between, use the commands by default available in the Visualization/Order submenu.

Alignment

To align two or several elements use the commands which are by default in the Visualization/Alignment submenu.

By default commands for the Visualization are provided in the "Visualization" menu. If necessary open the Customize dialog for the configuration of the menus and commands and there notice command category Visual Commands.

13.18.4 Grouping elements

Multiple elements can be selected and grouped. Concerning this please see the description of the visualization commands Group and Ungroup. Subelements of a group can only be selected via the Element List.

13.18.5 Visualization Element Properties

When the visualization editor is opened, you can insert visualization elements into the current visualization by dragging from the toolbox.

As soon as an element is inserted in the current visualization, it can be changed concerning position, size, order and alignment via mouse-actions in the editor view and/or via the commands of category Visual Commands (Visualization menu by default).

Further on it can be **configured** concerning its animation behaviour, see in the following:

Element Properties Editor

Basically the properties of an element get configured in the Properties editor. Exception: The **position** parameters can also be changed directly via moving the element by mouse and via menu commands, and the **text** input can also be done "inline".

The properties editor can be opened via command 'Element Properties' (per default in the 'View' menu). It shows the properties of the currently selected element according to the filter currently set via the editors' Filter menu (see below):

A **Property** can be modified by editing the **Value** field. For this purpose - depending on the property type - you can open an edit frame, a selection list or a dialog by a double-click on the selected field.

In case of properties of category Inputs / OnDialogClosed or Inputs / OnMouse-actions the current configuration is shown in one or several lines below the "Configure..." value field. To modify these particular sub-configurations, click on the respective line to get a simplified configuration dialog. To modify or delete the complete mouse-action configuration, click on the "Configure..." field to get the original configuration dialog.

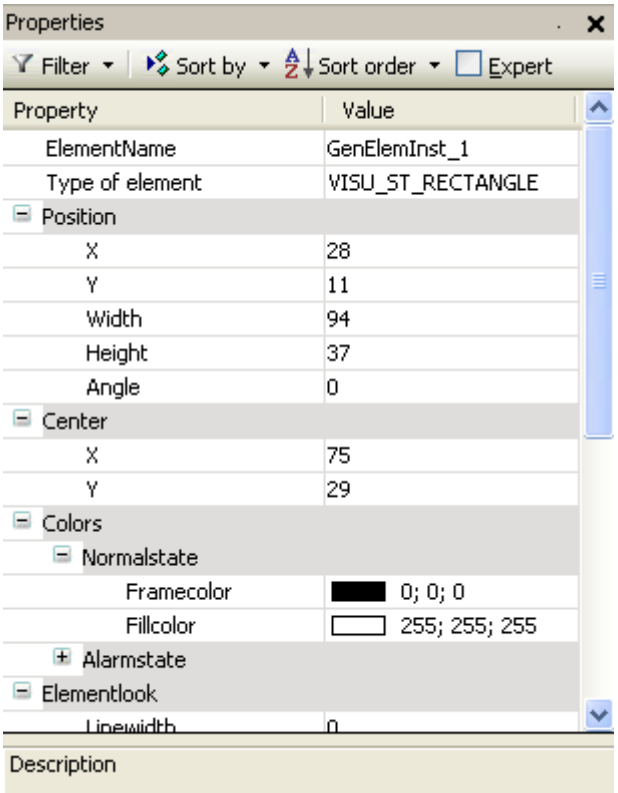


Abb. 441 Properties editor, example

Notice the possibility of using **sorting and filter functions** for the display of the of element properties list: Menu bar (Filter and Sorting).

The menu bar at top of the properties editor provides the following menus and options:

Filter menu: Select one of the filter settings in order to adapt the properties list to your needs:

Simple: Only the most basic properties (text, color, input)

Standard: The mostly used properties (selection bases on the property set used in V2.2 of the programming system)

Color: Only color referring properties

Animation: Only properties concerning any animation of the element (movement, toggle color, dynamic look)

Text: Only text referring properties

Input: Only properties concerning any user input

Show all categories: all properties are listed

Sort by menu:

Sort by type: The complete list of properties is arranged in the original order of categories.

Sort by name: The complete list of properties is arranged alphabetically concerning the category names.

Sort order menu:

Sort ascending: The complete list of properties is displayed in the original sequence of categories from down to up.

Sort descending: The complete list of properties is displayed the original sequence of categories from up to down.

Note: In visualizations associated to a device there might be device-depending restrictions concerning the configuration possibilities (for example: list of available elements, fonts, colors, image formats, max. number of visualization elements in a visualization, max. number of visualizations below the device).

The following tables contain the information on all possible properties. They can be displayed by clicking on the particular title.

Columns (only for Table)

The table element visualizes an array or structure variable. The index of the array elements resp. the components of a structure are displayed in one column resp. one row. Two-dimensional arrays or arrays of structures are displayed in multiple columns.

Here you define the look of the table columns in which the values of the particular array elements will be displayed

For each of the columns associated to index [x] of the array or to a certain component of the structure a separate configuration is available

Column		
[index]	column index of the table element; for example "[0]" for the single column representing the elements of array[2..8] of INT; for example "[0]", "[1]" or "[2]" for the three columns representing the elements of array[1..3][1..10] of INT.	Ta
Column header	by default the header displays the name of the array together with the index associated to the column; the header may be modified by entering a new title here	Ta
Column width	width of the column	Ta
Text in headline	alignment of column header: LEFT CENTER RIGHT	Ta
Use template	activating this checkbox a visual element of type rectangle, rounded rectangle or circle will be inserted in each element of the table column associated to the index (see above). The element properties list will be extended automatically by the element properties of this visual element that can be configured in the usual way.	Ta
Use text alignment in headline	activating this checkbox will only take effect if also the checkbox "Use template" is activated: If activated the settings made within the properties for font (size) and alignment will be applied for the associated column header as well.	Ta
Row header	checkbox to switch on or off the display of the labeling of rows (in form of associated row indices of the array displayed)	Ta
Column header	checkbox to switch on or off the display of the labeling of columns defined by "Column header"	Ta
Row height	height of table rows	Ta
Row header width	width of column containing the row headers	Ta
Scrollbar size	width of vertical and height of horizontal scrollbar	Ta

Static ID, Show frame, Clipping, Transparent, Transparent color Scaletype, Trace

Static ID	Identification of the image file to be used for this element (static definition) Enter the ID of the image file as used in the respective image pool (STRING); the name of the pool should be prefixed in order to make this entry unique (not necessary if the image is managed in the GlobalImagePool, because this one will be searched first anyway). For example: "imagepool2.button_image".	Im
Show frame	If activated: image will be displayed with a frame	F,Im
Clipping	If the Clip option is selected together with a "Fixed" Scaletype setting (see below), in case of a frame-element only that part of the visualization fitting into the frame will be displayed.	F,Im
Transparent	If activated, the color defined in property 'Transparent color' (see below) will be displayed transparently	Im
Transparent color	Button opens the color selection dialog for selecting a color which should be displayed transparently if option "Transparent" is activated.	Im
Scaletype	Specify here how the image should react on changes of the size of the elements frame. ISOTROPC: the image retains its proportions; thus even if height and width of the frame are modified independently, the heigth-width-ratio of the image will be kept. ANISOTROPIC: the image follows the size of the frame, so height and width of the image can be modified independently. FIXED: the original size of the image will be maintained regardless of the size of the frame. Notice if option 'Clipping' is also activated.	F,Im
Trace	Name of trace source for the visual trace element, The name is completed by a prefix reflecting the associated path within the device tree, for example "PLCWinNT.Application.Trace".	Tr

Position (X, Y, Width, Height, ShowCursor, Points)

Note: x/y = 0/0 is in the upper left corner of the editor window. x-axis= horizontal axis, y-axis=vertical axis.

X	Horizontal position in pixels. 0=left window border.	R,rR.E.L,F,Bu, Im	M,BD,Tr	Ta,Te,Sc
Y	Vertical position in pixels. 0=upper window border.	R,rR.E.L,F,Bu, Im	M,BD,Tr	Ta,Te,Sc
Width	Width of the element in pixels.	R,rR.E.L,F,Bu, Im	M,BD,Tr	Ta,Te,Sc
Height	Height of the element in pixels.	R,rR.E.L,F,Bu, Im	M,BD,Tr	Ta,Te,Sc
ShowCursor	Checkbox to be activated if you want the value of the traced variables at the position of the cursor to be displayed.		Tr	
Points				
[0], X / Y ..... [n], X / Y	X/Y-positions (pixels) of the particular element points. [0] is the starting point of the element. The subsequent points are numbered consecutively.	P,pL,C		

Center (X, Y)

X	X-position of the turning point (center, ⊕) of the element	R,rR,E,P,L,pL,C,F,Bu,Im
Y	Y-position of the turning point (center, ⊕) of the element	R,rR,E,P,L,pL,C,F,Bu,Im

Colors (Color, Bar color, Bar background, Framecolor, Alarm value, Condition, AlarmColor, Use color areas, Durable color areas, Begin of area, End of area, Color, Framecolor, Fillcolor, Normalstate, Alarmstate)

- - Choose a color from the selection list or open the color selection dialog via button

Color	Static definition: color of the element in basic state	L,F,Bu,Im		Sc
Bar color	Color of the bar display		BD	
Bar background	By default the background color of the bar is white; activating the checkbox will change the background color to black.		BD	
Framecolor	Color of frame surrounding bar display element in case of checkbox "Element frame" being activated		BD	
Alarm value	Barrier indicating alarm state		BD	
Condition	Alarm state will be set, if the value of the input variable exceeds (GREATER_THAN) or underruns (LESS_THAN) the Alarm value		BD	
AlarmColor	Static definition: color of the element in alarm state, that is when the respective variable is true	L,F,Bu,Im	BD	Sc
Use color areas	This checkbox being activated the specified color areas will be displayed		BD	
Color areas	A click on the button will cause the creation of a new color area, the properties of which can be defined in the text fields expanding below.			
Durable color areas	By default, only the color area embracing the actual value of the input variable will be displayed. Activating this checkbox, all color areas specified will be displayed.		M	
Areas	This folder contains all color areas specified			
[Number]	Number of color area in ascending order, may not be edited. A click on the button will erase the color area from the list and rearange their numbering.			
Begin of area	Value limiting area range from below		M,BD	

End of area	Value limiting area range from above		M,BD	
Color	Color the area shall be displayed in		M,BD	
Normalstate				
Framecolor	Static definition: color of the element frame in basic state	R,rR,E,P,pL,C		Te
Fillcolor	Static definition: fill color of the element in basic state;	R,rR,E,P,pL,C		Te
Alarmstate				
Framecolor	Static definition: color of the element frame in alarm state	R,rR,E,P,pL,C		Te
Fillcolor	Static definition: fill color of the element in alarm state	R,rR,E,P,pL,C		Te

Elementlook (Linewidth, Linestyle, FromTopLeft (declination), Fillattributes, Frameattributes, m_ShadowType)

These are static definitions concerning the looking of the outline and of the filling of the element.

For dynamic definition see category 'Look variables'.

Linewidth	Width of the outline in pixels; "0" equals "1"	R,rR,E,P,L,pL,C,F,Im	Te
Linestyle	Type of the line: PS_SOLID ____ PS_DASH _ _ _ PS_DOT . . . PS_DASHDOT _ . _ . PS_DASHDOTDOT _ . . _ . . PS_HOLLOW (not visible)	L	
FromTopLeft	If this option is activated, the line declines from top left to bottom right, otherwise it rises from bottom left to top right.	L	
Fillattributes	Look of the outline: BS_SOLID: the line is visible BS_HOLLOW: the line is not visible	R,rR,E,P,pL,C	Te
Frameattributes	Type of the outline: PS_SOLID ____ PS_DASH _ _ _ PS_DOT . . . PS_DASHDOT _ . _ . PS_DASHDOTDOT _ . . _ . . PS_HOLLOW (not visible)	R,rR,E,P,pL,C,F,Im	Te
m_ShadowType	Type of the shading of the outline: DEEPEND NONE RAISED		Te

Buttonheight (only for Button)

ButtonHeight	Determines the relief-view of the button element; "height" in pixels ( )	Bu

Bitmapinfo (only for Button: Static ID, Scaletype)

Static ID	see above	Bu
Scaletype	see above	Bu

Texts (Text, Tooltip)

These parameters are for a static definition of the elements' text. For a dynamic definition notice the categories 'Text variables' and Dynamic texts'.

For font and alignment definition notice category 'Text properties' (static) and 'Font variables' (dynamic).

Text	Text which should be displayed in the element; in case of a table element the text will be written in the foreground across the whole table. Example: "On/Off" see also below categories 'Text variables' and 'Dynamic texts' Example: "ON/OFF". See also: categories "Text variables" and "Dynamic texts". Note: The text can also be inserted "inline". When the element is selected in the editor window, pressing the <spacebar> opens a text input field.	R,rR,E,P,L,pL,C,F,Bu,Im	Ta,Te
Tooltip	text which should appear as tooltip in online mode... when the cursor is placed on the element. Example: "on/off button of machine 1"	R,rR,E,P,L,pL,C,F,Bu,Im	Ta,Te,Sc

Text properties (HorizontalAlignment, VerticalAlignment, Font)

These are static definitions, for a dynamic definition of the font notice category 'Font variables'.

HorizontalAlignment	Horizontal alignment of the elements' text: LEFT / HORIZONTAL / RIGHT	R,rR,E,P,L,pL,C,F,Bu,Im	Ta,Te,Sc
VerticalAlignment	Vertical alignment of the elements' text: TOP / VCENTER / BOTTOM	R,rR,E,P,L,pL,C,F,Bu,Im	Ta,Te,Sc
Font	Font type, color and size to be used for the elements' text; button ... opens the standard dialog for defining the font;	R,rR,E,P,L,pL,C,F,Bu,Im	Ta,Te,Sc

Arrow (only for Meter: Arrowtype, Arrowcolor, Arrow start, Arrow end, Additional arrow)

Arrowtype	the drop-down menu provides four different shapes for the arrow:THIN_NEEDLE, THIN_ARROW, NORMAL_ARROW, WIDE_ARROW	M
Arrowcolor	color of arrow	M
Arrow start	angle (in degree) between left boundary of tachometer and horizontal axis ranging from 0 to 360	M
Arrow end	angle (in degree) between right boundary of tachometer and horizontal axis ranging from 0 to 360	M
Additional arrow	this checkbox being activated a second arrow will be displayed which is located in the scale area and opposed to the original arrow	M

Bar (Diagram type, Orientation, Running direction)

Diagram type	the drop-down menu provides three different possibilities for locating the bar with respect to the scaling : BAR_INSIDE_SCALE, SCALE_INSIDE_BAR, SCALE_BESIDE_BAR	BD	
Orientation	chose between HORIZONTAL and VERTICAL orientation of the bar; in case of the scrollbar, the orientation results from the ratio of length to widthand may not be edited.	BD	SC
Running direction	chose between LEFT_RIGHT (for a bar growing from left to right) and RIGHT_LEFT in case of a horizontal bar, or between BOTTOM_UP and TOP_DOWN in case of a vertical one	BD	SC

Scale (Scale start, Scale end, Main Scale, Sub Scale, Frame inside, Frame outside, Element Frame)

Scale start	value limiting the scaling range from below	M,BD
Scale end	value limiting the scaling range from above	M,BD
Main scale	distance between two scaling marks indicating coarse scale	M,BD
Sub scale	distance between two scaling marks indicating fine scale; can be set to 0, if no subscaling marks shall be displayed	M,BD
Frame inside	by default this checkbox is activated; deactivating will make the lower frame line of the circular scaling arc disappear	M
Frame outside	by default this checkbox is activated; deactivating will make the upper frame line of the circular scaling arc disappear	M
Element Frame	by default this checkbox is deactivated; activating will insert a frame around the bar display	BD

Label (Unit, Font, Scale Format)

LabelPosition	chose "OUTSIDE" to locate the scale labels on the outside of the circular scaling arc, "INSIDE" to locate the scale labels on the inside of the circular scaling arc	M
Unit	string that will be displayed below the arrow (M) or below the midpoint of the range(BD) (per default the string is thought to indicate the unit used for scaling)	M,BD
Font	font for unit and scaling range	M,BD
Scale Format (C-Syntax)	use the C syntax for indicating the formating of the scale labels; e.g .entering the string "%3.2f s" in this edit field will cause the scale labels to be displayed with 3 digits,whereof two digits are located behind the decimal point and followed by the letter "s"	M,BD
max. text width of labels	value limiting the maximum width of text labels; usually the width of the labels will be adjusted automatically. Use this edit entry in case of failure of the automatic adjustment.	M
text height of labels	value defining the height of text labels; usually the height of the labels will be adjusted automatically. Use this edit entry in case of failure of the automatic adjustment.	M

Bitmap ID variable (only for Image)

Bitmap ID	project variable defining the image ID (see image pool)	Im

Absolute movement (Movement, X, Y, Rotation, Scaling, Interior Rotation)

Absolute movement: The element can be moved by changing the X- and Y-position (pixels) of the element upper left corner by a variable of type INT. Absolute coordinates are used here.

See in contrast: 'Relative movement' for shifting specified corners/borders of the element relative to a fix origin.

Movement		
X	enter an integer variable giving the current X- position of the upper left corner of the element in pixels; this can be used to shift the element in X-direction	R,rR,E,P,L,pL,C,F,Bu,Im
Y	enter an integer variable giving the current Y- position of the upper left corner of the element in pixels ; this can be used to shift the element in Y-direction	R,rR,E,P,L,pL,C,F,Bu,Im
Rotation	enter an integer variable defining the angle (angular degrees) by which the element will be moved around the turning point; positive values = mathematic positive = clockwise. notice that the element itself will not rotate, in contrast to 'Interior rotation' (see below) The turning point (center ⊕) appears after a single click on the element. You can drag it with a pressed left mouse button.	R,rR,E,P,L,pL,C,F,Bu,Im
Scaling	enter an integer variable defining the current scale value (percents); the element size will be changed linear according to this value. Implicitly the value will be divided by 1000, so that it is not necessary to use REAL-variables in order to get a reduction of the element. The scaling always will refer to the turning (center) point. The turning point (center ⊕) appears after a single click on the element. You can drag it with a pressed left mouse button.	R,rR,E,P,L,pL,C,F,Bu,Im
Interior rotation	enter an integer variable defining the angle (angular degrees) by which the element will rotate around its turning point; positive values = mathematic positive = clockwise. in contrast to 'Rotation' (see above) the element will really rotate The turning point (center ⊕) appears after a single click on the element. You can drag it with a pressed left mouse button.	R,rR,E,P,L,pL,C,F,Bu,Im

Relative movement (Movement topleft, Movement bottomright, X, Y)

The left, top, right and bottom edges of the element will be moved according to the x- and y-position defined here by a variable of type INT. This variable in contrast to the absolute movement (see above) specifies a "relative position, that is the distance to the initial position.

Positive values will shift the horizontal edges downward resp. the vertical edges, to the right.

See also: 'Absolute movemen't for shifting the whole element.

Movement topleft		
X	number of pixels by which the left edge will be shifted in x-direction	R,rR,E,L,F,Bu,Im
Y	number of pixels by which the top edge will be shifted in y-direction	R,rR,E,L,F,Bu,Im
Movement bottomright		
X	number of pixels by which the right edge will be shifted in x-direction	R,rR,E,L,F,Bu,Im
Y	number of pixels by which the bottom edge will be shifted in y-direction	R,rR,E,L,F,Bu,Im

⊞Dynamic values (for elements with multiple position points: pPointArray, Count)

pPointArray	Enter a variable which is a pointer to an array of structure **VisuStructPoint**. VisuStructPoint by components **iX** and **iY** describes the x/y-coordinates of a element's point. The array describes all points of the element and the structure can be filled dynamically by the project. Notice that the number of the elements points must be defined explicitly (see below) because using a pointer here for the dynamic points definition does not allow to control the number of the points. Example: pPoints : POINTER TO ARRAY[0..100] OF VisuStructPoint;	P,pL,C
Count	Enter variable of the type INT which defines the number of points of the element. Example: iCount : INT:=24; The element is defined to have 24 particular points. This specification is needed because the definition of the particular points is given by a pointer (see above pPointArray) and does not allow control on the number.	P,pL,C

⊞Text variables (Text Variable, Tooltip Variable)

These are dynamic definitions, see also category 'Dynamic texts' for using textlists.

For static definitions see category 'Texts'.

Text Variable	String variable defining the elements' text	R,rR,E,P,L,pL,C,F,Bu,Im	Te,Sc
Tooltip Variable	String variable defining the elements' tooltip text	R,rR,E,P,L,pL,C,F,Bu,Im	Te,Sc

Dynamic texts (TestList, TextIndex, TootipIndex)

These parameters serve for a dynamic text definitions basing on textlists (allows language switching!).

For another dynamic text definition, by providing the text via a string variable, see category 'Text variables'.
For static text definitions see category 'Texts'.

TextList	Name of the textlist object (string) as used in the POUs or devices view; for example " 'errorlist' "	R,rR,E,P,L,pL,C,F,Bu,Im	Te,Sc
TextIndex	Index of the text as defined in the textlist (string); enter the ID directly, for example 'M1' or a string variable; for example "error_22".	R,rR,E,P,L,pL,C,F,Bu,Im	Te,Sc
TooltipIndex	Index of the text as defined in the textlist (string); enter the ID directly, for example 'M1' or a string variable; for example "error_22_tooltip".	R,rR,E,P,L,pL,C,F,Bu,Im	Te,Sc

Font variables (Fontname, Height, Flags, CharSet, Color)

These are dynamic definitions for the elements' text font via a project variable.

For a static definition see category 'Texts'.

Fontname	enter a variable of type STRING providing the name of the font to be used for the elements' text; (as used in the standard dialog for defining the font); for example "plc_prg.font_var" (font_var:='Arial';)	R,rR,E,P,L,pL,C,F,Bu, Im	Ta,Te,Sc
Height	enter a variable of type INT providing the height of the elements' text in pixels; (as used in the standard dialog for defining the font); for example" prog1.font_height" (font_height:=16;)	R,rR,E,P,L,pL,C,F,Bu, Im	Ta,Te,Sc
Flags	enter a variable of type DWORD defining the font display by one of the flags listed below. A combined definition can be reached by addition of the flag numbers: 1 italic 2 bold 4 underlined 8 canceled for example "prog2.font_type" (if font_type:=6, the text will be displayed bold and underlined)	R,rR,E,P,L,pL,C,F,Bu, Im	Ta,Te,Sc
CharSet	The character set for the font to be used can be specified by the standard character set number. Enter an variable of type DWORD providing this number (see also the "skript" selection in the default font selection dialog)	R,rR,E,P,L,pL,C,F,Bu, Im	Ta,Te,Sc
Color	Enter a DWORD variable specifying the color for the elements' text.	R,rR,E,P,L,pL,C,F,Bu, Im	Ta,Te,Sc

Color variables (ToggleColor, Color, Alarmcolor, Normalstate, Alarmstate, Framecolor, Fillcolor)

ToggleColor	enter a boolean variable which toggles the element color between 'Color' (FALSE) and 'Alarmcolor' (TRUE)	R,rR,E,P,L,pL,C,F,Bu,Im	Ta
Color	enter a DWORD variable determining the fill color of the element (overwrites the value currently set by "Colors/Normal state"). The "Color" is used when the variable defined in 'ToggleColor' (see above) is FALSE.	L,Bu,Im	
Alarmcolor	enter a DWORD variable determining the alarm color of the element (overwrites the value currently set by "Colors/Alarm state") The "alarm color" is used when the variable defined in 'ToggleColor' (see above) is TRUE.	L,Bu,Im	
Normalstate			
Framecolor	define frame color of the element in normal state as described above, 'Color'	R,rR,E,P,L,pL,C	Ta
Fillcolor	define fill color of the element in normal state as described above	R,rR,E,P,L,pL,C	Ta
Alarmstate			
Framecolor	define frame color of the element in alarm state as described above, 'Alarmcolor'	R,rR,E,P,L,pL,C	Ta
Fillcolor	define fill color of the element in alarm state as described above	R,rR,E,P,L,pL,C	Ta

Look variables (Linewidth, Fillattributes, Frameattributes)

These are dynamic definitions concerning the looking of the outline and of the filling of the element.

For static definition see category 'Elementlook'.

Linewidth	Enter an integer variable defining the line width of the element in pixels	R,rR,E,P,pL,C,F,Im	Ta
Fillattributes	Enter a DWORD variable defining the fill attributes of the element: The color, as defined by the color variables, can be displayed or ignored: 0 = show color >0 = ignore setting	R,rR,E,P,pL,C	Ta
Frameattributes	Enter a DWORD variable defining the frame attributes of the element: Type of the outline: 0 = full, PS_SOLID ____ 1 = dashed, PS_DASH _ _ _ 2 = dotted, PS_DOT . . . 3 = dash-point, PS_DASHDOT _ . _ . 4 = dash-point-point, PS_DASHDOTDOT _ . . _ . . 8 = line is not visible, PS_HOLLOW	R,rR,E,P,pL,C,F,Im	Ta

State variables (Invisible, Deactivate inputs)

These are dynamic definitions on the availability of the element in online mode.

Invisible	Enter a boolean variable; if the variable provides TRUE, the element will be invisible in online mode.	R,rR,E,P,L,pL,C,F,Bu	Ta,Sc
Deactivate inputs	Enter a boolean variable; if the variable provides TRUE, any inputs on the element will be of none effect.	R,rR,E,P,L,pL,C,F,Bu	Ta,Sc

m_pLinewidthVariable Variable (only for Line: IntValue)

Dynamic definition of the line width via a variable:

IntValue	Enter an integer variable which provides the line width in pixels. "0" equals "1" This corresponds to the fix setting LineWidth in category "Elementlook".	L

m_pLinestyleVariable Variable (only for Line: IntValue)

Dynamic definition of the line style via a variable:

IntValue	Enter an integer variable which provides the line style. Possible values: 0 = full, PS_SOLID ____ 1 = dashed, PS_DASH _ _ _ 2 = dotted, PS_DOT . . . 3 = dash-point, PS_DASHDOT _ . _ . 4 = dash-point-point, PS_DASHDOTDOT _ . . _ . . 8 = line is not visible, PS_HOLLOW This corresponds to the fix setting LineStyle in category "Elementlook".	L

m_pLinedirectionVariable Variable (only for Line: DigitalVar)

Dynamic definition of the declination of the line via a variable:

DigitalVar	Enter an boolean variable which determines whether the line should decline from top left to bottom right (TRUE) or should rise from bottom left to top right (FALSE, default) This corresponds to the fix setting "FromTopLeft" in category "Elementlook".	L

Selection (only for Table: Selection color, Apply to columns, Selection type, Frame around selected cells, Variable for selection X, Variable for selection Y, Variable for valid selection X, Variable for valid selection Y)

Selection		
Selection color	fill color for selected cells	Ta
Apply to columns	by pressing this button the "Selection color" will overwrite the "Alarmstate - Fillcolor" of templates that would normally indicate a selection	Ta
Selection type	defines which part of the table will be selected by clicking on one table cell: SEL_NONE (no selection) SEL_CELL (the cell will be selected) SEL_ROW (selection of the row containing the cell) SEL_COLUMN (selection of the column containing the cell) SEL_ROW_AND_COLUMN (selection of row and column containing the cell)	Ta
Frame around selected cells	checkbox to switch on or off a frame around the selected part	Ta
Variable for selection X	variable of type INT that gets assigned to the array index related to the row of the selected element	Ta
Variable for selection Y	variable of type INT that gets assigned to the array index related to the column of the selected element; in case of a structure / POU the entries will be enumerated sequentially starting with 0 (Note: Be aware that the value represents the correct position within the array/structure/POU only if no columns are suppressed from being displayed within the table)	Ta
Variable for valid selection X	boolean variable that gets assigned to a value indicating a valid selection of a column within the table	Ta
Variable for valid selection Y	boolean variable that gets assigned to a value indicating a valid selection of a row within the table	Ta

Buttonstate variable (only for Button)

DigitalVar	Enter a boolean variable, which will control the display of the button as "pressed" (TRUE) resp. "not pressed" (FALSE).	Bu

Bitmap ID variable (only for Button)

This is a dynamic definition of the bitmap to be displayed in the button element.

For static definition see category 'Bitmapinfo'.

Bitmap ID	Enter a string variable providing the ID of the desired bitmap as defined in the respective image pool. See also the description in category 'Bitmapinfo'.	Bu

Isotropic (only for Frame)

Isotropic	see above	F

Clipping (only for Frame)

Clipping	see above	F

References (only for Frame: References, visualizations)

References		F
visualizations	The visualizations assigned to the frame by the Frame Configuration dialog are displayed here; **no input possible!**	

Inputs (OnDialogClosed, OnMouseClick, OnMouseDown, OnMouseLeave, OnMouseEnter, OnMouseUp, OnMouseMove, Toggle: Variable, Toggle on up if captured, Tap: Variable, Tap False, Tap on enter if captured) - Mouse actions: Write a Variable (edit field, numpad, keypad), Execute command (execute programs, print, recipe handling), Change the language, Change Shown Visualization, Execute ST-Code, Switch Framevisualization, Toggle a Variable, Open Dialog, Close Dialog, Hotkey)

Here you define what should happen if the user in online mode performs an input on the visualization element.

You can configure a "OnDialogClosed action", "OnMouse actions" as well as toggling or tapping of variables and a hotkey definition.

"OnDialogClosed":

This input is given by the closing of any dialog within the visualization, which has been opened before for user input purposes. You can assign one or several of the after-actions described below. See there also the descriptions of the after-actions "Open dialog" and "Close dialog", which can be used to open and close an user input dialog by a mouse action on a visualization element.

"OnMouse"-actions:

To each of the particular "OnMouse" actions (see table below) you can assign one or several of the after-actions. As long as no after-action is defined for a OnMouse-action on the current element, "Configuration" appears in the respective properties field. By a mouse-click on this field the 'Input Configuration' dialog will be opened, see below, where an after-action can be selected. An already assigned after-action will be displayed in the properties dialog below the respective OnMouse input entry and the corresponding configuration dialog will open on a double-click in its value field.

Toggle, Tap:

Here you can enter boolean project variables, which should get TRUE or FALSE on a mouse-click on the element resp. on a click + release of the mouse-button.

Hotkey:

Here you can define a key and associate a certain action (MouseDown, MouseUp), which should be executed on the respective movement of the key (KeyDown, KeyUp). Per default on KeyDown the MouseDown action gets executed and on KeyUp the MouseUp action. This might be useful if a visualization should be operated via mouse as well as via keyboard, because the input actions must be configured only once. This part of the element configuration will also be administered in the hotkeys configuration editor of the visualization. Any changes always will be synchronized immediately in the hotkeys editor as well as in the element properties.

See the following configuration options:

OnDialogClosed	defines what should happen, if a dialog, which has been opened before by an input on an element of the current visualization, gets closed. For example by using the "Execute ST Code" option you might implement a specific reaction on the given "result" of the lastly closed dialog (this result, for example "OK" or "Cancel", is defined in the"Close dialog" input configuration of an element) ; Regard: The OnDialogClosed property is not restricted to the current element, but valid in the whole visualization, that is reacts on each dialog-closing action; however currently there is no possibility to define a property for the complete visualization and therefore it must be assigned to one of its elements.	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouse actions			
OnMouseClick	mouse action:when the cursor is pointing on the element, perform a mouse-click (press and leave the mouse-button). *	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouseDown	mouse action: press mouse-button when the cursor is pointing on the element *	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouseLeave	mouse action: release the pressed mouse-button when the cursor is still pointing on the element *	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouseEnter	mouse action: draw the mouse-button on the element *	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouseUp	mouse action: press the mouse-button before drawing the cursor on the element, release the button when the cursor is pointing on the element *	R,rR,E,P,L,pL, C,F,Bu,Im	Te
OnMouseMove	mouse action: move the cursor *	R,rR,E,P,L,pL, C,F,Bu,Im	Te

Toggle			
Variable	here you can enter an boolean variable, which alternately will get TRUE or FALSE on each mouse-click on the element. The input assistant can be used via button ..., which is available after a double-click on the properties field..	R,rR,E,P,L,pL, C,F,Bu	Te
Toggle on up if captured	if this option is deactivated (default): if you press the mouse-button while the cursor is pointing on the element, but not released before the cursor has been moved out of the element, the variable defined above will not be toggled. So a started toggle-input can be aborted. if this option is activated: the boolean variable defined above will be toggled even if the mouse-button is not released before the cursor has been moved out of the element.	R,rR,E,P,L,pL, C,F,Bu	
Tap			
Variable	here you can enter a boolean variable, which will get TRUE as soon as the mouse-button is pressed while the cursor is pointing on the element, and which will get FALSE again when releasing the mouse-button.	R,rR,E,P,L,pL, C,F,Bu	Te
Tap False	If this option is activated, the tap-behaviour described for the above define variable will be reversed. That means that the variable will bet FALSE as soon as the mouse-button is pressed and it will get TRUE when releasing the button.	R,rR,E,P,L,pL, C,F,Bu	Te
Tap on enter if captured	if this option is activated, the above defined variable will be tapped even if the mouse-button has been pressed before drawing the cursor on the element.	R,rR,E,P,L,pL, C,F,Bu	Te
Hotkey		R,rR,E,P,L,pL, C,F,Bu	Te
Key	name of the key, e.g. "M", a selection list provides the currently supported keys		
Events	action to be executed when using the key + possibly defined modifier keys (Shift, Control, Alt); possible values provided in a selection list: - None (no action) - Mouse down (OnMouseDown action when key is pressed) - Mouse up (OnMouseUp action when key is released) - Mouse down/up (OnMouseUp and OnMouseDown actions when key is pressed or released)		
Shift	if this option is activated, the Key must be combined with the Shift key		
Control	if this option is activated, the Key must be combined with the Ctrt key		
Alt	if this option is activated, the Key must be combined with the Alt key		

*** "OnMouse..."- after-actions:**

One or several of the after-actions described in the following can be defined for a certain action (OnMouse..., tap, toggle) on a visualization element. This is to be done in dialog 'Input Configuration', which will open on a mouse-click on the 'value' field ("Configure...").

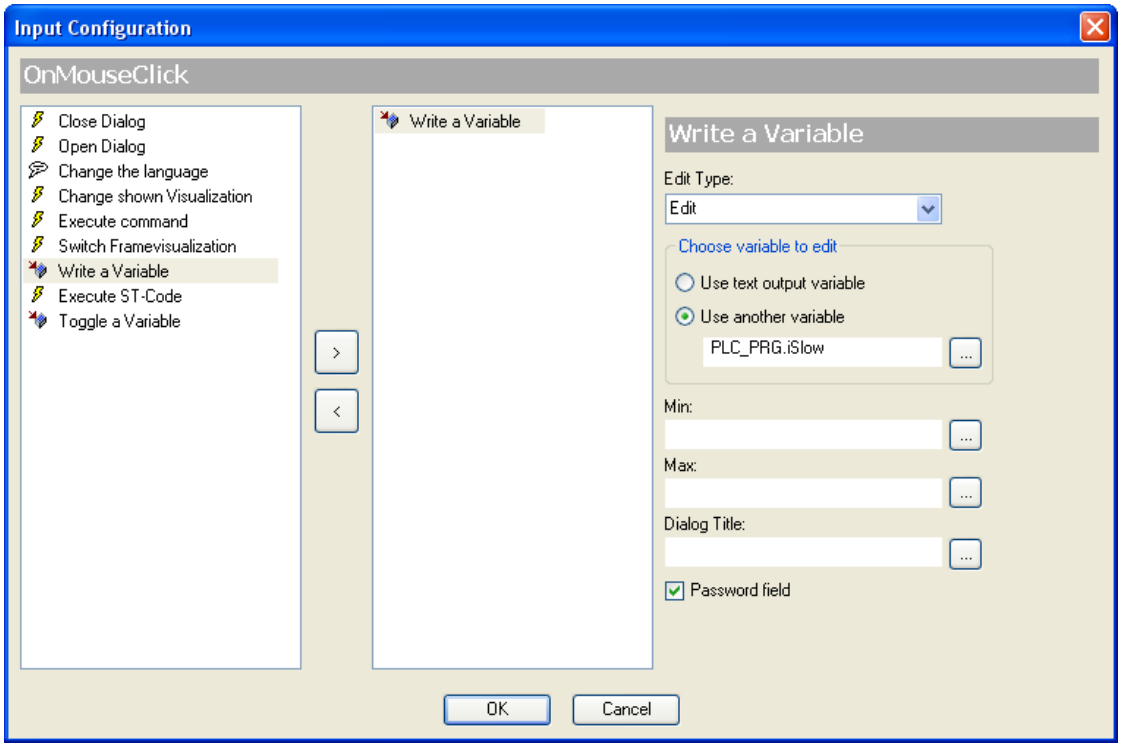


Abb. 442 Input Configuration dialog

In the left part of the dialog see in the title bar the name of the currently configured OnMouse-action. In the field below the possible after-actions are listed. In the center part see the after-actions already assigned to the OnMouse-action. The right part always provides the appropriate configuration options for the action which is currently selected in the center part.

In order to define one or several after-actions for the current mouse-action, select the after-action from the left field and use the right-arrow-button for copying it to the center part. Then select the action in the center part and fill the configuration fields in the right part. The left-arrow-button allows to remove an action which is currently selected in the center part.

See in the following the configuration options for the particular after-actions:

- - Write a Variable (edit field, numpad, keypad)

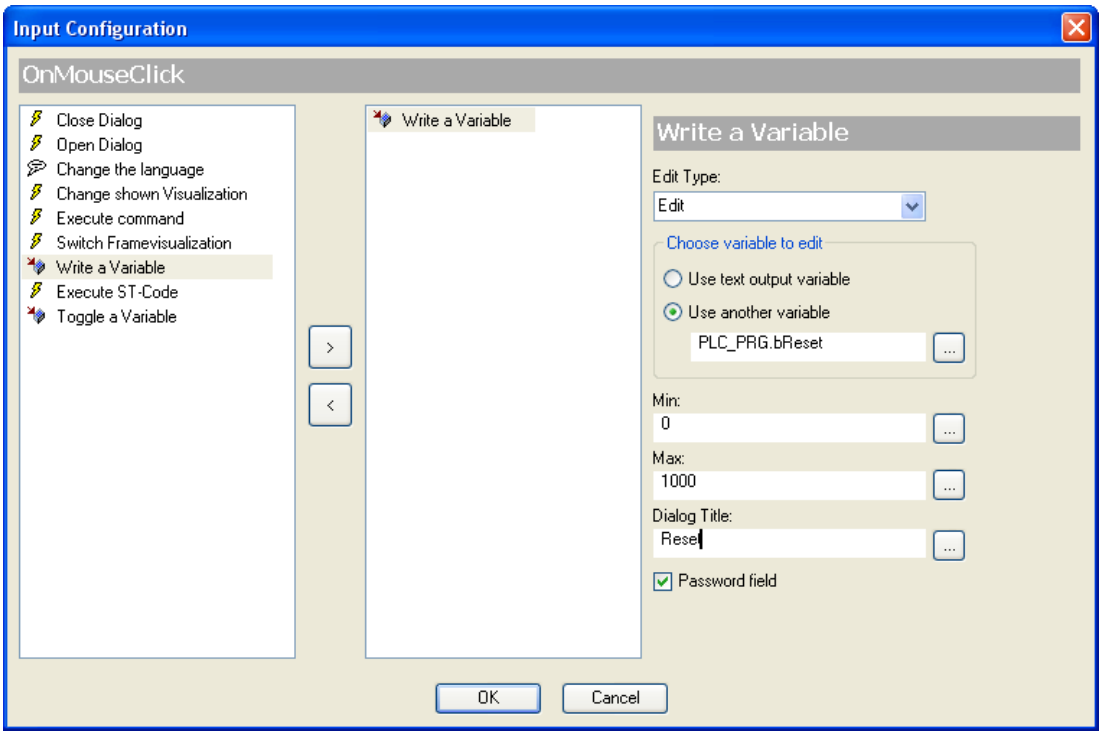


Abb. 443 Input Configuration, Write a variable

If there is an input configuration of type 'Write a Variable' for a visualization element, then as soon as the respective mouse action is performed on this element in online mode, the element will provide a possibility to enter a value. The value can be typed in as a sequence of characters or via a numpad or keypad and after having terminated the input will be written to the project variable specified in the Input Configuration dialog. The value will be interpreted according to the type of the project variable as a text string or as a numeric value.

Using the configuration dialog:

From the selection list choose one of the following input types:

Edit: The mouse action will open an edit field, where you can type in a string or numeric value.

VisuDialogs.Keypad: The mouse action will open a simulated keypad. You can enter a text string by mouse-clicks on the respective keys.

VisuDialogs.Numpad: The mouse action will open a simulated numpad. You can enter a numeric value by mouse-clicks on the respective keys.

Additionally, if there are user-defined visualizations in the project, which are specified in their properties to be used as "numpad/keypad":
<device>.<application>.<numpad/keypad-visualization>: The mouse action will open the numpad or keypad, which is represented in the chosen visualization. Notice that for this purpose the visualization option Register images files without path must be activated in order to get the needed background bitmaps loaded.

Note: The use of keypad/numpad requires the library VisuDialogs.library to be integrated in the library manager.

In the **Variable** field enter the project variable to which the value, typed in by the user, should be written. The input assistant might be opened for this purpose via the button.

A minimum and maximum value might be defined in the **Min.** and **Max.** fields by specifying a value or a project variable.

A title which should be displayed in the dialog title bar can be defined via a text string or a text variable in the **Dialog title** field.

If you want the entry to be displayed hidden by "**********" (as in case of a password), mark the checkbox in front of **Password field**.

Example:

See for example the picture above and assume that this is the configuration dialog of a rectangle element. If in online mode the mouse-key is pressed on this rectangle, a numpad will be displayed, showing the title given by PLC_PRG. MyTitle.

Enter a value: For example by mouse-clicks on 1, 2 and 3 enter "123". This value will be displayed in the upper part of the dialog, where also the minimum and maximum values as defined by PLC_PRG.minVal and PLC_PRG.maxVal are shown. As soon as you have confirmed the input by a mouse-click on the numpad OK button, "123" will be written to variable PLC_PRG.ivar. If "ivar" is declared of type STRING, it will get value '123', if it is of a numeric type it will get value 123.

- Change the language

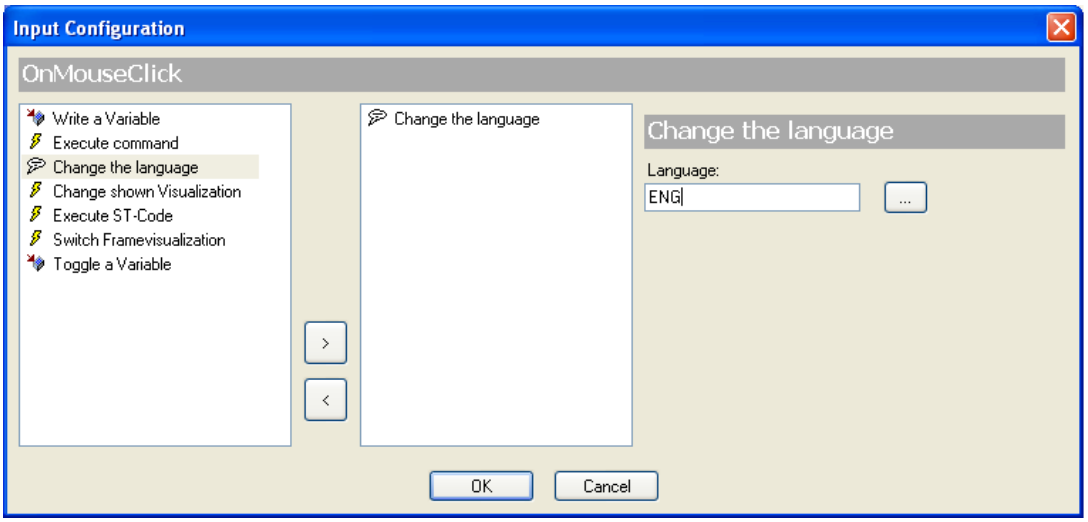


Abb. 444 Input Configuration, Change the language

Enter the language which should be used for the display of the visualization texts. Use the language identifier as defined in the corresponding textlist(s). Alternatively you may click on ... for selecting the appropriate language within the input assistant.

- Open dialog

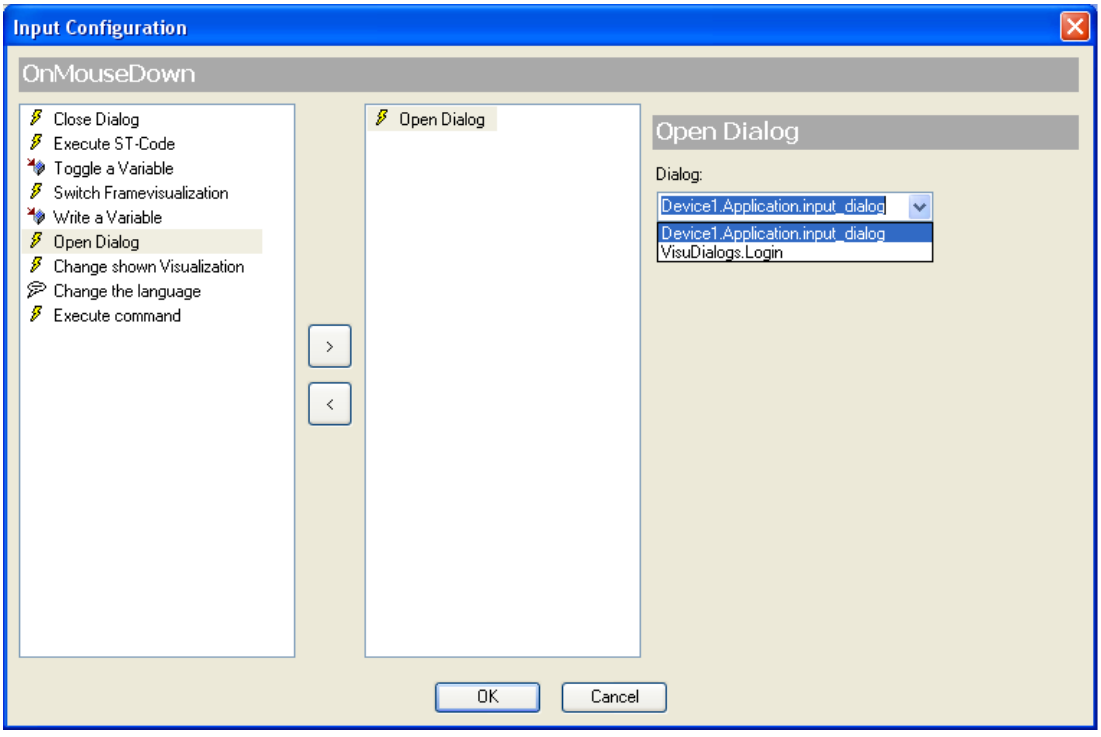


Abb. 445 Input Configuration, Open dialog

Here you can define that on the mouse-action a dialog should be opened, which is realized by another (default or user-designed) visualization. The selection list will offer all visualizations which are defined in their object properties as to be used as a dialog. So you might provide a self-designed dialog as user input mask in a visualization.
At least the default object "VisuDialogs.Login" will be available.

- Execute command (execute programs, print, recipe manager commands)

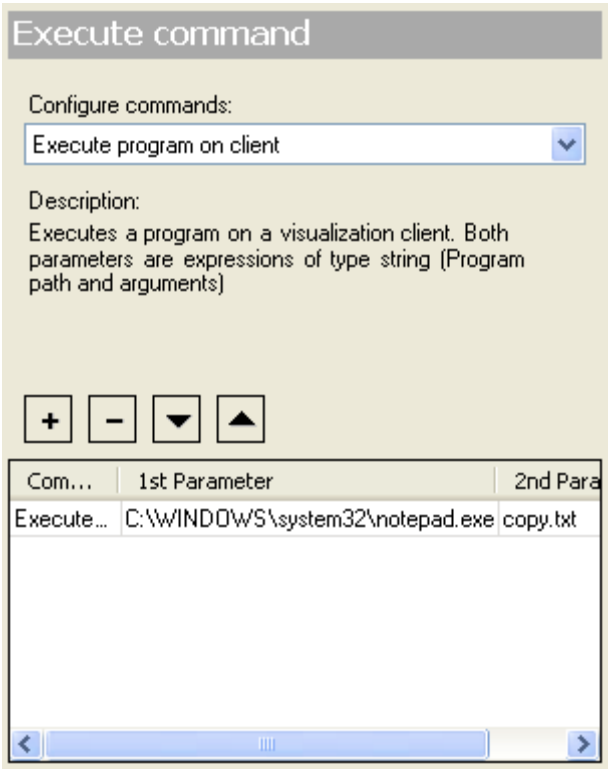


Abb. 446 Input Configuration, Execute command

Here you can define a sequence of commands to be executed on the mouse action.
From selection list **Configure commands** choose the desired command and use button + in order to enter the command in the table in the lower part of the dialog. This table always contains the commands currently selected for the local input configuration.
A short description on the command currently selected in the upper list always appears in the center part of the dialog. Accordingly supply the parameters for the command in the table columns **1st Parameter** and **2nd Parameter**. (Column **command** shows the internal command name.) See also further below here on the help page the descriptions of the particular commands.
Via button − the currently selected entry can be removed from the table.

Later in online mode, when the user input will be performed on the visualization element, the configured commands will be executed in that order in which they are arranged here in the table from up to down. You can change this order by shifting a currently selected entry by the buttons or .

The following commands are available:

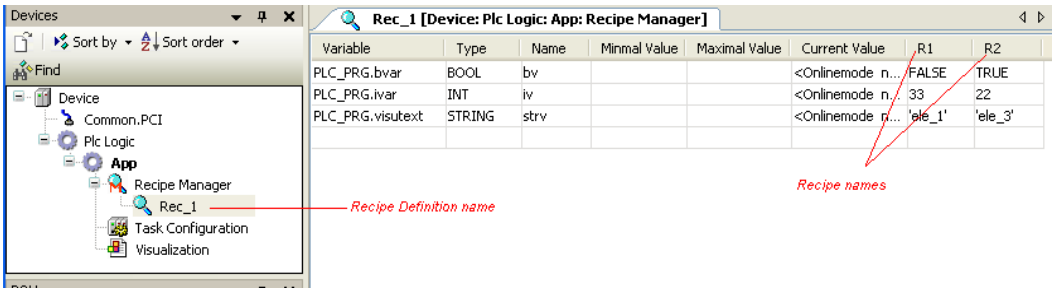
Execute program on the PLC	Executes the given program on the controller (PLC) resp. the visualization client. 1st parameter: STRING, path of the program (for example "C:\programs\notepad.exe" 2nd parameter: STRING, arguments needed by the program (for example name of file to be opened: "copyfile.txt")
Execute program on the client	
Print	The standard Print dialog gets opened, where settings for the print area, the printer parameters etc. can be configured and a printout of the current visualization can be generated.

For the following commands concerning recipes notice the following:

The "**recipe definition**" is an object in the devices tree below the Recipe Manager. It consists of a list of variables and the recipes defined for these variables (that is variables values). It can be edited in the Recipe Manager editor window.

The "**recipe name**" is the name of a particular recipe for the variables within a recipe definition. The recipe names are there visible as column headers.

▶Recipe definition, Recipe name



Create recipe	A new recipe will be created in the specified recipe definition. 1st parameter: Recipe definition name 2nd parameter: Name for new recipe
Delete recipe	The specified recipe will be removed from the specified recipe definition. 1st parameter: Recipe definition name 2nd parameter: Recipe name

Safe a recipe in a file	The values of the specified recipe will be written to a file with extension *.txtrecipe, the name of which you have to define. For this purpose the standard dialog for saving a file in the local file system will open. ATTENTION: The implicitly used recipe files, necessary as a buffer for reading and writing of the recipe values, may not get overwritten. This means that the name for the new recipe file must be different from "<recipe name>.<recipe definition name>.txtrecipe" ! 1st parameter: Recipe definition name 2nd parameter: Name of the particular recipe to be saved in the specified file
Load a recipe from a file	The recipe which has been stored in a file (see above "Save recipe") can be reloaded from this file. The standard dialog for browsing for a file will open for this purpose. The filter automatically is set to extension "*.txtrecipe". After having reloaded the file, the recipe values will be updated accordingly in the Recipe Manager. 1st parameter: Recipe definition name 2nd parameter: Name of the particular recipe, for which the values should be loaded from the specified file to the Recipe Manager.
Read recipe	The current values of the variables of the specified recipe definition will be read from the controller and be "written" to the specified recipe. This means that the values will be stored implicitly (in a file on the PLC) and also immediately will be monitored in the recipe definition table in the Recipe Manager. In other words: The recipe managed in the Recipe Manager gets updated with the actual values from the PLC. 1st parameter: Recipe definition name 2nd parameter: Name of recipe for which the values should be read from the PLC and be stored as actual recipe values.
Write recipe	The values of the given recipe, as visible in the Recipe Manager, will be written to the variables on the PLC. 1st parameter: Recipe definition name 2nd parameter: Name of the particular recipe the values of which should be written to the variables on the PLC.

- Close dialog

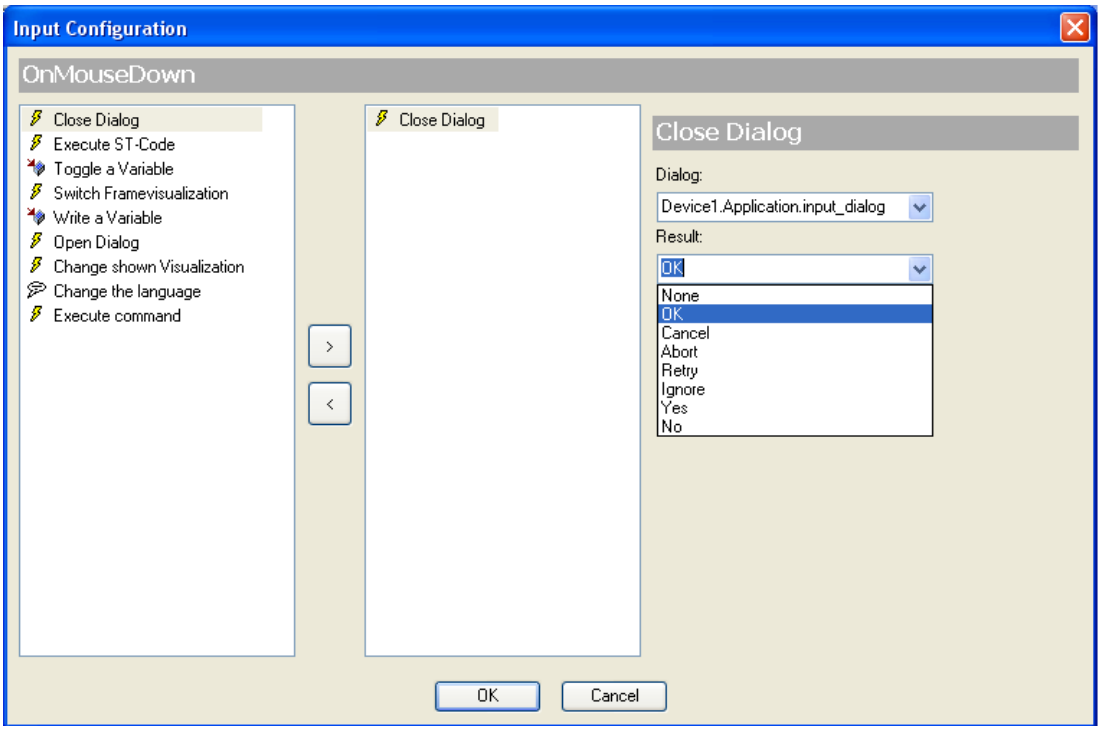


Abb. 447 Input Configuration, Close Dialog

Here you can define that on the mouse-action the specified dialog will be closed with the specified result. Choose the desired dialog from the selection list, which - like in the 'Open Dialog' configuration (see above) - offers all currently available "dialog" visualizations.

The 'Result' list offers the standard options which are used in dialogs that require an user response: OK, Cancel, Abort, Retry, Ignore, Yes, No.

Notice that the result, given by the lastly closed dialog, can be scanned and a respective reaction on this result can be configured in any element of the current visualization. For this purpose use the input configuration parameter 'OnDialogClosed'.

- Change Shown Visualization

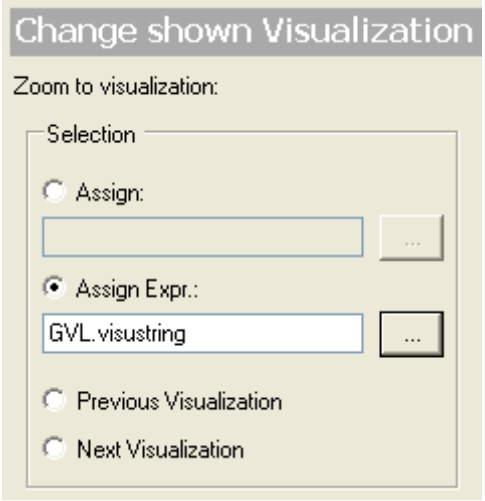


Abb. 448 Input Configuration, Change Shown Visualization

Select one of the following options to define which visualization should be displayed in online mode on the respective mouse action:

Assign: A visualization can be specified. The Input Assistant will open for this purpose and offer all visualizations currently available in the project . The selected visualization will be entered with the complete path, that is "<device name>.<application name>.<visualization name>".

Assign Expr.: A global variable can be specified which determines the name of the visualization to be shown. The Input Assistant will open for this purpose. The selected variable will be entered, for example "GVL1.visustring".

The sequence in which visualizations get displayed in online mode due to user inputs will be stored internally. With the following two options you can jump back or forward within this sequence:

Previous Visualization: The previously displayed visualization will be shown again. If no visualization had been displayed previously, the current one will be kept.

Next Visualization: The visualization following to the active one in the stored sequence of display will be shown. This is only possible after previously having changed back from this visualization via 'Previous Visualization'.

- Execute ST-Code

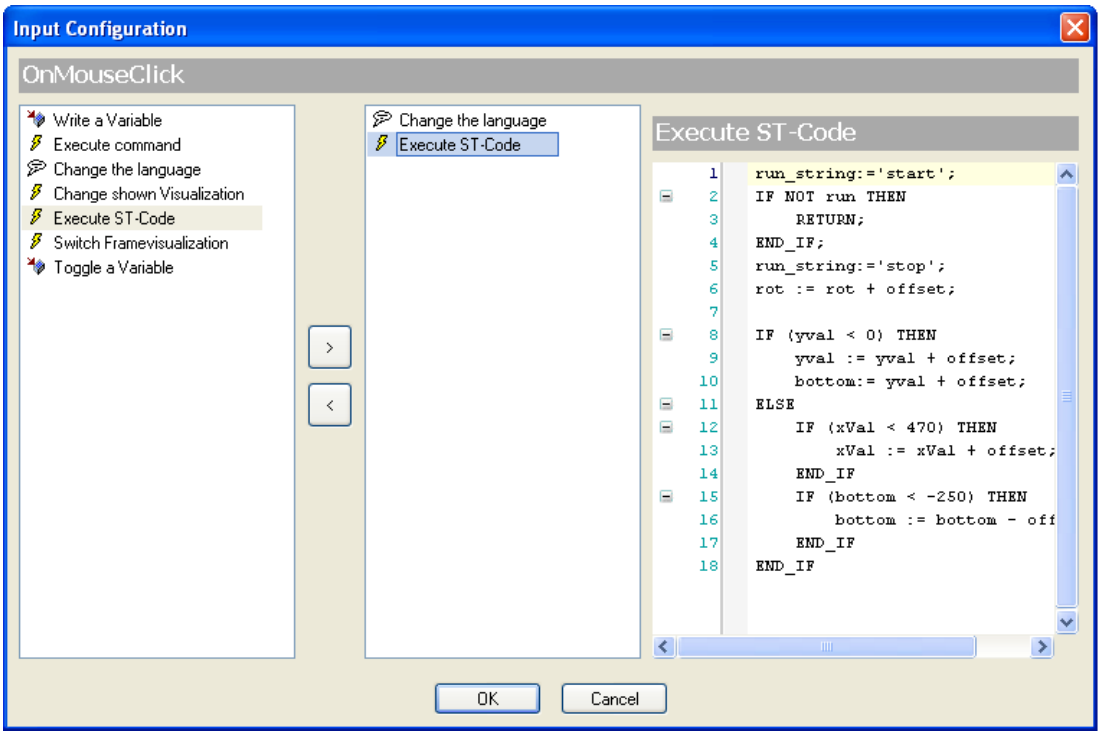


Abb. 449 Input Configuration, Execute ST-Code

In the editor field enter the code in Structured Text which should be executed on the mouse action in online mode.

- Switch Framevisualization

Preconditions: There is a visualization (or several) in your project, containing frame elements, which have got assigned several visualizations via the 'Frame Selection' command ('Frame Configuration' dialog). These visualizations get numbered within the frame by indices 0, 1, 2, etc. and by default in online mode the first of the assigned visualizations (index 0) will be displayed in the frame.

Switch Framevisualization:

In the current configuration dialog you can define, that - if the chosen mouse action is performed on the current visualization element in online mode - in a specified frame a definite one of its assigned visualizations should be displayed. So you can use the current visualization element as a switch for the display of the available visualizations in a frame.

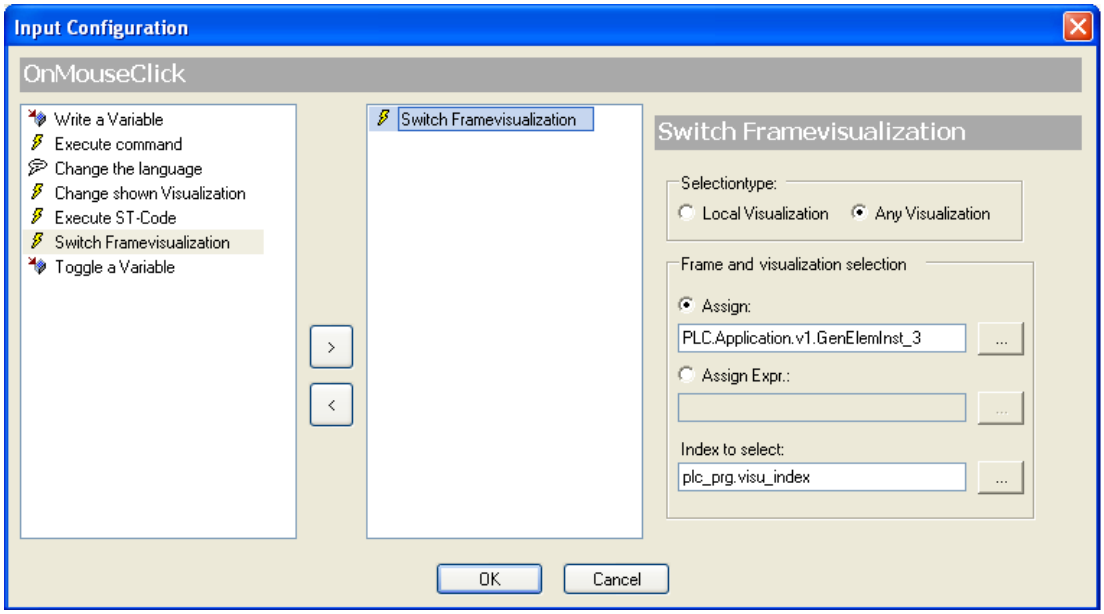


Abb. 450 Input Configuration, Switch Framevisualization

Selectiontype:

The selection of the frame element ...

1. ... can be limited to frames which are inserted in the current (local) visualization. This means getting a simplified configuration dialog without the possibility of dynamic configuration -> "Local Visualization".

2. ... can be extended to all visualizations available in the project, which also allows dynamic definitions of the frame and the visualization to be displayed -> "Any Visualization".

Local Visualization: Only frames of the currently edited visualization can be addressed and thus are available in the 'Frame selection' field.' This is a simple dialog for quick and direct configuration within the local visualization, however does not provide the possibility to define the desired visualization dynamically via a project variable. If dynamic definition is required, use option 'Any Visualization', see below.

The 'Frameselection' field shows the locally available frames and each indented below the assigned visualizations.

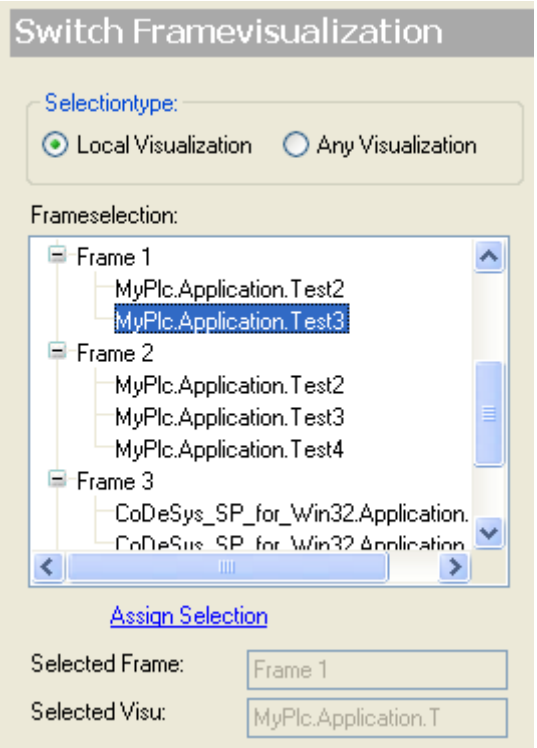


Abb. 451 Frame selection local

Below a frame entry select that visualization which should be displayed on the respective mouse action in online mode. Click on '**Assign Selection**' in order to store the selection. The current setting then will be displayed in the fields **Selected Frame** and **Selected Visu**.

Any Visualization: All frames + assigned visualizations, which are available in the whole project can be addressed in the current configuration. In this case the frame and the visualization also can be defined by a project variables, that is dynamically. The 'Frame and visualization selection' will be done via the following fields and options:

The desired frame element can be defined directly (Assign) or by a variable (Assign Expr.):

Assign: Direct assignment; if this option is activated, the complete path of the frame element must be entered. Via button ... the input assistant can be used.

Syntax: <device name>.<application name>.<visualization name>.<frame element name>

See an example in the following image.

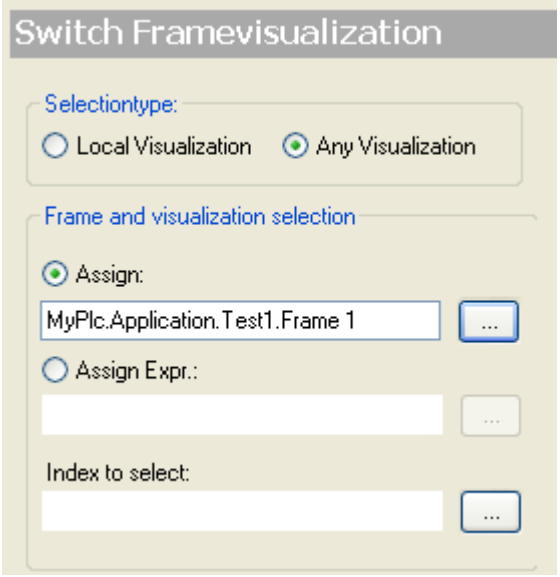


Abb. 452 Frame selection, direct assignment

Assign Expr.: This option can be used alternatively, if a project variable should dynamically define the frame element. The variable must be of type STRING. Via button the input assistant can be used. The variable must specify the complete path of the frame element.

Syntax: <device name>.<application name>.<visualization name>.<frame element name>.

Example: frame_var: STRING:='MyPlc.Application.Test1.Frame1';

The desired visualization, to be displayed on the mouse action in the above defined frame, is to be specified by its index number. This index results from an integer numbering of all visualizations assigned to a frame in the 'Frame Configuration' dialog. The index number of the first visualization in this list is "0". So initially by default this first visualization will be displayed in online mode.

Index to select: Enter here the index number of the desired visualization, either directly (for example "1") or via a variable used by the application. Via button ... the input assistant can be used.

- Toggle a Variable

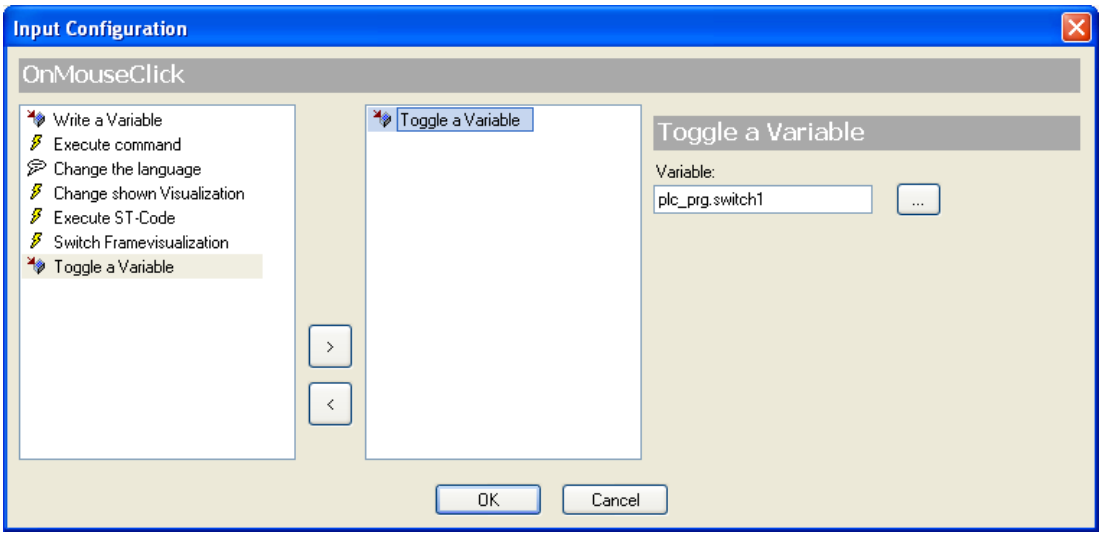


Abb. 453 Input Configuration, Toggle a Variable

Enter a boolean project variable, the value of which should be toggled between TRUE and FALSE by the mouse action. Example: "plc_prg.switch1".

Each mouse-input which has got assigned any action via the Input Configuration dialog, will get the property "Configured" in the Properties table.

Scrollbar specific entries (only for Scrollbar: Input Variable, Output Variable, Minimum value, Maximum value)

Input Variable	variable whose actual value determines the position of the actuator	Sc
Output Variable	variable whose value is determined by positioning the actuator	Sc
Minimum value	value corresponding to the most left position (horizontal scrollbar) resp. to the top position (vertical scrollbar) of the actuator	Sc
Maximum value	value corresponding to the most right position (horizontal scrollbar) resp. to the lowest position (vertical scrollbar) of the actuator	Sc

Defining color values:

A color is defined by a hex number which is composed of the Blue/Green/Red (RGB) components.

For each color value 256 (0-255) colors are available.

FF Blue component
00 Green component
FF Red component

Example: 16#FF00FF

For **static definitions** (category 'Colors' in the element properties) the color can be chosen from a selection list or in the color selection dialog (to be opened via button ...). Example: 0; 128; 64

For **dynamic definitions** (category 'Color variables') a project variable of type DWORD must be entered in the element properties which provides the color definition:
for example "prog1.dwFillColor".

The variable must define the color value in hexadecimal format. The first two zeros after "16#" should be set to in each case, to fill the DWORD size.
Example: dwFillColor := 16#FF00FF;

13.18.6 Visualization Interface Editor

This part of the Visualization editor can be switched on or off via command 'Interface Editor' (by default in the 'Visualization' menu). It will be provided in the upper part of the Visualization editor as a separate tab besides other additional editors like for example the Hotkeys Configuration editor or the Element List. editor.

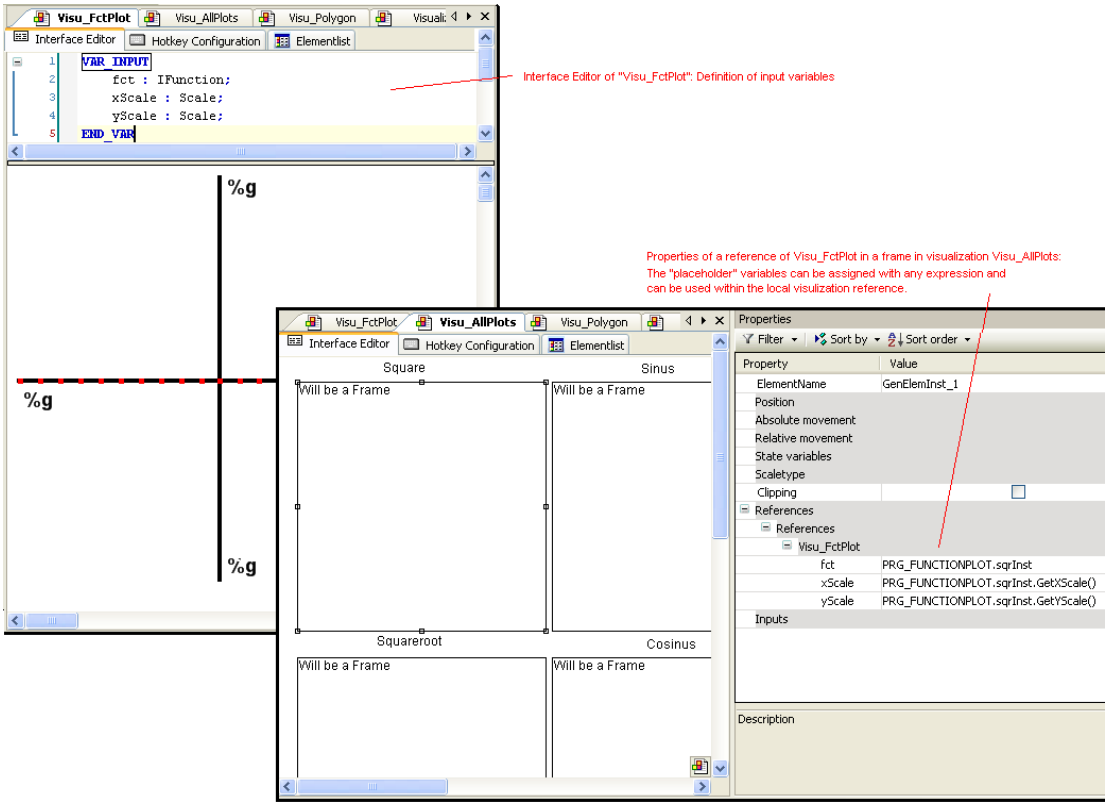


Abb. 454 Visualization Interface Editor, example

The editor is used for defining "**placeholder**" variables in a visualization which is intended to get referenced in a frame in another visualization.

See Frames, References, Placeholders... for information on references and frames.

See Visualization for the preconditions and an overview on the Visualization in »PLC Designer« V3.x.

Due to the fact that a visualization is handled as a function block, the Interface Editor will be added as a usual declaration editor in the upper part of the visualization editor window. There you can define input variables (VAR_INPUT), which later, when including the visualization object in a frame element of another visualization, can be replaced by locally used expressions. This replacement is to be done in the Visual Element Properties dialog of the frame element. The included (instantiated) visualizations and their input variables are listed there and can get assigned any expressions which then can be used in the configuration of the respective instance.

If the interface of a visualization which is referenced in a frame element of another interface, gets modified, the dialog **Updating the frame parameters** will be opened when you try to save or to compile the project or to open a concerned object. In this dialog you have to (re-)configure the interface parameters:

See in the following example, how to use this dialog:

The interface of visualization *basic_visu* initially only defines parameter *iVisu*. *basic_visu* is inserted in *visu2* as well as in *visu3* in a frame each, whereby in *visu2* parameter *iVisu* is assigned to project variable *PLC_PRG.ivar*. The project initially is used with this configuration. Later the inteface of *basic_visu* gets extended by the parameters *bToggleColor*, *strText* and *bVisu*:

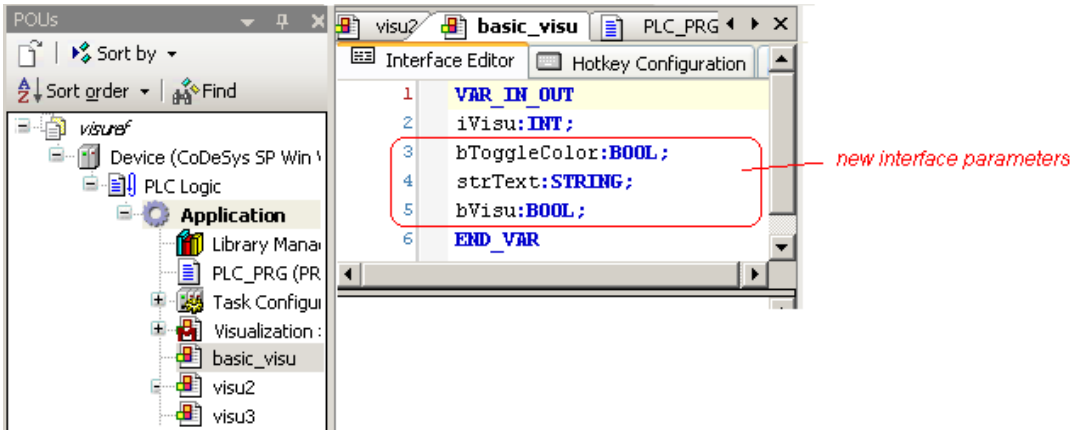


Abb. 455 Changes in Interface Editor

Now, if you for example try to compile or save the project, first the dialog for the configuration of the new parameters will appear:

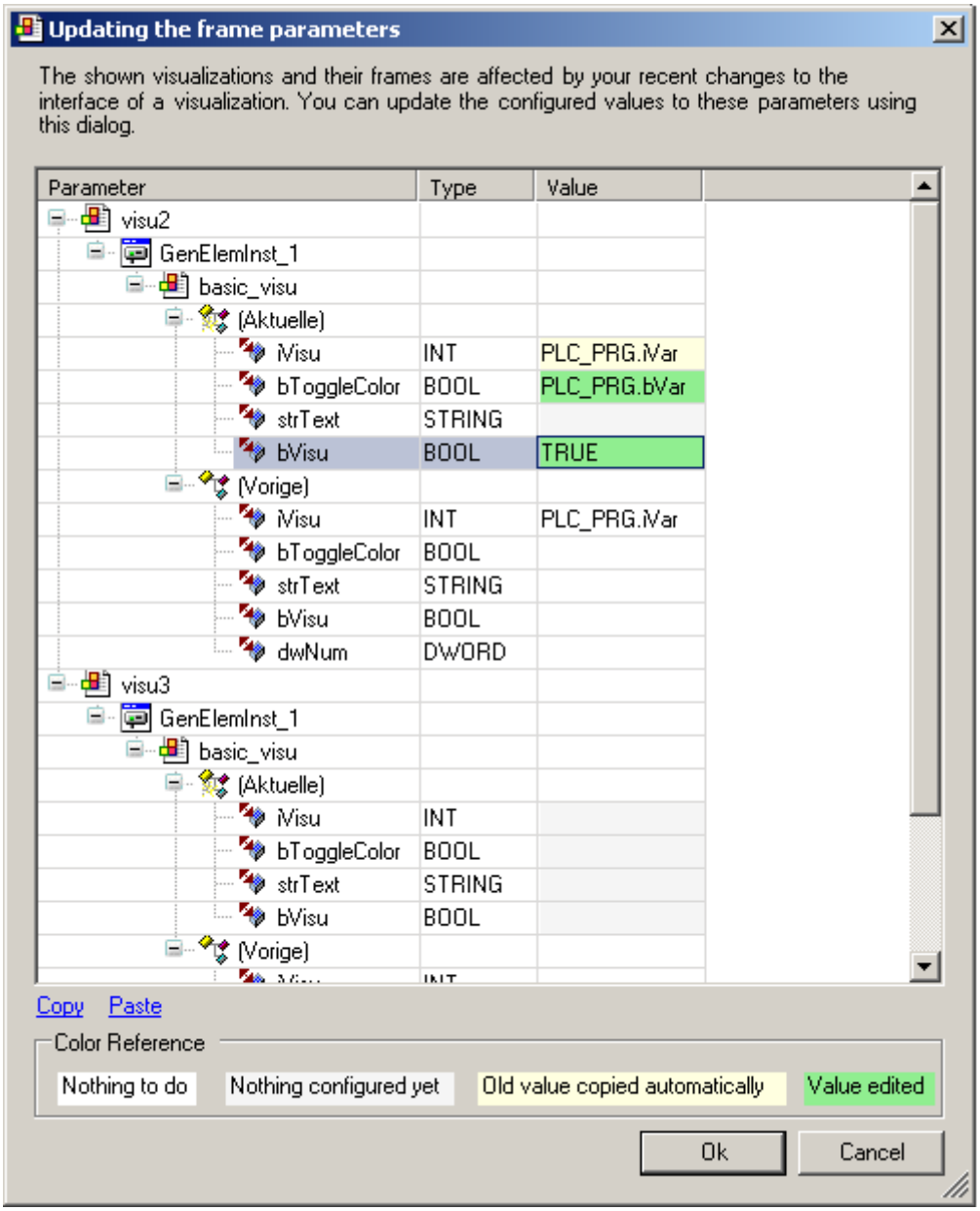


Abb. 456 Dialog for Updating the frame parameters

For all visualizations referencing the modified *basic_visu*, thus *visu2* and *visu3*, each the current and the previous values of the interface parameters are displayed. The current value can be (re-)configured here. The shading of the value-fields in the table indicates where still entries are required resp. have been done already (see also the legend in the bottom of the dialog):

beige shading: This parameter automatically has been copied from the previous configuration (for example *PLC_PRG.iVar* for *visu2*).

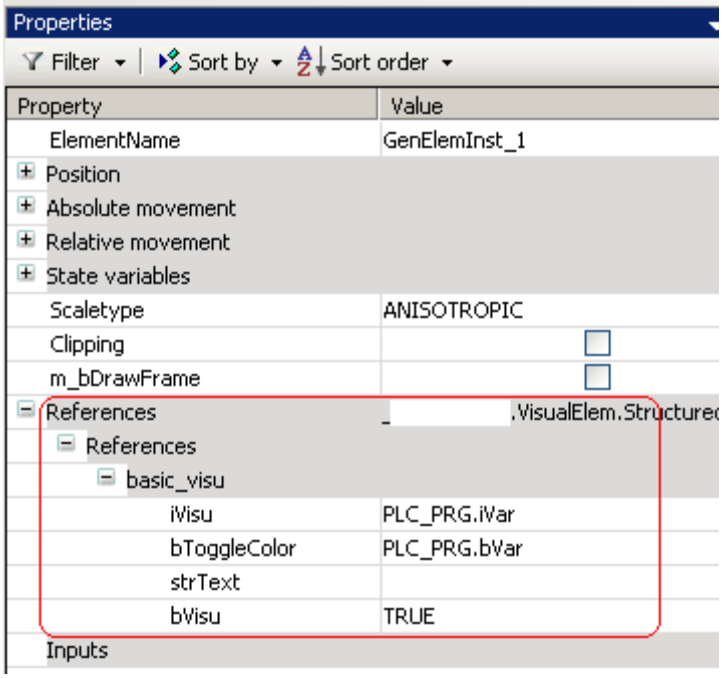
grey shading: Not yet any value has been assigned to this new parameter (for example *strText* for *visu2* and *visu3*).

green shading: To this new parameter already a value has been assigned (for example *bToggleColor* and *bVisu* for *visu2*).

white shading: Nothing must be entered here.

A value field can be edited after a mouse-click on the field and pressing the <Space>-key. An existing value-entry can be copied to another field. Select the field containing the entry to be copied and use button **Copy,** then select the field where the entry should be copied to and use button **Paste**.

After having confirmed the settings in the update dialog by OK the new parameter assignments will be displayed in the Properties dialog (category 'References') of the respective frame element, for example shown in the following for the frame in *visu2*:



If, in the referenced visualization (here "basic_visu"), you want to make use of the instance name of an input variable passed, you will have to work with the pragma attribute 'parameterstringof' providing a corresponding string. See attribute 'parameterstringof' for an example demonstrating all details.

13.18.7 Hotkeys Confguration Editor

In addition to the standard keyboard usage this part of the Visualization Editor allows to configure special hotkeys for operating a visualization by keyboard in online mode. That is, an action can be assigned to a certain key or key combination. This configuration is valid exactly for the current visualization. Key configurations which should be valid within all visualizations of an application, should be done in the visualization manager.

Regard in this context also the "Hotkey" property of a particular visualization element, which might be configured additionally, in the element Properties (Input). Such an element-specific key definition will also be represented in the Hotkeys Configuration Editor. It can be edited in both places whereby any modification always will get updated also in the other editor.

The Hotkey Configuration Editor - to be opened via the Hotkeys Configuration command - is available as a separate tab besides the Interface Editor and other editors in the upper part of the visualization editor.

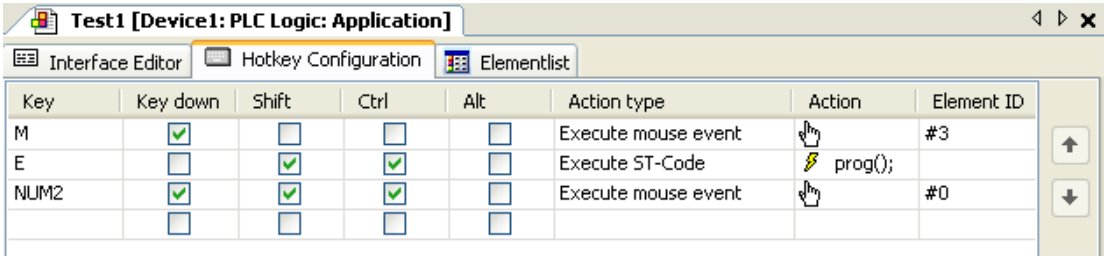


Abb. 457 Hotkeys Configuration Editor

In each row of the table editor a key or a combination of the key and one/several modifier keys can get assigned to an action. See the following columns:

Key: Name of the key. The key name can be entered manually or chosen from the selection list available on a double-clidk on the field. The list provides all keys defined by the **device description**, resp. in case of a visualization not directly assigned to a device all keys which are supported by each device.

Key down: If this option is activated, the Action will be executed, when the Key gets pressed. Otherwise the Action will be executed when the Key gets released. If the Action should be executed on Key down and Key up inputs, two appropriate definitions for the Key must be available in the table.

Shift, **Ctrl**, **Alt**: If this option is activated, the Shift resp. Ctrl resp. Alt key must be pressed together with the Key in order to get executed the Action.

Action type: The action type can be chosen from a selection list which is available on a double-click on the field. The action types correspond to those used in the input configuration dialog of a visualization element.

Action: Detailed configuration of the action to be executed; depends on the Action type and corresponds to the mouse action configurable in the input configuration dialog of a visualization element.

Element ID: ID of the visualization element the Key is assigned to (via the "Hotkey" Property). Unique identifier within the current visualization.

If a key is associated with several actions, these will be executed in the same order as they are listed top down here in the configuration table. You might change this order by selecting a key definition (click on the table row) and using the **arrow buttons** right to the table to move the entry up or down.

Note: It is defined by the device description, which keys are supported in a visualization running on that device!

Regard the following **sequence of calls** when processing key actions:

1. Eventhandler of the application, if activated (optional); e.g. g_VisuEventManager)
2. Key configuration valid for all visualizations of the application, in the visualization manager
3. Standard keyboard usage definition
4. Special key configurations of the particular visualizations (as handled by the Hotkeys Configuration editor), whereby main visualizations will be regarded before those which are contained in frames.

13.18.8 Element List Editor

This part of the Visualization Editor provides a list of the visualization elements of the currently opened visualization.

The Element List can be opened via the Element List command (per default in the Visualization menu) and then will be available as a separate tab besides the Interface Editor and other editors in the upper part of the visualization.

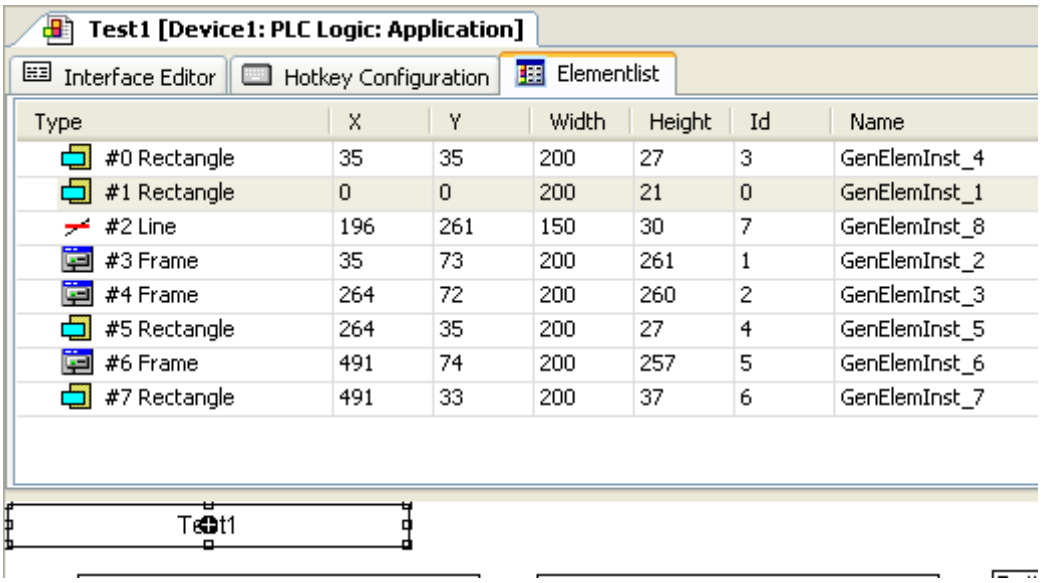


Abb. 458 Element List

The elements are listed top down according to their position on the Z-axis of the visualization area, the back-most in the first line. The following values are displayed, but cannot be edited here:

Type: Element type and icon as used in the toolbox, and element number initially resulting from the sequence of inserting

X, **Y**: Position of the upper left corner of the element; (0,0 = upper left corner of the visualization area)

ID: Internally assigned element identifier

Name. Element name as defined in the element Properties (Elementname)

Element(s) can be selected by selecting the respective row(s), whereby the selection always will be synchronized with that in the main visualization editor. Regard however that **subelements of a group** can only be selected here in the element list.

The position of a selected element on the Z-axis, that is on the **background-foreground** axis, can be modified by using the following commands:

- Bring to Front
- Bring one to Front
- Send to Back
- Send one to Back

The following commands can be used in order to **restore any edit actions** in the element list or to **remove** elements:

- Undo
- Redo
- Delete

13.19 Visualization Manager Editor

The Visualization Manager serves for handling common settings for all visualizations assigned to an application. After modifications the settings will be applied immediately.

The Visualization Manager object will be added automatically as soon as a visualization is inserted below an application in the device tree.

The main editor dialog contains separate tabs for the Settings and the Default Hotkeys configuration.

If supported by the device, additionally the client objects TargetVisualization respectively WebVisualization will be inserted in the device tree below the Visualization Manager, providing the settings for Target- resp. Web-Visualizations.

Regard that as from V 3.3.2.0 of the programming system the "Diagnosis Visualization" mode will be used automatically, if there are no client objects available below the Visualization Manager. For details please see: Visualization running in the programming system.

Note: If in a project created with a version < V3.3.2 a defined compiler version had been set, then the project will behave according to the visualization features available in that version!

In the following figure the Visualization Manager below 'Application' is responsible for visualizations PLC_VISU, PLC_VISU_1, Vis2 and Vis3. Target-Visualization is supported.

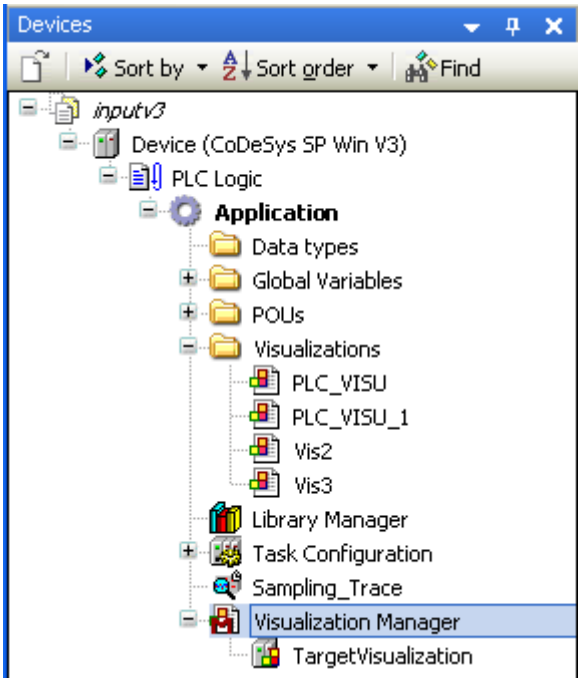


Abb. 459 Visualization Manager in device tree

To open the **Visualization Manager editor**, perform a double-click on the entry in the device tree or select the entry and use the 'Edit Object' command from the context menu. The editor opens in a window titled with "Visualization Manager <application path>, containing the subdialogs Settings and Default Hotkeys.

Settings

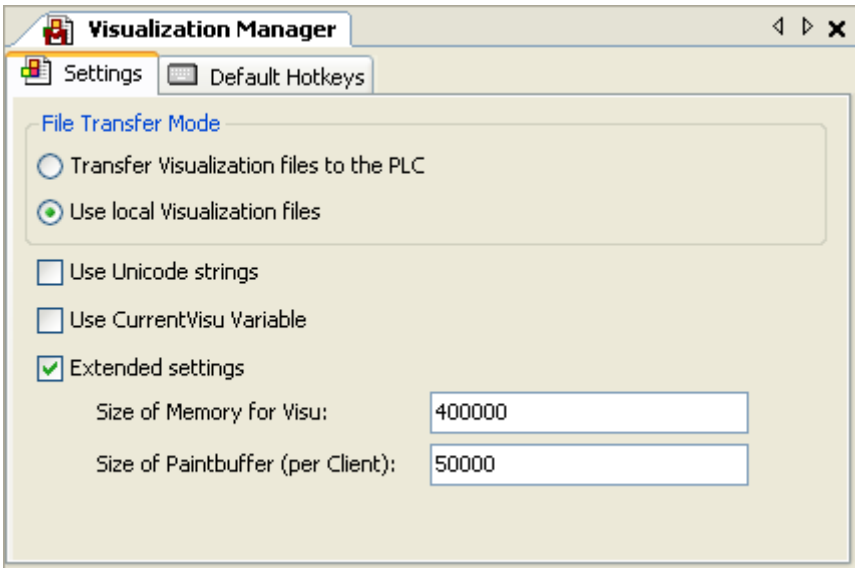


Abb. 460 Visualization Manager Editor, Settings

This tab of the Visualization Manager dialog contains settings valid for all visualizations assigned to the application.

Note: If the Diagnosis Visualization mode is used, the currently not available settings will be greyed.

File Transfer Mode:

Transfer Visualization Files to the PLC: If this option is activated, the files used for the visualizations (like image and textlist files) will be copied from the local file system (see the option described subsequently) to the PLC when the application is downloaded. For example this is needed for target visualizations, as well as for the default numpad and keypad visualizations which might be used in the input configuration of a visualization element.

Use local Visualization files: The textlist and image files used by the visualizations are read from the local file directory, which is defined in the Project Settings, category Visualization.

Use Unicode Strings: If this option is activated, all strings used in the visualizations will be processed in Unicode format.

Use CurrentVisu Variable: If you activate this option, the global string variable "CurrentVisu" will be taken into account. Thereby, you may control the visualization to be displayed within the program by setting this variable to the name of the desired visualization.

Extended settings: Activate this option if you must modify the following settings, which usually is not necessary for standard visualization applications:

Size of Memory Visu: memory space in Bytes, allocated for visualization, default: 400000

Size of Paintbuffer (per Client): memory buffer in Bytes, allocated for painting actions on the client, default: 50000

Default Hotkeys

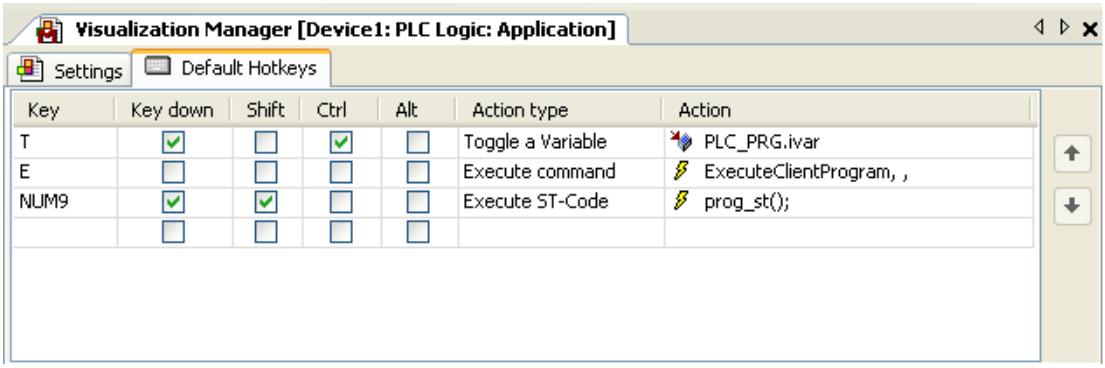


Abb. 461 Visualization Manager Editor, Default Hotkeys

This tab of the Visualization Manager dialog contains the key configurations valid for all visualizations assigned to the application. That is that here shortcuts are defined which - if supported by the respective device - can be used for user inputs on the visualizations in online mode via the keyboard. The usage of the configurator is the same as of that integrated in the visualization editor for a particular visualization, so please see the Hotkeys Configuration Editor for a description.

Additionally always some Standard Shortcuts for navigation within a visualization are supported device-independently.

13.19.1 Target-Visualization Editor

The **TargetVisualization** object inserted below the Visualization Manager object contains the settings for the Target-Visualization. They can be modified in the editor window which opens on a double-click on the object.

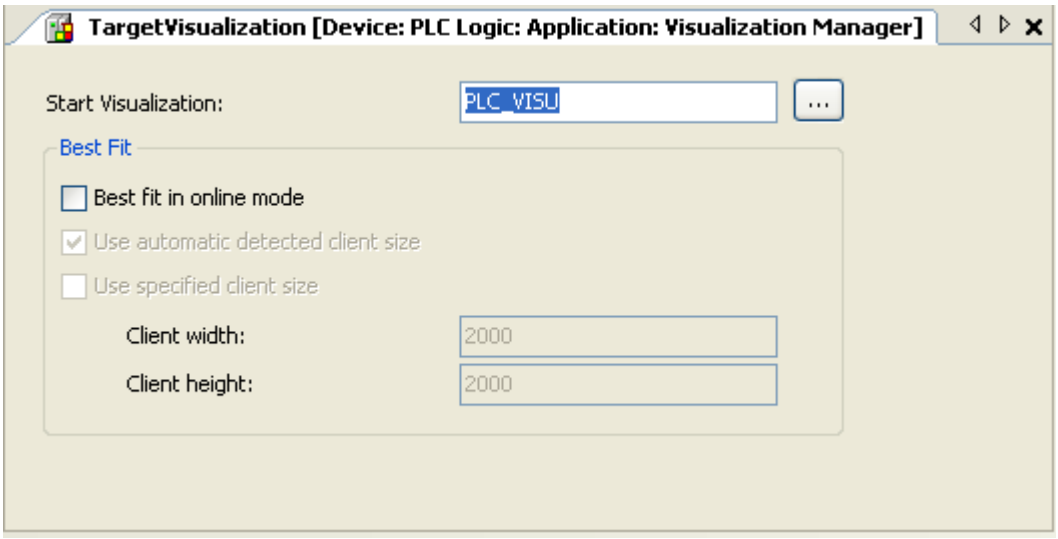


Abb. 462 Target-Visualization Editor

Note the preconditions for running a Target-Visualization described on the Target-Visualization help page.

Start Visualization: Name of the visualization to be displayed automatically as target visualization, when the target system starts running. By default "Visualization" is entered here. To set another visualization you might use the Input Assistant offering the currently available visualizations.

Best Fit:

Best fit in online mode: If this option is not activated, the visualization will be displayed on the target visualization screen in exactly that size that it is defined in the visualization object properties. If the option is activated, the target visualization display will be adapted according to the following settings:

Use automatic detected client size: The target visualization will fill the client screen.

Use specified client size: The visualization will fill the screen area defined by:

Client width: Width in pixels

Client height: Height in pixels

13.19.2 Web-Visualization Editor

The **WebVisualization** object inserted below the Visualization Manager object contains the settings for the Web-Visualization. These can be modified in the editor window which opens on a double-click on the object:

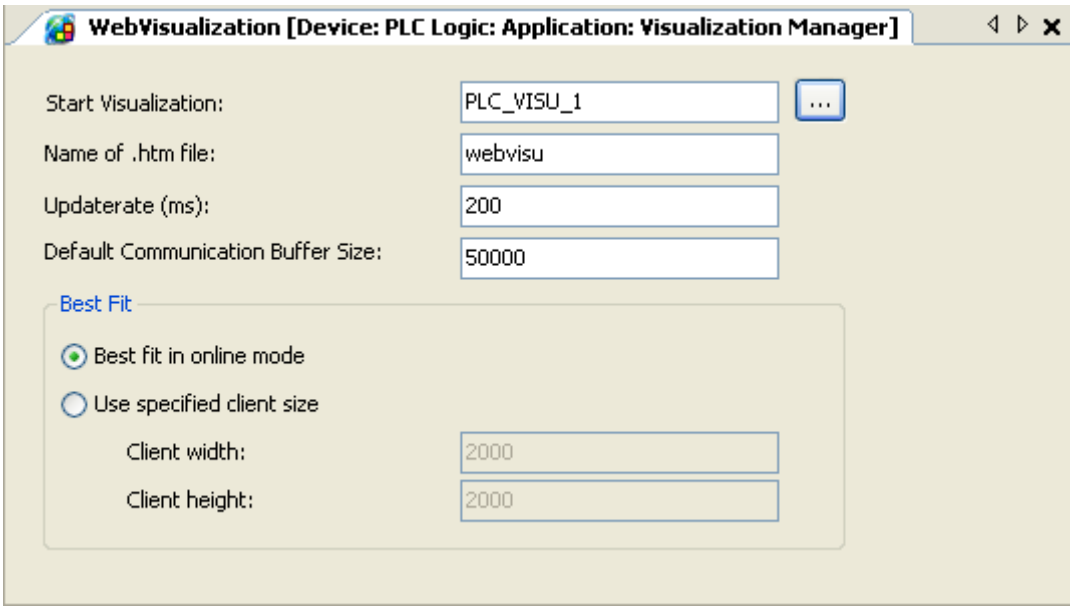


Abb. 463 Web-Visualization Editor

For some basics and the preconditions for running a Web-Visualization see the Web-Visualization help page.

Start Visualization: Name of the visualization to be displayed automatically when the Web-Visualization gets started. By default "Visualization" is entered here. To set another visualization you might use the Input Assistant offering the currently available visualizations.

Name of .htm file: Name of the basic HTML-page of the visualizaton. Must also be entered as address in the web browser; example: http://localhost:8080/webvisu.htm

Updaterate (ms): Update rate within the web browser, in milliseconds

Default Communication Buffer Size: in Bytes, defines the maximum available buffer for the data transfer between web client and web browser.

Best Fit:

Best fit in online mode: If this option is activated, the size of the visualization will be adapted to the size of the web-browser window.

Use specified client size: If this option is activated, the size of the visualization will be defined by the following settings:

Client width: Width in pixels

Client height: Height in pixels

13.20 Watch List Editor

The watch list functionality is provided by the "Watch List" plug-in.

A watch view can be opened via the Watch command submenu (by default in the View menu). It provides an editor for creating watch lists.

A watch list can be used to define and monitor a list of expressions of various objects and to write or force values to those expressions on the controller in online mode. By default four individual watch lists can be set up in the watch views **Watch 1**, **Watch 2**, **Watch 3**, **Watch 4**. View **Watch all Forces** in online mode always gets filled automatically with all currently forced values of the active application.

13.20.1 Create watch list

To set up a **Watch <n>** list in the watch view perform a mouse-click in a field of the **Expression** column to open an edit frame. Enter the complete path for the desired watch expression:

Syntax for watch expression:

<device name>.<application name>.<object name>.<variable name>,

Example: "Dev1.App1.PLC_PRG.ivar" (as shown in the image below)

The type of the variable will be indicated by an icon: = input, = output, = "normal" variable. If a comment has been added to the declaration of a variable, it will be displaced in column **Comment**.

After having closed the edit frame the type will be added automatically in column **Type**.

The **Value** column will be used in online mode for displaying the current value of the expression.

To appoint a prepared value to a variable you may click in the assigned field of the column **Prepared value** and directly type in the desired value.
In case of a boolean variable the handling is even easier: You can toggle boolean preparation values by use of the **Return** or **Space** key according to the following order: If the value is TRUE, the preparation steps are FALSE -> TRUE -> no entry; else, if the value is FALSE, the preparation steps are TRUE -> FALSE -> no entry.

Do the same for the desired further expressions/variables in further lines. See an example in the next picture which shows the watch view in offline mode: It contains expressions of objects PLC_PRG and Prog_St. Notice that in case of a structured variable, like here the function block instance, the particular instance components automatically get added when you enter the instance name (see in the example: "Dev1.App1.PLC_PRG.fbinst"). They can be displayed resp. hidden in a **fold** via a mouse action on the plus-/minus sign.

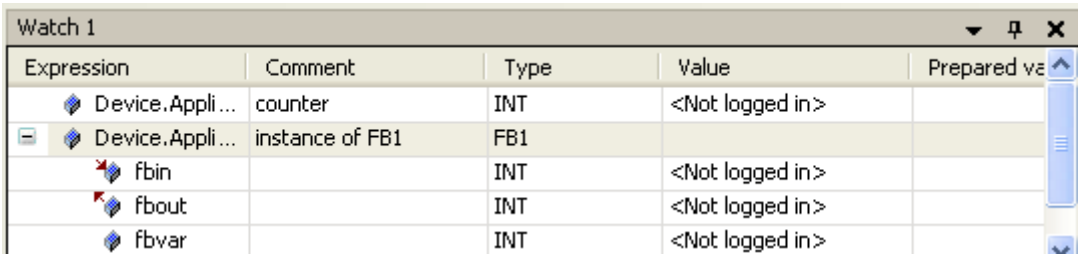


Abb. 464 Example, Watch view in offline mode

In online mode the view can be used for monitoring.

13.20.2 Watch list in online mode

Monitoring:

A watch list "Watch <n>" in online mode shows the current value of a variable in the **Value** column.

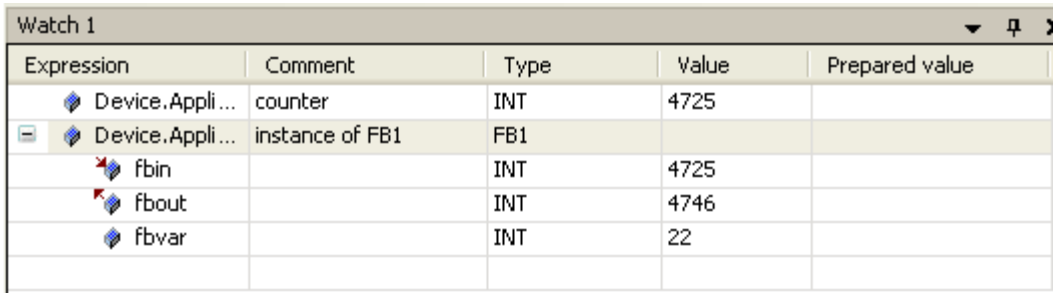


Abb. 465 Example, Watch view in online mode

Write values:

In column **Prepared value** you can enter a desired value which will be written or forced to the respective expression on the controller by command 'Write values' or 'Force values' like usable in other monitoring views (for example declaration editor).

Watch all Forces:

This is a special watch list view, which in online mode gets automatically filled with all currently forced values of the active application. Each Expression, Type, Value and Prepared value will be shown, like in the online view of an 'Watch <n>' list.

You can unforce values by one of the following commands available via the **Unforce...** button:

- **Unforce all selected Expressions, without modifying the value.**
- **Unforce all selected Expressions and restore the variable to the value it had before forcing it.**

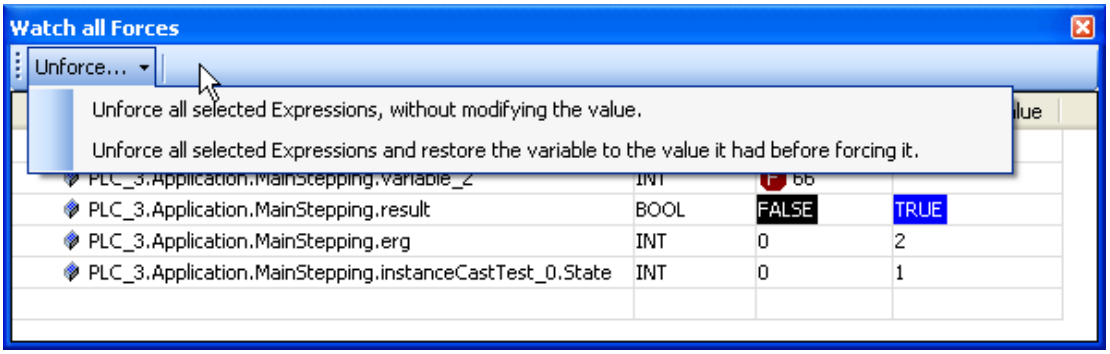


Abb. 466 Example, Watch all Forces

14 Device Editors

14.1 Generic Device Editor

The Device Editor provides dialogs for the configuration of a device which is managed in the Devices view window.

The editor can be opened via command 'Edit Object' or by a double-click on the device object entry in the Devices window.

Notice the Device Editor Options for general settings for the editor. For example generic dialogs might be set invisible.

By default the **Device dialog** can be opened by a double-click or by using command 'Edit object' when the device entry is selected in the devices tree.

The main dialog is entitled with the device name

and, depending on the device type, can provide tabs containing the following sub-dialogs:

- Communication Settings: Configuration of the connection between programming system and a programmable device (PLC). Not available for pure I/O-devices.
- Configuration: Display resp. configuration of the device parameters.
- Applications: Shows the applications currently running on the PLC and allows to remove applications from there.
- Files: Configuration of a file transfer between host and PLC.
- Log: Display of the PLC log file.
- PLC settings: Configuration of: application noticed for I/O handling, I/O behaviour in stop status, bus cycle options
- I/O Mapping: Mapping of the input and output channels of an I/O device on project (application) variables.
- Users and Groups: User management concerning device access during runtime (not to be mixed up with the project user management)
- Access Rights: Configuration of the access rights on runtime objects and files for the particular user groups
- Status: Device-specific status and diagnosis messages.
- Information: General information on the device (name, provider, version etc.)
- Bus-specific configuration dialogs, for example:
  - DP Profibus configuration
  - Ethercat configuration

Note: The 'Communication Settings' and 'Applications' tabs concern the communication with the PLC and are provided by the 'Device Communication Editor' plug-in.

Communication Settings

This dialog is provided on a tab of the **Device** dialog (Device Editor), which can be opened via command 'Edit Object' or by a double-click on the device object entry in the Devices window. It serves to configure the communication parameters for the device.

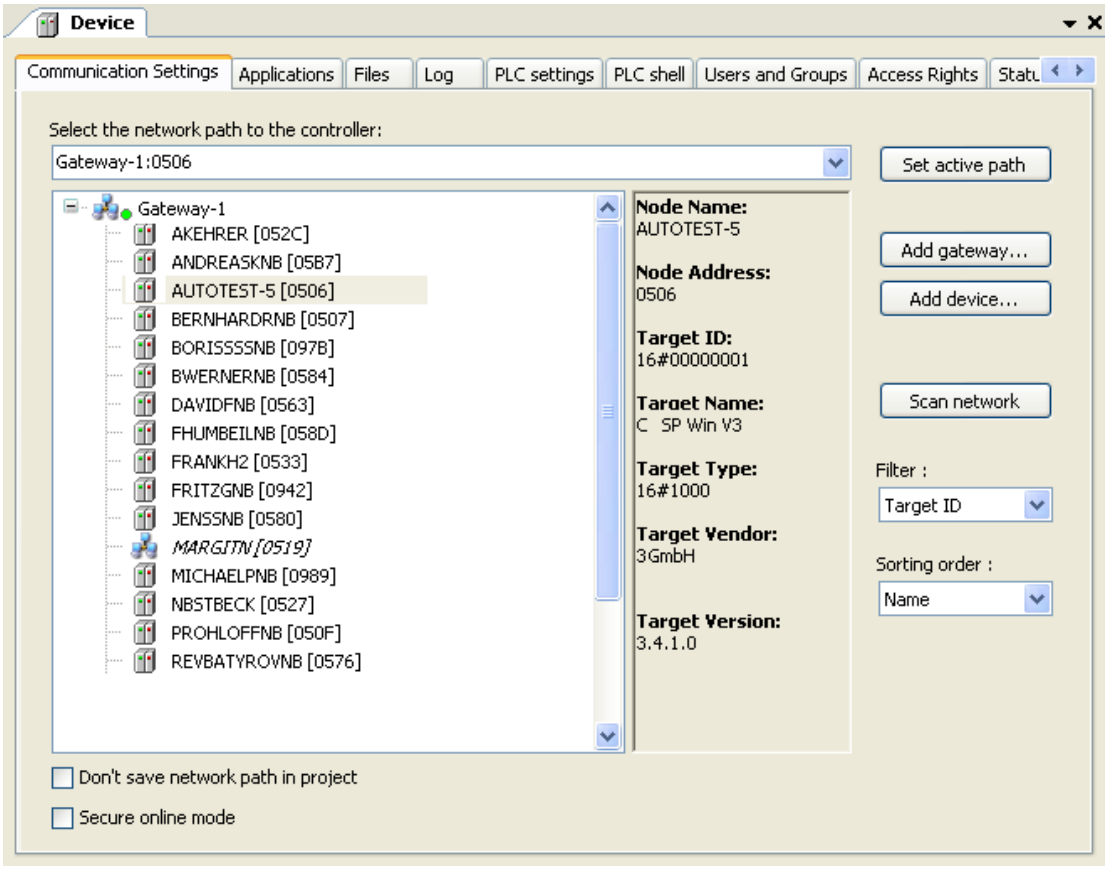


Abb. 467 Device dialog, Communication Settings

The dialog contains a **bipartite window**, which in the left part shows in a **tree structure** the currently configured gateway channels and in the right part the corresponding data and information.

When creating the first project at your local system, at least one entry will be in the configuration tree: The local gateway (which is started at system start automatically):

»PLC Designer« Gateway service:
The »PLC Designer« Gateway Server is started automatically at system start as a service and this service is represented by a control icon in the system bar:
Icon indicates a running, icon indicates a stopped Gateway. You can stop and restart the Gateway via the commands of the menu which is accessible by a click with the right mouse-button on this icon. If necessary the service also can be started in the Programs menu by selecting the "Gateway" entry.

The settings of this gateway are displayed in the right part of the window (see also below, 'Gateway' dialog): Example:

Node Name: "Gateway-1"
Driver: TCP/IP
IP-Address: 127.0.0.1
Port: 1217

As long as the gateway is properly running, a green bullet is displayed before the entry, otherwise a red one . The bullet stays grey, if the gateway has not been contacted yet (depending on some communication protocols, it is not allowed to poll the gateway, so the status cannot be displayed).

Indented below the gateway entry (open/close via the +/- sign) you will see entries for all devices which are reachable through this gateway: Regard that those entries are stored locally on your system and not with the project. The device entries are preceded by a symbol. Entries with a target ID different to that of the device currently configured in the project, are displayed in grey font. To get an up-to-date list of the currently available devices use button or command 'Scan network' (see below).

The device entries consist of a symbol followed by "<**Node Name**> [**Node Address**]" and in the right window part you additionally see the respective **Target ID, Target Name, Target Type, Target Vendor** and **Target Version**.

The display of the gateway and devices nodes' tree can be controlled by filter and sorting functions. See the selection boxes in the right part of the dialog:

- **Filter:** The list can be reduced to all devices with a **Target ID** matching with that of the device currently configured in the project
- **Sorting:** The list can be sorted according to the **Name** resp. **Node Address** in alphabetical resp. ascending order.

The currently selected gateway channel always will be displayed in the '**Select the network path to the controller**' field.

To modify the communication configuration use the following buttons respectively commands from the context menu:

Set active path	This command sets the currently selected communication channel as the active one. See "Set active path". A doubleclick on the entry in the channel tree effects the same.
Add gateway...	This command opens the "Gateway" dialog, where you can define a gateway to be added to the current configuration. See "Add Gateway"
Add device...	This command opens a dialog titled "Device", where you can manually define a device to be added to the currently selected gateway entry (Regard also the 'Scan network' functionality). See "Add Device..."
Edit Gateway...	This command opens the "Gateway" dialog for editing the settings of the currently selected gateway. See "Edit Gateway"
Delete selected Device	This command removes the selected device from the configuration tree. See "Delete selected Device".
Scan for Device by Name	This command serves to scan the network for devices of the same name as specified in the "Communication Settings" devices list below the corresponding gateway entry (case sensitive search!). If any are found, each of them will be added to the list with its specific node address, i.e. the entry will be displayed with name + node address. See "Scan for Device by Name"
Scan for Device by Address	This Command serves to scan the network for devices with the same node address as specified in the "Communication Settings" devices list below the corresponding gateway entry. If any are found, each of them will be added to the list with its name, i. e. the entry will be displayed with name + node address. See "Scan for Device by Address"
Connect to local Gateway	This command opens a dialog for the configuration of a local gateway and therefore provide an alternative to manual editing the file "Gateway.cfg". See "Connect to local Gateway"
Scan network	This command starts a search for available devices in your local network and the configuration tree of the concerned gateway will be updated accordingly. See "Scan network"

Secure online mode: Activate this option if the user for security reasons should be prompted for confirmation when selecting one of the following online commands: Force values, Multiple download, Release force list for <application>, Single cycle, Start <application>, Stop <application>, Write values.

Don't save network path in project: Activate this option, if the current network path definition should not be stored in the project, but in the local option settings on your computer. So the path setting will be restored if the project is reopened on the same computer, but will have to be redefined if the project is used on another system.

Configuration

If option 'Show generic device configuration views' is activated, the Configuration dialog will be available on a tab of the **Device** dialog (Device Editor). The Device editor can be opened for the device currently selected in the devices tree via command 'Edit Object' or by a double-click on the device entry.

The tab is entitled with "**<module type> Configuration**" (for example "DP-Module Configuration") and shows the properties of device specific parameters and - if allowed by the device description - provides the possibility to **edit** the parameter values.

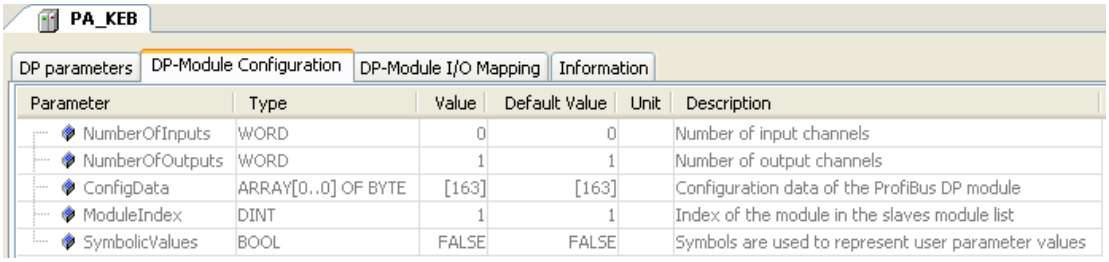


Abb. 468 Device dialog, Configuration

The following columns are available to describe a parameter:

Parameter: Parameter name, not editable.

Type: Data type of parameter, not editable.

Value: Primarily the default value of the parameter is displayed, directly or by a symbolic name. If the parameter can be modified (this depends on the definition given by the device, non-editable parameters are displayed grey-colored), an **edit** frame an be opened by a mouse-click on the table field resp. a selection list can be used to change the value. If the value is a file specification, the standard dialog for opening a file can be opened by a double-click on the entry and another file can be selected.

Default Value: Default parameter value, not editable.

Unit: Unit of the parameter value, for example "ms" for Milliseconds, not editable.

Description: Short description of the parameter, not editable.

Applications

This dialog is provided on a tab of the **Device** dialog (Device Editor), which can be opened via command 'Edit Object' or by a double-click on the device object entry in the Devices window. It serves to scan and to remove applications on the PLC.

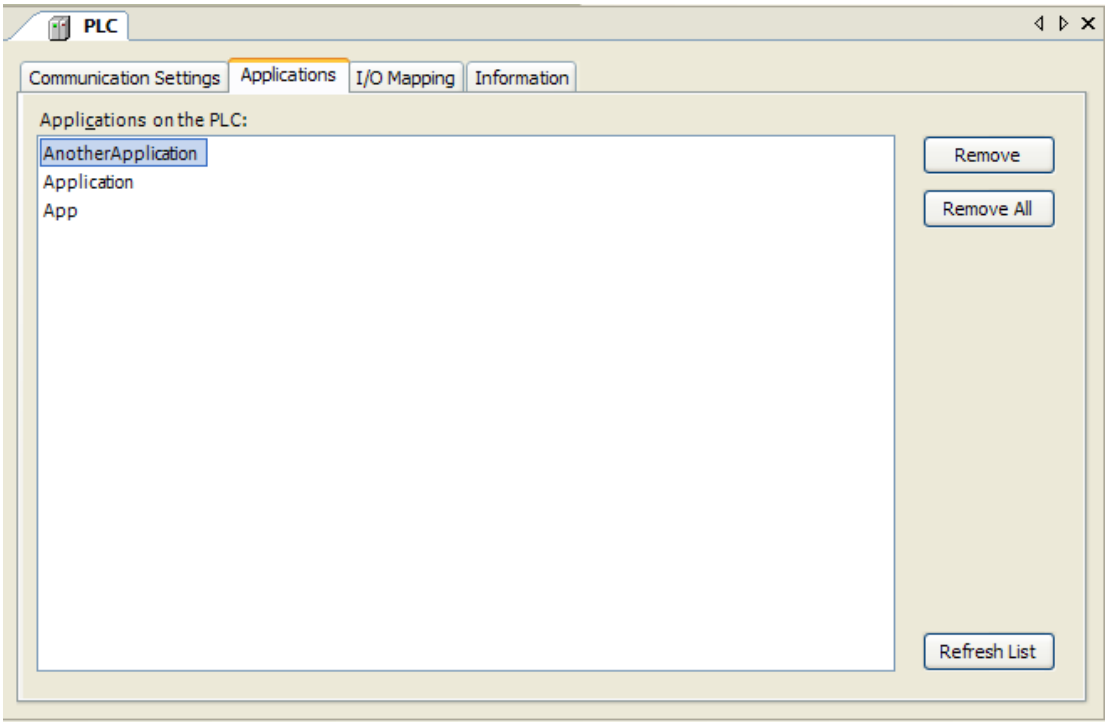


Abb. 469 Device dialog, Application

Applications on the PLC: List of the names of applications which have been found on the PLC during the last scan.

Refresh List: The PLC will be scanned for applications, the list will be updated.

Remove resp. **Remove All**: The application currently selected in the list resp. all applications will be removed from the PLC.

When an application gets loaded to the PLC, it will be checked whether the list of applications on the PLC matches the list of applications in the devices tree and also whether the parameters of an application on the PLC match the parameters of the application in the device configuration ("signatures"). If inconsistencies are detected, an appropriate message ("unresolved reference(s)" resp. "signature mismatch(es)") will be generated in a message box as well as in the message view in category "Download".

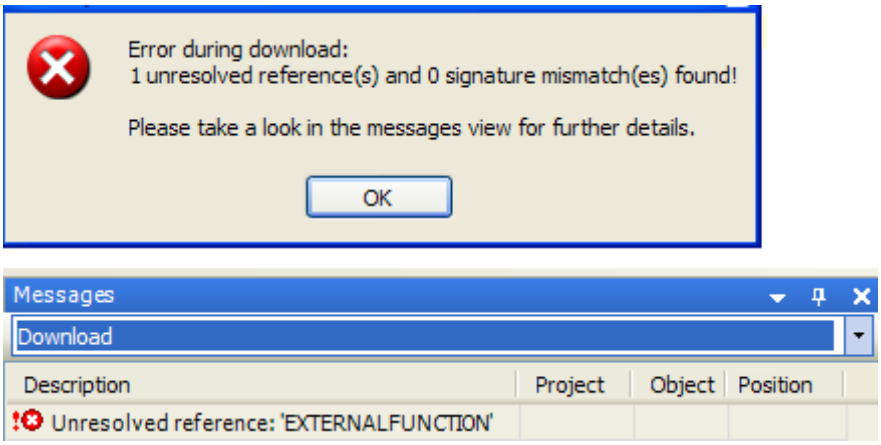


Abb. 470 Message about unresolved references, example

If multiple applications are directly assigned to the device notice the actual setting concerning I/O mapping.

See also Application for general information on handling application projects.

Files

This dialog is provided on a tab of the **Device** dialog (Device Editor), which can be opened for the device currently selected in the devices tree via command 'Edit Object' or by a double-click on the device entry in the Devices window.

The dialog serves to transfer files between the host and the controller. This means you can choose any file from a directory of the local network to get it copied to the files directory of the currently connected runtime system, or the other way round.

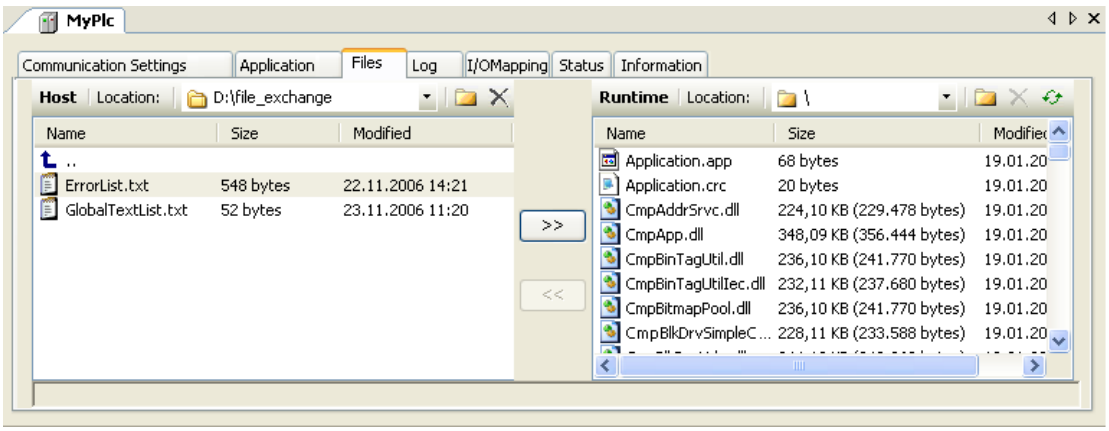


Abb. 471 Device dialog, Files

In the left part of the dialog the files on the **Host** are displayed, in the right part that on the **Runtime** system.

In order to update the Runtime file list, use button .

If you want to create a new folder where the file should be copied, use button .

If you want to delete the currently selected file, use button .

In the **Location** selection fields of Host and Runtime each enter the appropriate directories between which one or several files should be transferred. This is to be done via the selection list at the entry field or via browsing in the file system tree.

The files to be copied have to be selected in the file system tree. Multiple selection is possible, also a folder can be selected in order to get copied all contained files.

For transferring the selected files to the defined host or runtime system directory, use button >> resp. <<. To "transfer" in this context means to "copy". So, if a file is not yet available in the "target" directory, it will be created also there. If however already a file with the given name is available and is not write-protected, this will be overwritten. In case of write-protection an appropriate message will be generated.

Log

This dialog is provided on a tab of the **Device** dialog (Device Editor) and serves to view the events, which have been logged on the runtime system, in the programming system.

This concerns:

- Events at system start or shutdown (loaded components and their versions)
- Application download and boot project download
- Customer specific entries
- Log entries of I/O drivers
- Log entries of the Data Server

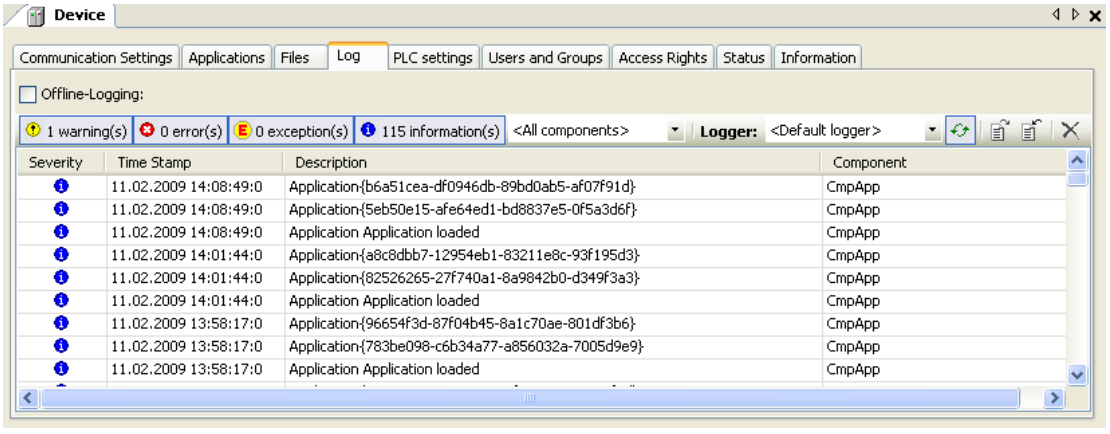


Abb. 472 Device dialog, Log

A log entry line contains the following information:

Severity: There are four categories: warnings, errors, exceptions, information. The display of the entries of each category can be switched on or off by using the corresponding button from the bar above the listing. Each button always contains the current number of loggings in the respective category:

Time Stamp: Date and Time for example "12.01.2007 09:48"

Description: description of the event, for example "Import function failed of <CmpFileTransfer>"

Component: ID and name of the component

Selection list of component names: Here you can choose a particular component in order to get only displayed log entries concerning this component. The default setting is "<All components>".

Logger: The selection list provides the available loggings. The default setting is "<Default Logger>", which is defined by the runtime system and currently is identical with "PlcLog" for runtime system »PLC Designer« SP.

not yet implemented: The log list will be updated automatically by the runtime system.

Currently the user manually can update the list by using button .

The list can be **exported to or imported from a XML-file**. To export use button to get the standard dialog for saving a file. The file filter is set to "xml-files (*.xml)". The log-file will be stored with the specified file name with extension ".xml" in the chosen directory. To view log entries stored in an xml-file which might have been exported in this way, use button . The standard dialog for browsing for a file will open. Also here the filter is set to "xml-files (*.xml)". Choose the desired log-file and its entries will be displayed in a separate window.

For clearing the current log table, that is removing all displayed entries, use button .

If option **Offline-Logging** is activated, also certain offline actions will be logged, which do not refer to the connection with the PLC. This however is currently only realized for the »PLC Designer« Safety version of the programming system.

PLC settings

This dialog is provided on a tab of the **Device** dialog (Device Editor), which can be opened via command 'Edit Object' or by a double-click on the device object entry in the Devices window. It serves to configure some general settings for the device (PLC).

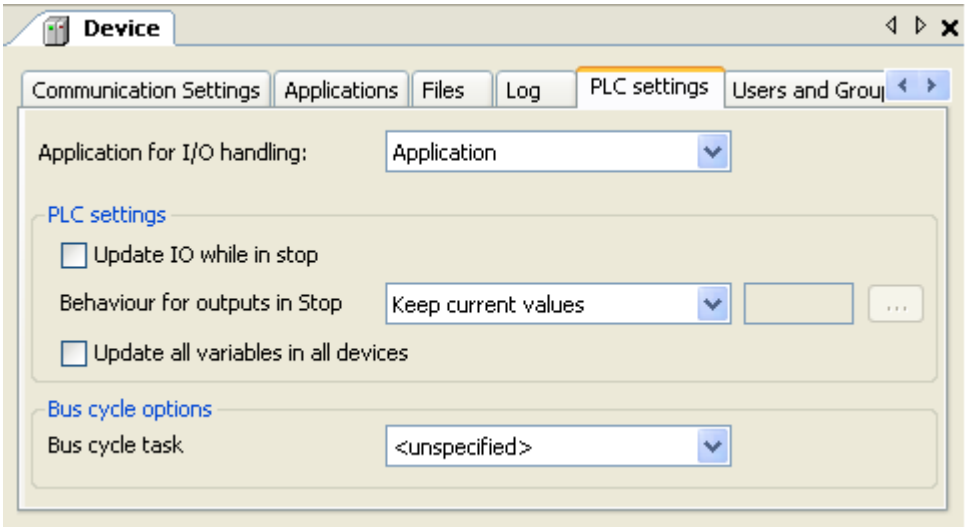


Abb. 473 Device dialog, PLC settings

Application for I/O handling: Define here, which of the applications assigned to the device in the device tree should be noticeed for the I/O-mapping. Initially the standard application, which is always inserted automatically in a standard project, will be set, but you may choose another one from the selection list.

PLC settings:

Update IO while in stop: If this option is activated (default), the values of the input and output channels get also updated when the PLC is stopped. In case of expiration of the watchdog, the outputs are set to the defined default values.

Behaviour for outputs in Stop: From the selection list choose one of the following options, how the values at the output channels should be handled in case of PLC stop:

Keep current values: The current values will not be modified.

Set all outputs to default: The default values resulting from the mapping will be assigned.

Execute program: You might determine the outputs behaviour by a program available within the project. Enter the name of this program here and it will be executed when the PLC gets stopped. Via button [...] the Input Assistant can be used for this purpose.

Update all variables in all devices: If this option is activated, then for all devices of the current PLC configuration all I/O variables will get updated in each cycle of the bus cycle task. This corresponds to option Always update variables, which can be set separately for each device in the "I/O Mapping" dialog.

Bus cycle options:

Bus cycle task: The selection list offers all tasks currently defined in the Task Configuration of the active application (for example "MainTask", "EtherCAT Master" etc. The default setting as defined by the device description is automatically entered. You might switch to another setting, but **absolutely regard the following:**

Note: Before modifying the setting to "<unspecified>", make sure to know about the effect. "<unspecified>" means that the basic default behaviour as defined in the device description will be applied. So check this description in this respect: As default the task with the shortest cycle time might be defined, but just as well it could be that with the longest cycle time!

Additional settings:

Generate force variables for IO mapping: This setting is only available if supported by the device. If the option is activated, for each IO channel, which has got assigned to a variable in the I/O Mapping dialog, two global variables will be created as soon as the application gets built. These variables can be used e.g. in a HMI visualization for forcing the I/O value. See I/O Mapping for detailed information.

I/O Mapping

This tab of the **Device** configuration dialog (Device Editor) is entitled with "**<device type> I/O Mapping**" (for example "DP-Module I/O Mapping"). It serves to map the PLC device I/Os on project variables, that is to assign the addresses of the device input, output and memory locations to variables used in the application program.

The application which should handle the I/Os, must be defined in the PLC settings dialog.

See in the following:

- General information on I/O mapping
- Using the I/O Mapping dialog
- Implicit variables for forcing I/Os

General on I/O mapping

Whether at all an I/O mapping can be configured for the current device, depends on the device; it might be that the dialog only is used to show the implicitly created device instance, see below: "IEC Objects".

Basically the following must be noticed for the mapping of I/Os on variables:

- Variables requiring an input cannot be accessed by writing.
- Instead of using the I/O Mapping dialog an address also can be assigned to a variable via the "AT declaration". On this regard the following: 1. "AT declarations" only can be used with local or global variables, not however with input and output variables of POUs. 2. The possibility of generating force variables for I/Os (see below: Implicit variables for forcing I/Os) will not be available for "AT declarations". 3. If "AT declarations" are used with structure or function block members, all instances will access the same memory location, which corresponds to "static variables" in classic programming languages like for example C.
- The memory layout of structures is determined by the target device.
- For each variable which is assigned to an I/O channel in the I/O Mapping dialog, "force variables" can be created during a build run of the application. These might be used for forcing the input resp. output value during the commissioning of a machine, e.g. via a visualization (HMI). See below: Implicit variables for forcing I/Os.

Using the dialog

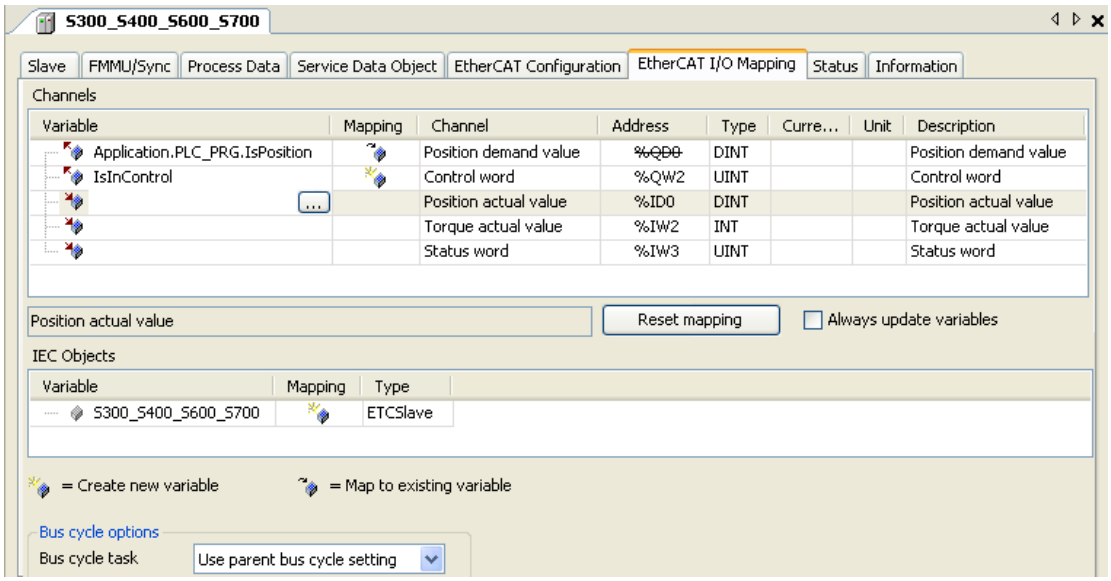


Abb. 474 Device dialog, I/O Mapping, Example

Channels:

The following entries will be displayed if provided by the device:

Channel: Symbolic name of the input resp. output channel of the device

Address: Address of the channel, for example "%IW0".

You can modify and fix the currently displayed address of an output or input here in the dialog. This might be useful in order to adapt the addressing to a given hardware configuration resp. to keep the address value even if the order of the modules gets changed (which per default would cause an automatic adaptation of the address values).

Regard that you can only modify the address of the entire input or output, not however that of its subelements, i.e. bit channels. So, if an input or output is represented here in the mapping table with any subtree, only the address field of the uppermost entry can be edited (see the figure below: only the address field in the first line can be opened).

In order to fix the address value, select the entry in the "Address" column and press the <space> bar to open the edit field. Now either modify the value or leave it unmodified and close the edit field via the <Return> key. The address field will get marked by a Ⓜ symbol, which indicates that the current value is fixed. If the value has been modified, the subsequent addresses (up to the next fixed address) will be adapted correspondingly. See the following example for the modification and fixation of an address value:

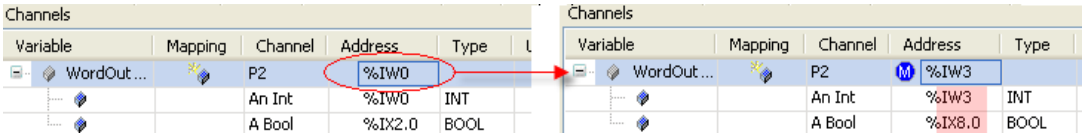


Abb. 475 Example, Address modified manually

If you want to remove the fixation of the value, reopen the address edit field, delete the address entry and close with <Enter>. The address and the concerned succeeding addresses will be set back to the values they had before the manual modification, and the M symbol will be removed.

Type: Data type of the input resp. output channel, for example "BOOL". If the data type is not standard, but a structure or bit field defined in the device description, it will be listed only if it is part of the IEC standard and indicated as IEC type in the device description. Otherwise the entry of the table will be empty.

Unit: Unit of the parameter value, for example "ms" for Milliseconds

Description: Short description of the parameter.

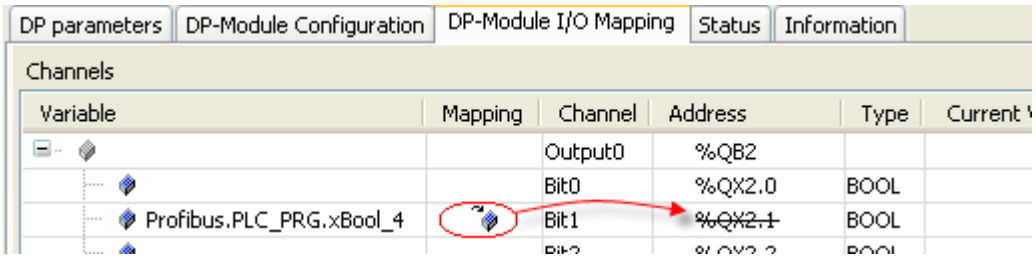
Current Value: Current value of the parameter, displayed in online mode.

The I/O mapping has to be done by specifying the corresponding project variables in column 'Variable':

The **type** of the channel is already indicated in the **Variable** column by an icon: for input, for output. In the respective table cell the path resp. name of the variable is specified, on which the respective channel should be mapped. You can map on an already existing project variable or define a new variable, which then will be declared internally as a global variable.

When mapping **structured variables** the editor will prevent that both the structure variable (for example on %QB0) and particular structure elements (for example in this case on %QB0.1 and QB0.2 etc.) can be entered. This means: When there is a main entry with a subtree of bit channel entries in the mapping table (as shown in the figure below), then either in the line of the main entry a variable can be entered or in those of the subelements (bit channels), never in both.

- If you want to **map on an existing variable**, you must enter the complete path, that is "<application name>.<pou path>.<variable name>"; example: "app1.plc_prg.ivar". For this purpose it is recommended to use the input assistant via button [...]. In the **Mapping** column the symbol will appear ("map on existing variable") and the address value will be shown **crossed out**. This does not mean, that this memory address is not existing any longer. However it is not used directly, because the value of the existing variable is managed on another memory location and, especially in case of outputs, not at all another already existing variable should be stored to this address ("%Qxx" in the I/O mapping), in order to avoid ambiguities during writing the values.
  See in the following example an output mapping on the existing variable xBool_4:



- If you want to **map on a new variable**, just enter the desired variable name; example: "bVar1". The icon will be displayed in the **Mapping** column and the variable will be **declared internally** as a global variable. As from there on the variable will be available globally within the application, the mapping dialog is another place for the declaration of a global variable.

Note: Monitoring: If a structure variable is mapped on the root element of the address (the uppermost in the tree of the respective address in the mapping dialog), then in online mode no value will be displayed in this line. If however for example a DWORD variable is mapped to this address, then in the root line as well as in the bit channel lines indented below the respective values will be monitored. Basically the field in the root line always will stay empty if the value is composed of multiple subelements.

Note: Mapping of "to big" data types: If a variable of a datatype which is bigger than a byte, gets mapped on a byte address, there the value of the variable will be cut to byte size! This means the following for the monitoring of the value in the mapping dialog: In the root element of the address the value of the variable - as currently valid in the project - will be displayed. In the bit elements below the particular bit values will be monitored, which however might not be enough for the complete variable value.

Reset mapping: Using this button will reset the mapping settings to the defaults defined by the device description file.

Always update variables: If this option is activated, all variables will be updated in each cycle of the bus cycle task, no matter whether they are used or whether they are mapped to an input or output.

Hint: Alternatively an address may be read out or written to within the programming code. Example in ST:

```
Var := IB100;
QX12.3 := TRUE;
```

Noticing possible changes of the device configuration we advise to implement such assignments within the device configuration dialog.

IEC Objects:

This part of the dialog only is available, if implicitly an **instance** of the device object has been created, which can be accessed by the application (for example in order to restart a bus or to pull information). Whether such an instance is available and how it might be used, depends on the device type and is described in the online help pages for the specific device configuration.

Bus cycle options:

This configuration options will be available for devices with cyclic calls before and after reading inputs or outputs. It allows to set a device specific bus cycle task:

Per default the "parent" bus cycle setting will be valid (**Use parent bus cycle setting**), that is the device tree will be searched up for the next valid bus cycle task definition.

To assign a specific bus cycle task, select the desired one from the selection list. The list provides the tasks currently defined in the applications' task configuration.

Implicit variables for forcing I/Os

During the commissioning of a plant or a machine it might be needed to force I/Os, for example by a HMI visualization. For this purpose special "force variables" can be generated for each I/O channel which is mapped on a variable in the I/O Mapping dialog of the device editor.

As a precondition the setting **Generate force variables for IO mapping** must be activated in the PLC settings dialog. Then at each **build** run of the application, for each mapped IO channel, two variables will be generated according to the following syntax, whereby any empty spaces in the channel name will be replaced by underscores:

- "devicename“_“channelname“_“IECaddress“_“Force" of type BOOL, for activating and deactivating the forcing
- "devicename“_“channelname“_“IECaddress“_“Value" of datatype of the channel, for defining the value to be forced on the channel

These variables will be available in the input assistant in category "Variables", "IoConfig_Globals_Force_Variables". They can be used in any programming objects, in visualizations, symbol configuration etc. within the programming system.

A rising edge at the "Force"-Variable activates the forcing of the respective IO with the value define by the "Value"-Variable. A falling edge deactivates the forcing. Deactivating by setting the force variable back to FALSE must be done before a new value can be forced!

Regard the restrictions listed below.

Example

If the mapping is done like shown in figure "Device dialog, I/O Mapping, Example" (see above), then at a build (F11) of the application the following variables will be generated and be available in the input assistant:

- Digitax_ST_Control_word_QW0_Force : BOOL;
- Digitax_ST_Control_word_QW0_Value : UINT;
- Digitax_ST_Target_position_QD1_Force : BOOL;
- Digitax_ST_Target_position_QD1_Value : DINT;
- Digitax_ST_Status_word_IW0_Force : BOOL;
- Digitax_ST_Status_word_IW0_Value : UINT;
- Digitax_ST_Position_actual_value_ID1_Force : BOOL;
- Digitax_ST_Position_actual_value ID1_Value : DINT;

Restrictions

- Only channels which have got mapped on a variable in the I/O Mapping dialog (i.e., a variable must be defined in the "Variable" column, no matter whether it is a new one or an existing one) can be forced by the above described implicit variables.
- Unused inputs / Outputs as well as such which are mapped via AT-declaration in an application program cannot be forced.
- The respective I/O channels must be used in at least one task.
- Forced I/Os are not indicated in the monitoring (watch view, I/O mapping dialog etc. The value is only used implicitly in the IO driver for writing onto the device
- Forced inputs are displayed correctly by the red force symbol , not however forced in-/outputs.

Users and Groups

For devices supporting an "**online user management**", this dialog is provided on a tab of the **Device** dialog (Device Editor). It allows for setting up user accounts and user groups, which in combination with the Access Rights management serves to control the access on objects on the PLC in online mode.

Common

It might be desired that certain functions of a controller can only be executed by authorized users. For this purpose the "online user management" feature provides the possibility to set up user accounts, to assign access rights for user groups and to force an user authentication at login.

The device specific user management might be pre-defined by the device description and it also depends on this description to what extent the definitions can be edited in the configuration dialogs in the programming system.

Like in the project user management users have to be members of groups and only user groups can get certain access rights.

Using the configuration dialog

Basically the handling of the user management dialogs is very similar to that of the project's user management. There is even the possibility to "import" user account definitions from the project's user management.

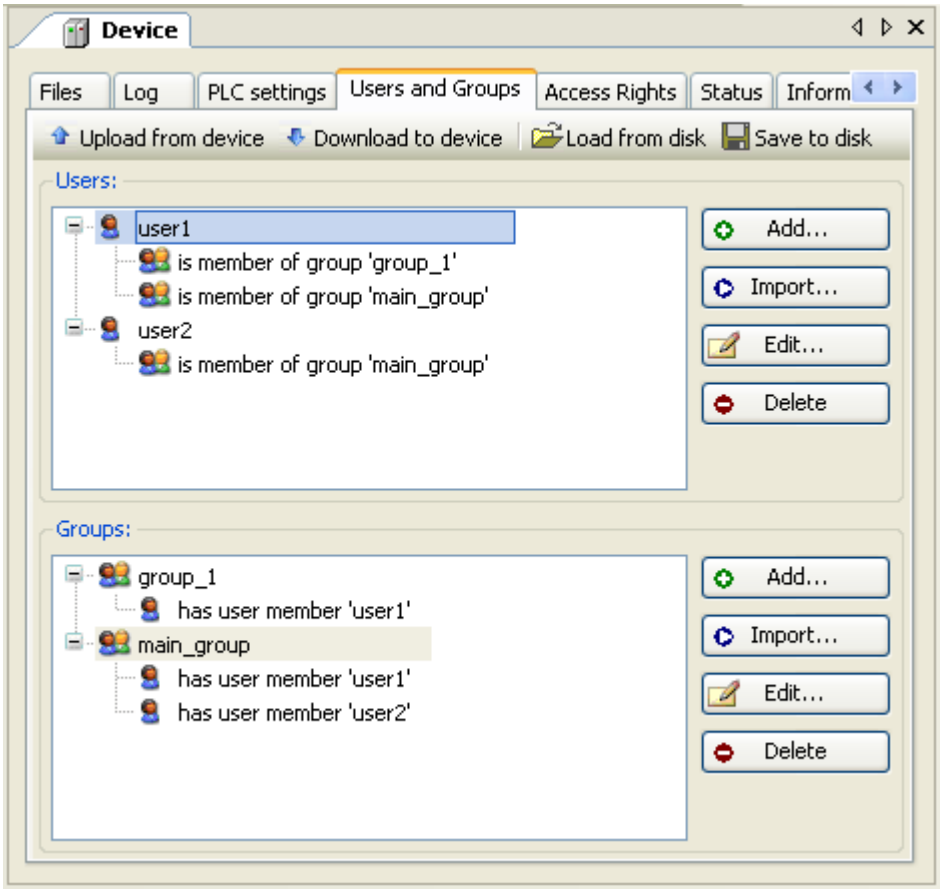
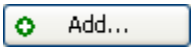


Abb. 476 Device dialog, Users and Groups

Users:

The following buttons are available for setting up user accounts:



The dialog '**Add User**' opens where you can define a **user name** and a **password**. The password must be repeated in the **Confirm password** field.

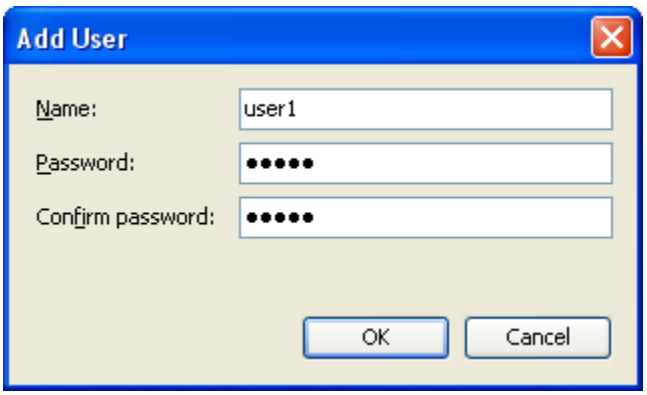


Abb. 477 Add User dialog

Import... The dialog 'Import Users' opens showing all user names which are currently defined in the project user management. Select one or several entries and confirm with OK. The dialog **Enter password** will open where you have to enter the corresponding password as it is defined in the project user management, in order to get the user account imported to the device-specific user management. Notice however, that for security reasons these passwords will not be imported and have to be re-defined.

Note: Each imported user account basically will have an empty password definition!

Edit... The currently selected user account can be modified concerning user name and password. This '**Edit User <user name>**' dialog corresponds to the 'Add User' dialog (see above).

Delete The currently selected user account will be deleted.

Groups:

Add... The dialog '**Add Group**' opens where you can define a new group name and select from the currently defined users those who should be members of this group.

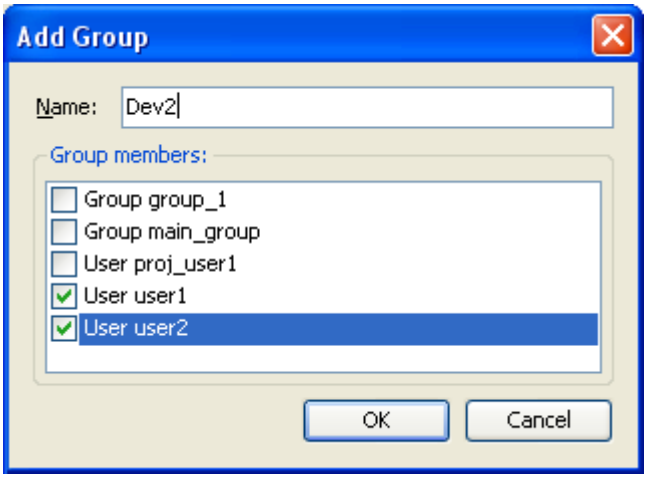


Abb. 478 Add Group dialog

Import... The dialog '**Import Groups**' opens where the groups currently defined in the project user management are listed. You can select one or several entries and confirm with OK to get them integrated in the groups list of the device-specific user management.

Edit... The currently selected group can be modified concerning group name and associated users. For this purpose the '**Edit Group <group name>**' dialog opens, which corresponds to the 'Add Group' dialog (see above).

Delete The currently selected group can be deleted.

Applying and storing the current configuration:

See the respective buttons in the top bar of the dialog:

,

: The current User Management configuration must be downloaded to the device to get it effective. The configuration currently applied on the device can be uploaded into the configuration dialog.

,

: The current configuration can be stored in a xml-file (***.dum**) and re-loaded from this file, which is useful to set up the same user configuration on multiple systems. The standard dialog for browsing in the file system will be provided for this purpose. The file filter automatically is set to "*.dum", which means "device user management" files.

The actual settings can also be documented as printed version by use of the command "Print..." (File menu) or "Document..." (Project menu).

Access Rights

This dialog is provided on a tab of the **Device** dialog (Device Editor). It is part of the "online user management" feature and serves to grant resp. deny the currently defined user groups certain permissions, thus defining the users' access rights on files or objects (like for example an application) on the PLC during runtime.

Notice that these permissions **can only be assigned to groups**, not to particular users. For this reason a user must be defined as member of a group. The configuration of users and groups is done in the Users and Groups tab of the device editor.

See for example in the following picture the permission to add and remove children to/from the PlcLogic object granted for user group "main_control".

See in the following how to define the access permissions and how to get them loaded to the device or stored in a reloadable file.

Defining the access permissions

To define the permission for performing an action on one or multiple object(s), you have to select the object entri(es) below the desired action type in the 'Actions' window, then select the desired group in the 'Permissions' window, and then click on the Grant or Deny button (also in the Permissions window).

See the instructions in the Howto window.

See in the following a description of the particular dialog windows:

Actions:

This part of the dialog lists the possible actions which might be performed during runtime on files in the PLC file system resp. runtime objects like for example applications. The tree is structured in the following way:

Object categories:

Top-level, for structuring purposes, there are two "folders" for **File system objects** and **Runtime objects**.

Action types:

Indented below there are nodes for the four **types of actions**, which might be performed on the particular objects. These nodes also just serve for structural purposes:

Add/remove children (adding or removing of "child" objects to an existing object)

Execute (for example start/stop application, setting breakpoints etc.)

Modify (for example downloading application...)

View (monitoring)

Objects (action "targets"):

Below each action type node finally find the "targets" of the action, that is the PLC file or runtime objects, for example Device.
These object entries are displayed in a tree structure mapping the device tree or the file system structure.

Note: Assigning an access right definition to a "father" in the objects tree usually means that the "children" will inherit this definition, as long as they do not get an explicit own definition. However, depending on the device, this might be handled differently. In any event, inheritances are not visualized here in the dialog.

Permissions:

This field shows the currently defined user groups. Before each group one of the following icons indicates the currently assigned permission concerning the target which is currently selected in the Actions window:

: The action(s) currently selected in the Actions window are granted for the group.

: The action(s) currently selected in the Actions window are denied for the group.

: Currently there is no explicit access right definition for the action(s) currently selected in the Actions window.

If currently multiple actions are selected in the Actions window, which do not have unique settings referring to the currently selected group, no icon will be displayed.

Button bar: After having selected the desired object(s) below the desired action in the Actions window and having selected the desired group in the Permissions window, one of the following buttons can be used:

Grant: Explicit granting access permission.

Deny: Explicit denying access permission.

Clear: The currently granted access right for the action(s) currently selected in the Actions window will be deleted, that is set back to the default.

Applying and storing the current configuration:

See the respective buttons in the top bar of the dialog:

Download to device, Upload from device: The currently configured Access Rights definitions must be downloaded to the device to get it effective. The configuration currently applied on the device can be uploaded into the configuration dialog.

Save to disk, Load from disk: The current configuration can be stored in a xml-file (***.dar**) and re-loaded from this file, which is useful to set up the same user configuration on multiple systems. The standard dialog for browsing in the file system will be provided for this purpose. The file filter automatically is set to "*.dar", which means "device access rights" files.

The actual settings can also be documented as printed version by use of the command "Print..." (File menu) or "Document..." (Project menu).

Status

This dialog is provided on a tab of the **Device** dialog (Device Editor). It shows status information (for example "Running", "Stopped") and specific diagnosis messages from the respective device; also on the used card and the internal bus system.

Information

This dialog is provided on a tab of the **Device** dialog (Device Editor) which can be opened for the device currently selected in the devices tree via command 'Edit Object' or by a double-click on the device entry in the Devices window.

If available for the current device, the following general information will be displayed: Name, Vendor, Type, Version number, Order Number, Description, Image.

14.2 CANbus Configuration Editor

The 'DeviceEditorCANbus' plug-in provides dialogs for the configuration of CANopen devices and modules according to CANopen Draft Standard 301. Special CANbus configuration dialogs will be added to the standard Device Editor dialogs. The appropriate device objects must be inserted in the devices tree in order to map the respective CANbus hardware.

The CANbus devices and modules get a configuration which describes their time- and detected error-dependant behaviour during the exchange of messages. Further on the mapping of inputs and outputs on **PDO**s (Process Data Objects), serving for sending and receiving process data, is done for the particular modules. A list of **SDO**s (Service Data Objects) will then be created which transfer - request+response services - the complete configuration data to the CANbus target system. There the communication and process data objects are managed in the "Object Dictionary".

The CANopen modules configuration initially is determined by the device description files of the used master and slave devices and might be modified by the user in the configuration editor dialogs. Notice the recommendation to use the autoconfiguration mode for standard applications and modular devices.

So in order to set up a CANbus configuration in the devices tree...

1. Install the required description files
2. If CANbus shall be used together with a SJA Chip, the »PLC Designer« configuration file (...\GatewayPLC\»PLC Designer«SP.cfg) has to be enlarged by the following entry:
   Component.<subsequent number>=CmpSJACanDrv
3. Map the bus structure by inserting the respective modules in the devices tree
4. Set the appropriate configuration parameters in the device editor dialogs

Installing CANopen devices:

As a precondition for mapping the CANbus hardware in the devices tree the corresponding **device descriptions** must be installed via the Device Repository dialogs on the local system. This concerns the description files (***.devdesc.xml**) for the basic modules CANbus and CANopen_Manager, which by default are provided with the standard CANbus configuration plug-in, as well as the manufacturer-specific **EDS-files** (Electronic Data Sheet) for the CANopen devices and modules which are used in the bus system. These files each contain the parameter definitions and configuration options of a module.

Basic hierarchy of a CANbus configuration in **»PLC Designer« /** Inserting the CANbus devices :

- The basic node, that is the uppermost entry in a CANbus configuration tree, must be the "**CANbus**" object. A CANbus can only be inserted below an appropriate programmable device. So select this device in the tree and choose "CANbus" in the 'Add Device' dialog:

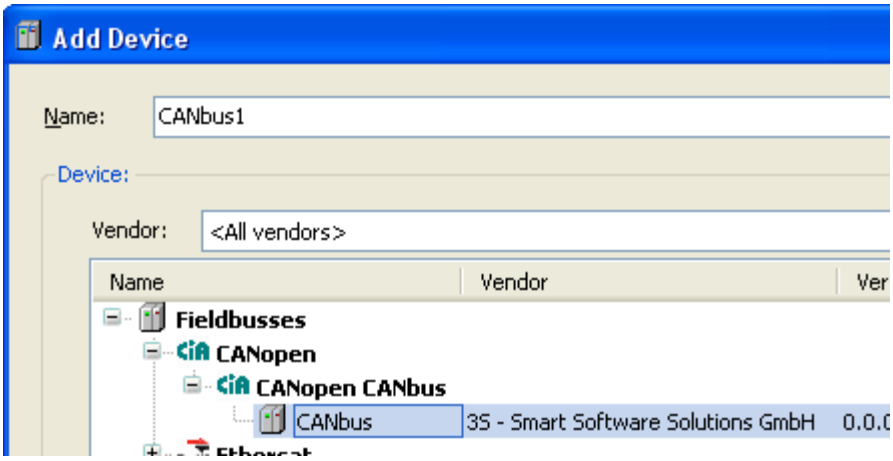


Abb. 479 Add CANbus object

- In the CANbus node check or adapt the baud rate for the data transfer on the bus and the number of the CAN network to be connected via the CANbus interface.
- Below a CANbus entry the obligate device object "**CANopen Manager**" must be inserted. This can be seen as the "master" of the CANbus. It currently supports the configuration of the CANbus by internal functions and so no configuration input by the user is needed. The CANopen Manager simplifies the setup of the connections within the network, especially by helping to simplify the PDO-Linking between the particular modules

- Below the CANopen Manager finally the particular **CANopen devices** objects ("slaves") provided with the hardware can be added. Here two types are possible: non-modular devices and modular devices. Below modular devices further **CAN modules** can be inserted and automatic I/O mapping can be used.

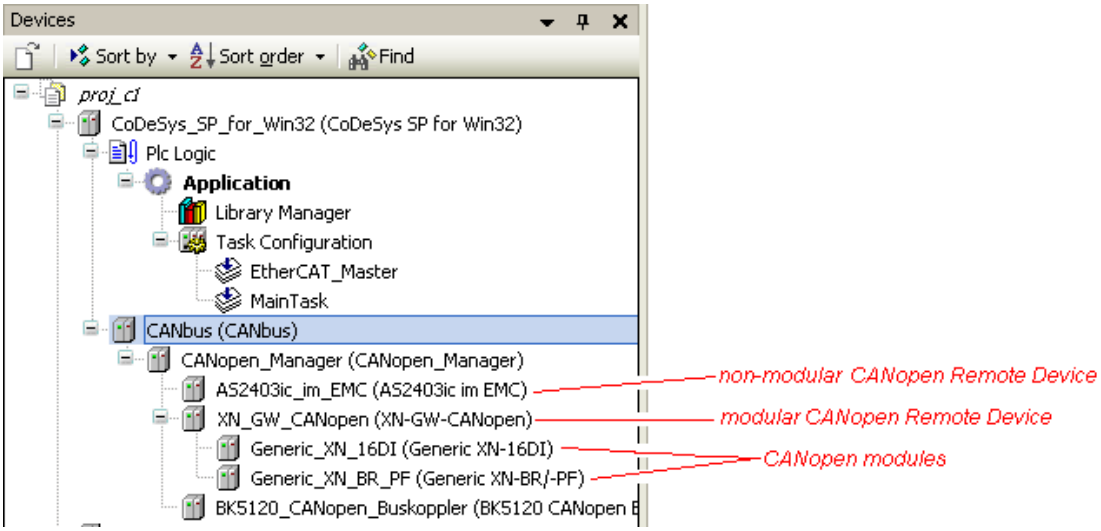


Abb. 480 Example: CANbus configuration tree

Configuration

After having mapped the CANbus hardware environment in the devices tree, use the configuration dialogs to check or adapt the default configuration settings:

For the device entry which is currently selected in the devices tree, the appropriate configuration dialog will open in the Device Editor window when you perform a double-click or use command 'Edit object'.

The configuration possibilities for a device or module depend on the definitions read from the device description file.

Notice the Device Editor Options for general editor settings: For example generic dialogs might be set invisible.

A CANopen configuration dialog always is entitled with the name of the module and can provide tabs containing the following sub-dialogs:

For a CANbus:

- CANbus (general settings, currently only: baud rate)
- CANbus Configuration (settings as provided by the *.devdesc-file)
- CANbus I/O Mapping
- Status
- Information

For the CANopen Manager:

- The standard dialogs of a device editor will be available, but you do not have to use them for configuration purposes, because the device description of the CANopen Manager currently contains no parameters (Common.CANbus Configuration, CANopen Manager, CANopen I/O Mapping, Status, Information).

For a CANopen device:

- CANopen Remote Device (main module settings)
- PDO Mapping
- Receive PDO Mapping
- Send PDO Mapping
- Service Data Object
- CANopen Configuration (parameters as defined by the EDS-file)
- CANopen I/O Mapping
- Status
- Information

For a CANopen module:

- CANopen-Module Configuration (parameters as defined by the EDS-file)
- CANopen-Module I/O-Mapping
- Status
- Information

Interface for a CANopen device

To be able to access a CANopen device out from an IEC application, you may implement the interface provided by the library IIoDrvBusControl.Library.

CANbus Node

CANbus

The **CANbus node** object is the obligatory top level entry of a CANbus Configuration in the device tree.

In the corresponding configuration dialog the following is defined:

Baudrate (bits/s): Baud rate for the data transfer on the bus. The default value is read from the description file (*.devdesc) of the CANbus device. The following baud rate values can be set: 10000, 20000, 50000, 100000, 125000, 250000, 500000, 800000 and 1000000. The unit is bits/s, whereby in the configuration data the baud rate will be stored in KBit/s.

Network: Number of the CAN network to be connected via the CANbus interface. Values from 0 to 100 can be entered.

Online Bus Access

Block SDO and NMT access while application is running: If this checkbox is activated it is not possible for users to interact with a CANopen Slave while the IEC App is running on the target (e.g. the online services on the status page).

CANbus Configuration

This configuration dialog for the **CANbus** node is currently without function, because the device description - with exception of the baud rate and net number - is not providing any further configuration parameters.

Status

This tab of the configuration editor for an CANbus device or module provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog in the CANopen device or module configuration shows information on the device resp. module which is currently selected in the devices tree. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

CANopen Manager

The **CANopen Manager** is an obligatory node below the CANbus node in a CANbus configuration. It is supporting the CANbus configuration by internal functions, thus usually serving as the CANbus master.

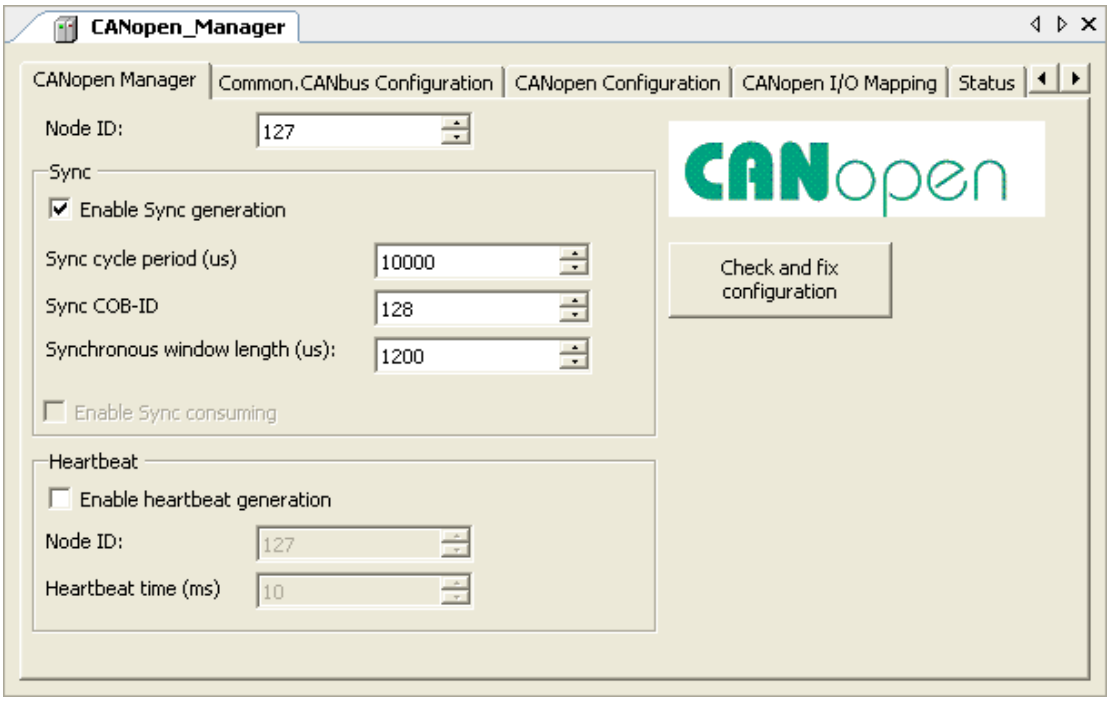


Abb. 481 CANopen Manager dialog

In the CANopen Manager dialog you can enable the synchronized transmission mode for the PDO transfer on the bus and set some default values, which might be adapted to the special use case in the Common.CANbus Configuration:

Network management:

Autostart CANopen Manager: This option specifies whether the CANopen Manager is to automatically change over to the status 'Operational'. Default: active.

Start slaves: This specifies whether the CANopen Manager is to start the slaves. Default: active.

NMT Start All: If this option is set, the Node command is dispatched as a Broadcast command. If the option is not set, individual Start Node commands are sent to each slave. Default: not active.

N.B.: Some slaves do not support the Broadcast Start Node command, e.g. the Lenze ECS drive.

Polling optional slaves: Optional slaves are polled cyclically by means of an SDO read request. This function is only needed if the optional slave does not autonomously send any boot telegrams after switch-on. Default: not active.

Sync:

Enable Sync generation: If this option is enabled (default: disabled), the transmission on the bus will be synchronized and the following defaults can be set:

Sync cycle period (µs): Interval in microseconds in which the synchronization message will be sent. Possible values: [100,2^{32}-1].

Sync COB-ID: Communication Object Identifier, which identifies the synchronization message. Possible values: [1, 2047].

Synchronous window length (µs): Contains the length of the time window for synchronous PDOs in microseconds. Possible values: [1,2^{32}-1] or 0 if not used.

The synchronous PDO's will be transmitted directly after the synchronisation message.

Enable Sync consuming: If this option is enabled (default: disabled), another device will generate the synchronization messages that the CANopen manager will receive.

Note: Enabling the Sync generation for the CANopen Manager disqualifies all of the other bus clients from enabling the option Cycle.

Heartbeat:

Working with Heartbeats is an alternative guarding mechanism: In contrast to the Nodeguarding functionality it can be executed by Master- and Slave-Modules. Usually the master will be configured to send heartbeats to the slaves.

Enable heartbeat generation: If this option is activated, the master will send heartbeats according to the interval defined in "Heartbeat time". If new slaves with heartbeat functionality are added, their heartbeat behaviour will automatically be enabled and configured appropriately, that is, the node-ID is identical to the setting in the manager configuration, whereas the heartbeat interval gets multiplied by the factor 1,2. If heartbeat generation of the CANopen manager is not enabled, then nodeguarding is enabled instead for the slave (with life time factor 10 and a guard time of 100ms). Notice that the CANopen (slave) devices might be configured as heartbeat producer.

Node ID: Unique identifier of the heartbeat producer (1 - 127) on the bus.

Heartbeat time (ms): Interval of heartbeats in milliseconds.

When multiple devices are added to the CANopen manager, you may get error messages reporting multiple assigned node IDs or wrong COB-IDs stemming from the device description files. The button **Check and fix configuration** allows to resolve these conflicts within the opening dialog:

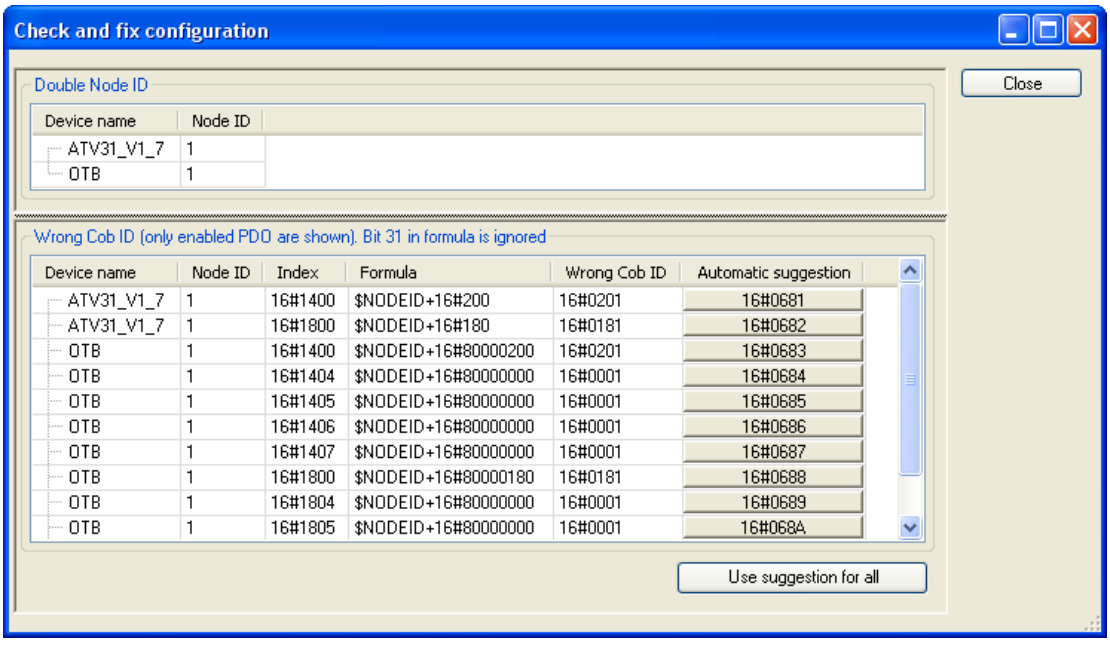


Abb. 482 Dialog - Check and fix configuration

Double Node ID: This box contains a list of all devices carrying identical IDs. You may assign new node IDs by editing the fields in column "Node ID".

Wrong Cob Id (only enabled PDO are shown). Bit 31 in formula is ignored: Unfortunately the Cob Ids created automatically out of the device description file might not be correct. All incorrect entries are listed with their related device names, node IDs and indices. There are three possibilities to correct the COB IDs:

- Either you edit the displayed formula in such way that a valid COB ID will result. The editing may be done directly in the corresponding table field.
- Or you adopt the "Automatic suggestion" given in the last column of the table by clicking on it.
- Or you adopt all automatically created suggestions (displayed in the last column) at once by clicking on **Use suggestion for all**.

Corrected items will be removed from the list automatically.

Common.CANbus Configuration

This dialog will only be available in the device editor for a CANopen Manager device, if option 'Show generic device configuration views' is activated. It corresponds to the Configuration dialog provided by the Device editor for other device modules.

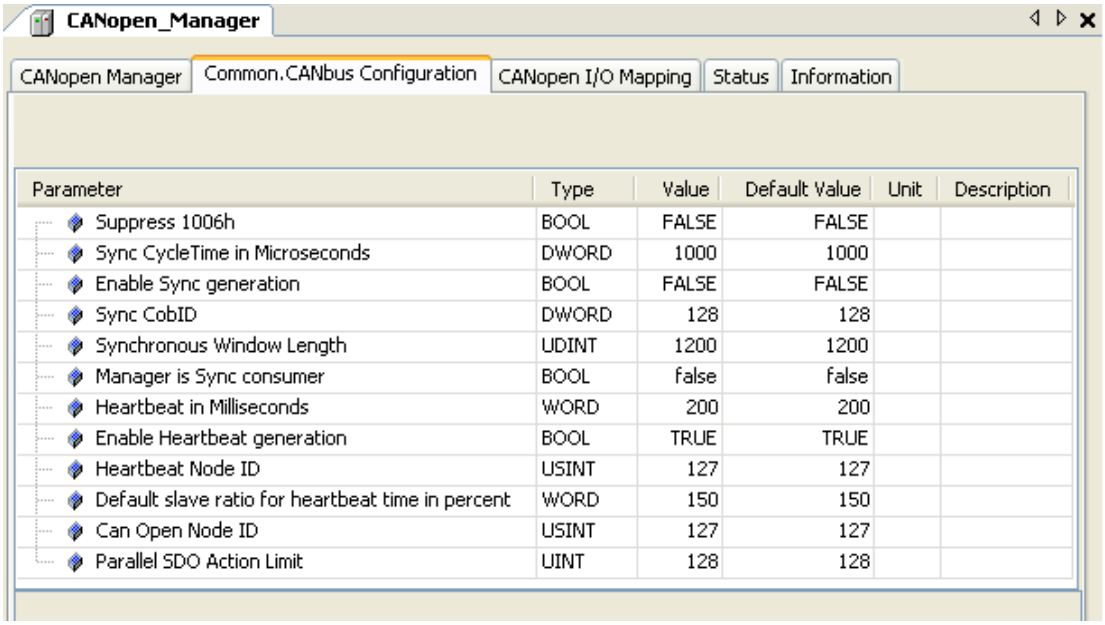


Abb. 483 Common.CANbus Configuration dialog

The device-specific parameters defined by the manufacturer in the device description file will be listed in a table and might be edited there if allowed by the description file.

For most of them the defaults can be set in the CANopen Manager dialog, the following ones however are only available here in the Common.CANbus Configuration dialog:

Suppress 1006h: If TRUE, no object 0x1006 will be written even if Sync Generation is activated. If FALSE or not available, object 0x1006 will be written like in previous versions. Default: FALSE.

Default slave ratio for heartbeat time in percent (Parameter ID 275): If Enable heartbeat generation is activated for the CANOpenManager, then this value will be taken for calculating the default Heartbeat Consumer Time of a slave. Default: 150.

Optional Device Default (Parameter ID 276): Specifies the default value for the Optional Device checkbox in the slave configuration dialog. Default: FALSE.

Parallel SDO Action Limit: Number of SDO actions which can be processed simultaneously, that is the number of SDO channels of the CANopen Manager. Concerns the SDOs used for the configuration as well as those requested by the application. Default: 128.

Status

This tab of the configuration editor for an CANbus device or module provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog in the CANopen device or module configuration shows information on the device resp. module which is currently selected in the devices tree. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

CANopen Device

Non-modular and modular devices

In a CANbus configuration two types of CAN devices can be inserted below the CANopen Manager node:

A **modular device** allows to insert CAN modules below. It does not have a CANopen //O Mapping dialog, because the particular sub-modules have their own mapping lists. The CANopen Remote Device dialog of a modular device provides option 'Autoconfig PDO Mapping' for activating an **automatic mapping** configuration, which is recommended for standard applications.

A **non-modular device** can not get further modules below. So in contrast to modular devices the configuration dialogs also include an I/O-Mapping dialog and there is no option for automatic mapping configuration.

CANopen Remote Device

In this dialog of the CANopen device configuration you may specify the node ID (1-127) of the device. After marking the checkbox "**Enable Expert PDO settings**" the dialog will be enlarged showing all main default parameters for the CANopen module, as provided by the device description file (EDS-file). They might be modified by the user.

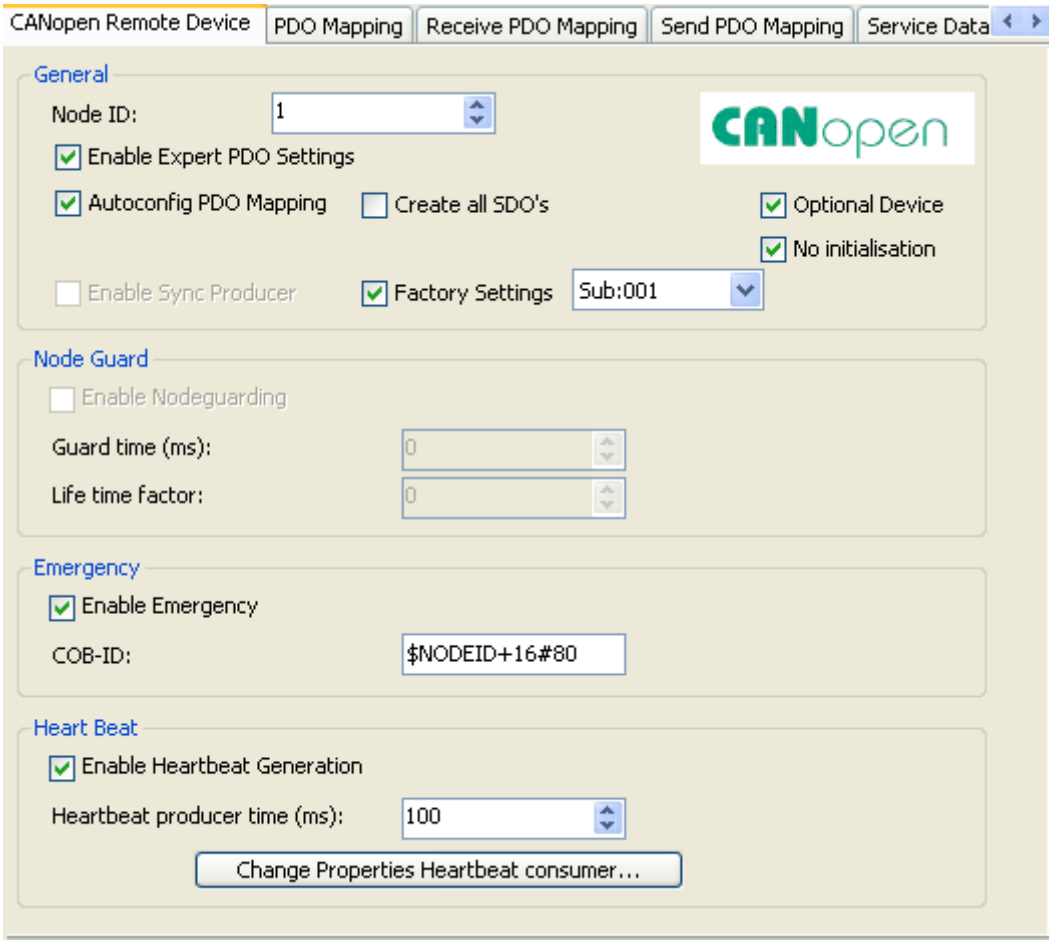


Abb. 484 CANopen Remote Device configuration

General:

Autoconfig PDO Mapping: If this option is activated (**which is strongly recommended for standard applications!**), the PDO Mapping (dialogs Receive PDO Mapping and Send PDO mapping) will be generated automatically basing on the definition given by the device description file and cannot be modified in the Mapping dialogs. This option is not available for non-modular devices.

Node-ID: The Node-ID serves to identify the CAN module uniquely and corresponds to the set number on the module itself which is between 1 and 127. The Id must be entered as a decimal number.

Write DCF: Currently without function !

Create all SDO's: All objects independent on the fact the default value has been changed or not will be downloaded to the PLC.

Factory settings: It depends on the content of the device description file, if this option is available. If so, the option is activated by default. It causes a reset of the CANopen communication parameters of the slaves to their defaults before downloading the configuration. It depends on the device, which parameters can be reset at all, and the particular configuration is done by choosing a subindex from the selection list next to the option box. For example per default via subindex 2 (1000h to 1FFFh, object 1011h) the CANopen communication settings are addressed.

Optional device: If this option is activated (availability in dialog is target dependent), the master only once will try to read from this node. Then the node, if not answering, will be ignored, that is the master will return to normal operation mode.

No initialization: If this option is activated (availability in dialog is target dependent), the master immediately will activate the node, without sending configuration SDOs. (The SDO data nevertheless will be created and saved on the controller.)

Node Guard:

Enable Nodeguarding: If this option is activated, a message will be sent to the module according to the **Guard Time** interval (milliseconds, 100 by default). If the module does not then send a message with the given Guard COB-ID (Communication Object Identifier), it will receive the status "timeout". As soon as the number of attempts (**Life Time Factor**, 10 by default if there are no other default settings within the device configuration file or if this default setting equals 0) has been reached, the module will receive the status "not OK". The status of the module will be stored at the diagnosis address. No monitoring of the module will occur if the variables Guard Time and Life Time Factor are not defined (0).

Emergency:

A module sends an emergency message, with a unique **COB-ID**, when an internal error has been detected. These messages, which vary from module to module, are stored in the diagnosis address.

Heart Beat:

The heartbeat protocol is a protocol for error detection. The heartbeat producer is sending a cyclic message with low priority. This message gets received and evaluated by multiple heartbeat consumers . If a consumer does not receive the heartbeat, it will send an appropriate message.

Enable Heartbeat Generation: If this option is activated, the module will send heartbeats according to the interval defined in **Heartbeat Producer Time (ms**, 10 by default if there are no other default settings within the device configuration file or if this default setting equals 0).

Change Properties Heartbeat Consumer: This button opens a dialog where the nodes defined in the EDS-file are listed and can be selected for getting guarded. For this purpose set a check in the Enable field and enter the desired value in milliseconds in the Heartbeat time field (by a double-click on this field you can open a selection box of time values). If the Hearbeat Consumer option is activated, then the respective module will listen to heartbeats which are sent by the master. As soon as no more heartbeats are received, the module will switch off the I/Os.

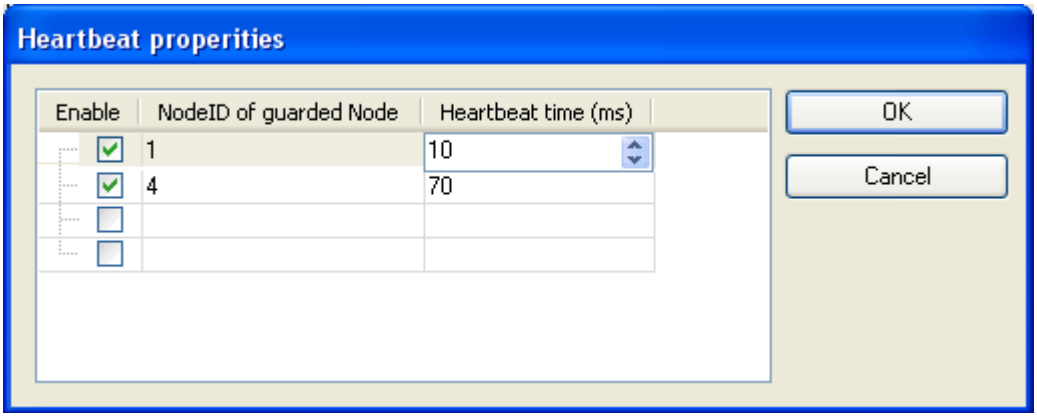


Abb. 485 Heartbeat Properties dialog

Note: If a device with heartbeat functionality is inserted, its heartbeat settings automatically will be adjusted to those of the master (CANopen Manager).

PDO Mapping

This dialog of the CANopen device configuration is just an alternative and simplified display of all currently defined Send and Receive PDOs.

If write access is defined for the PDO (definition in the device description file, default: write access) the PDO can be enabled resp. disabled here in the dialog. For modular devices however only if the Autoconfig mode is deactivated.

The transmission mode always can be modified here. For this purpose open the PDO properties dialog by a doubleclick on the respective PDO entry.

Further modifications of the PDO configuration can only be done in the dialogs Receive PDO Mapping and Send PDO Mapping.

Receive PDO Mapping

This dialog is only available, if option Enable Expert PDO Settings is activated for the device.

In this dialog of a CANopen device configuration the mapping for Receive PDOs is defined, that is the assignment of **outputs** (object dictionary entries) to a PDO for the purpose of data exchange in the CANbus. The corresponding allocation to the IEC addresses will be displayed in the I/O Mapping table.

The dialog shows the PDOs and indented below each PDO the respective mappings. In column **Name** you find either the COB-ID of the PDO resp. the name of the mapped object, as defined in the device description file and as used in the object dictionary. Also the **Index**, **Subindex** and **Bitlength** for the PDO resp. for the mapped object are displayed.

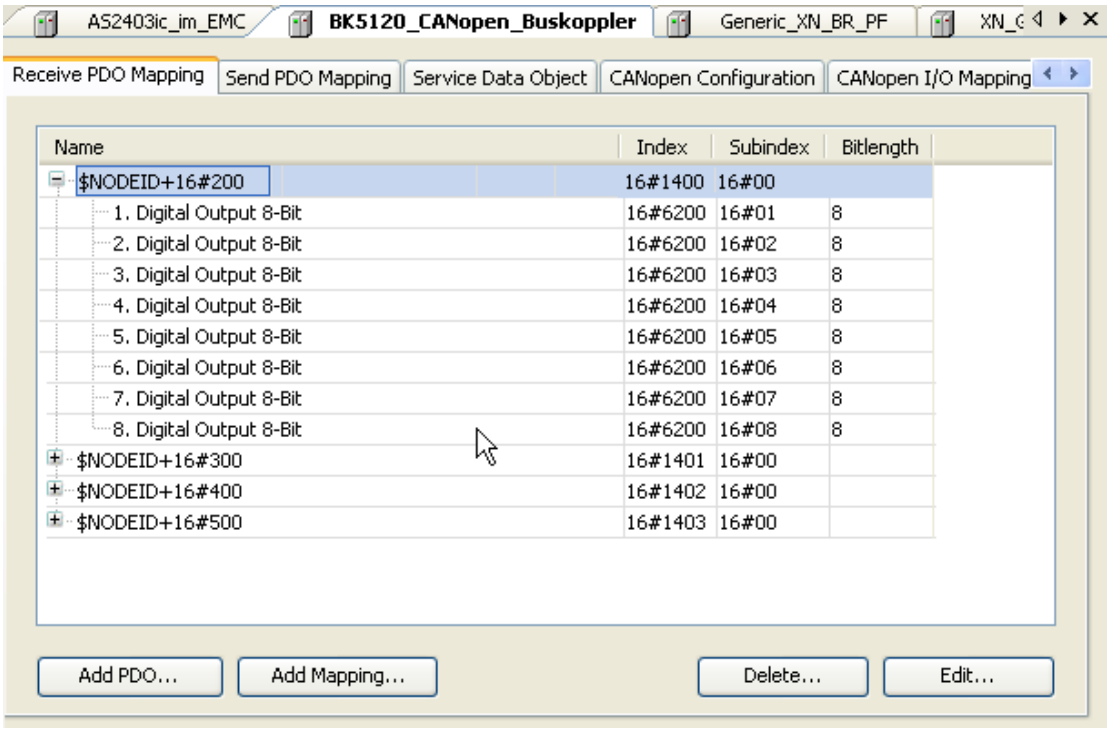


Abb. 486 Receive PDO Mapping dialog

By a double-click on a PDO entry the dialog **PDO Properties** dialog will open, showing the parameters as described below.

If option **Autoconfig PDO Mapping** is activated ...

.... which is strongly recommended for standard applications, the mapping displayed in this dialog will be generated automatically from the definitions in the device description files and you cannot modify it. In this case all relevant parameters (for example digital inputs and outputs) will be mapped according to a defined prioritization.

If option **Autoconfig PDO Mapping** is not activated **...**

you can configure the mapping manually. In this case you can modify existing PDOs or define new PDOs and fill them with the desired mappings. Via the **Delete** button the currently selected PDO resp. particular mapping can be removed from the configuration.

Modifying an existing PDO mapping

To modify an existing PDO, select it in the list and use button **Edit**. The dialog **Select item from object dictionary** will open, where you can add a new mapping or modify an existing one. See below for a description of this dialog.

Adding new PDOs and mappings

To add a new PDO, use button **Add PDO**. The **PDO Properties** dialog will open, where you might modify the default entries for the PDO.

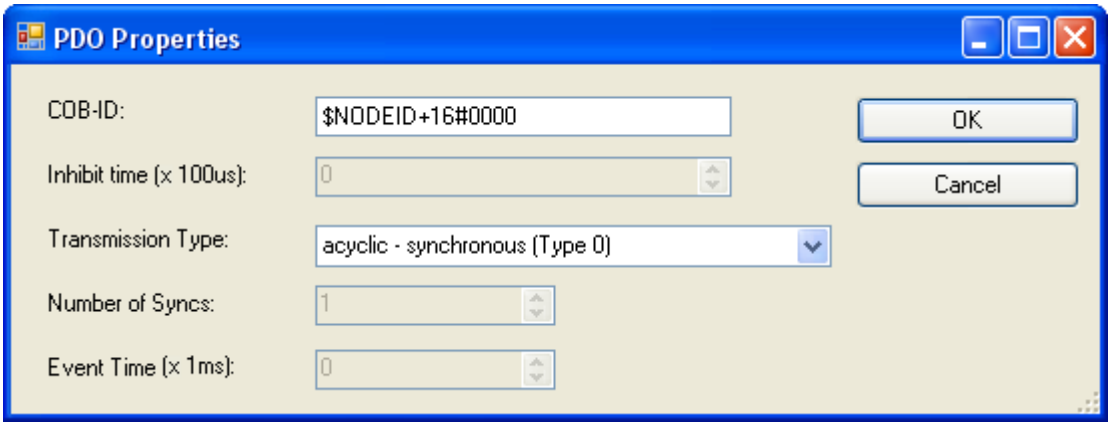


Abb. 487 PDO Properties dialog

COB-Id: Each PDO message must have an unique COB-Id (Communication Object Identifier). The COB-Id field shows the current definition of the COB-Id as defined in the EDS-file. An usual way is to generate it basing on the node number of the module whereby "$NodeID" is used to represent the actual node number. Example: "$NodeID+0x200". The current entry however might be modified by the user.

Inhibit-Time (x 100µs): This field is only editable, if the device supports this function: The Inhibit Time is the minimum time multiplied with 100µs between two messages transmitted via the same PDO. This setting for example can be used to avoid, that PDOs, which are sent each time when a value changes, get sent too often. The default entry will be "0" Possible values 0-65535.

Transmission Type: Here the desired transmission mode can be set. Possible settings:

acyclic - synchronous: the PDO will be transmitted synchronously but not periodically. (default)

cyclic – synchronous: the PDO will be transmitted synchronously, whereby the Number of Sync's gives the number of synchronisation messages, which lie between two transmissions of this PDO.

synchronous – RTR only: the PDO will be updated after each synchronisation message but not sent. It is only sent when there is an explicit request to do so (Remote Transmission Request)

asynchronous – RTR only: the PDO will only be updated and transmitted when there is an explicit request to do so (Remote Transmission Request)

asynchronous – device profile specific and asynchronous - manufacturer specific: the PDO will only be transmitted when specific events occur.

Number of Syncs: If cyclic transmission has been set, enter here the number of synchronisation messages (see 'Com. Cycle period' in the CANopen Remote Device dialog) which should be sent between two transmissions of the PDO. The default entry will be "1". Possible values 1-240.

Event-Time (x 1ms): This field is only editable, if the device supports this function: If an corresponding transmission type is set, enter here in milliseconds (ms) the interval between two transmissions. The default entry will be "0" Possible values 0-65535.

To **add mappings to a PDO**, select the PDO entry and use button **Add Mapping...**. The dialog **Select item from object dictionary** will open, where you can select the desired object to get added to the PDO.

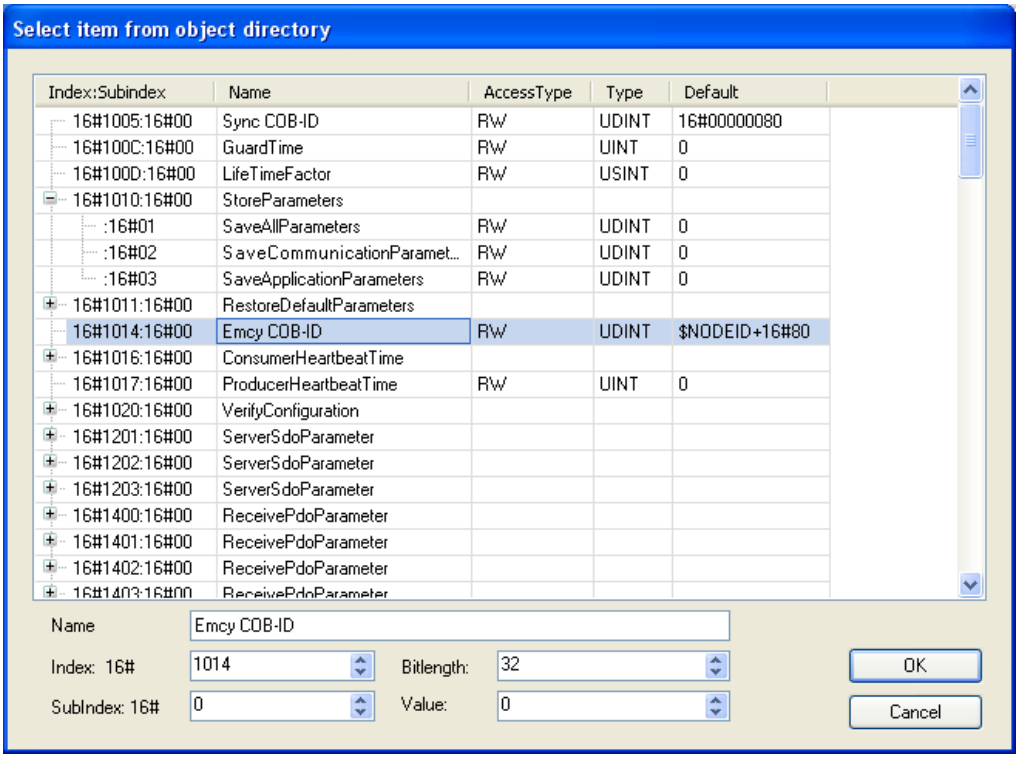


Abb. 488 Dialog Select item from object dictionary

The list will offer all mappable entries as defined by the EDS-file of the device. If applicable, use the plus-signs before the entries to make visible the particular subindex entries. In case of Receive-PDOs **only objects with write access** are listed (flag = w), in case of Send-PDOs those with read access. Each object is shown with Index, Subindex, Name, Value, Flag for access (RW=readable and writable, RO=readable only, WO=writeable only, CONST=constant and only readable, etc.), data Type and Default value.

For the currently selected entry the Name, Index, Subindex and Bitlength will be displayed in the fields below the table and can be **modified** there.

Send PDO Mapping

This dialog is only available, if option Enable Expert PDO Settings is activated for the device.

In this dialog of a CANopen device configuration the mapping for Sending PDOs is defined, that is the assignment of **inputs** (object dictionary entries) to a PDO for the purpose of data exchange in the CANbus.

The description of the dialog corresponds to that of the Receive PDO Mapping dialog.

Service Data Object

During the initialization of the CANbus the current configuration values will be transferred via Service Data Objects (SDO) to the CAN module.

In the Service Data Object configuration dialog the desired SDOs can be configured and it can be defined in which sequence they should be transferred and what should happen in case of an incomplete transfer.

Note: If option Enable expert PDO settings is not activated for the current device, only the user-defined SDOs will be listed here.

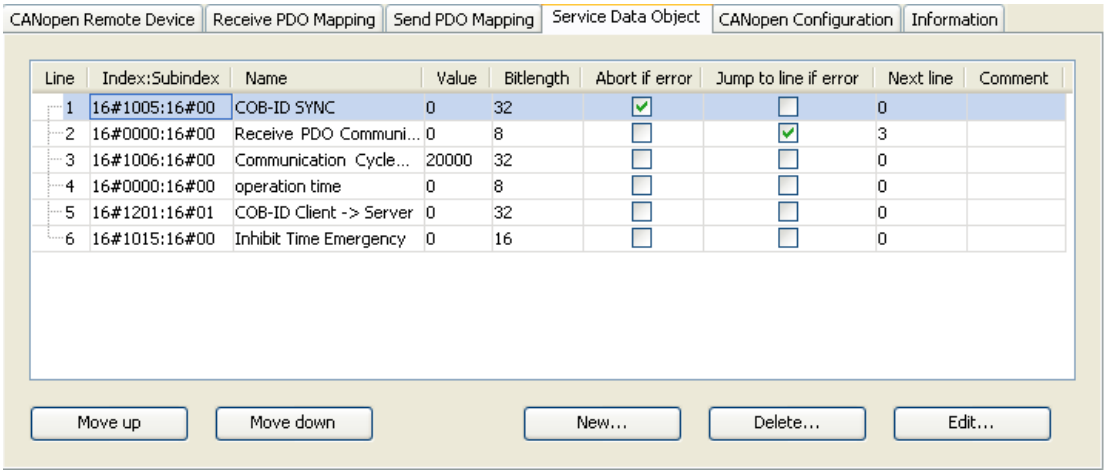


Abb. 489 Service Data Object dialog

To add an SDO entry to the configuration table, use button **New...**

The dialog **Select item from object dictionary** will open which - like for the mapping configuration - offers the available object dictionary entries as defined in the EDS-file.

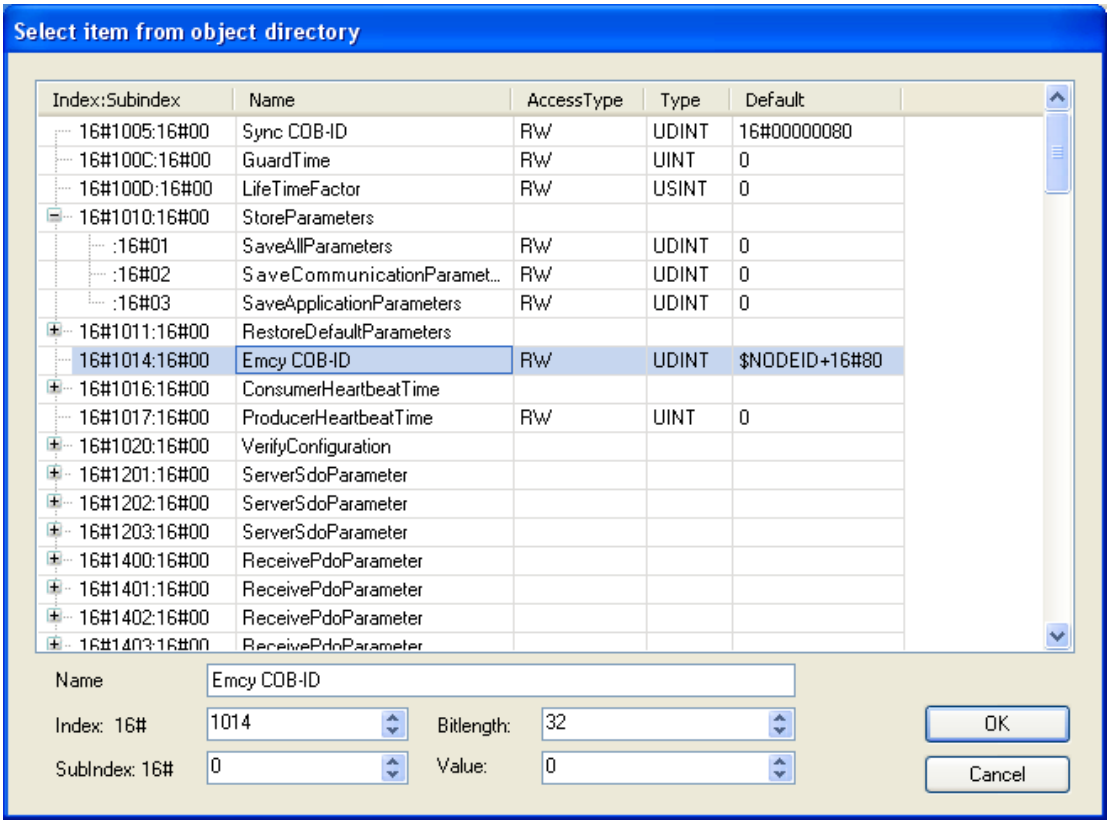


Abb. 490 Dialog Select item from object dictionary

In this case only the writable objects will be displayed (AccessType = *W*). Before adding an SDO, you might modify its parameters by editing in the fields below the objects table. So, by defining new index/subindex you optionally might add a new object to the SDO, which is not described in the EDS-file.

The order (top-down) in the SDO table represents the **sequence** in which the SDOs will be transferred to the module. By buttons **Move Up** and **Move down** the currently selected entry can be shifted within the list.

The **Value** field can be edited. For this purpose select the field and press <space> or perform another mouse-click.

You can define the behaviour in case of an error during SDO transfer by activating one of the following settings :

Abort if error: If an error was detected, the transfer will be terminated with error state.

Jump to line if error and **Next line**: If an error was detected, the transfer will be continued with the SDO specified in the line which is entered in field 'Next line' (line number, the line numbers are displayed in the **Line** column)

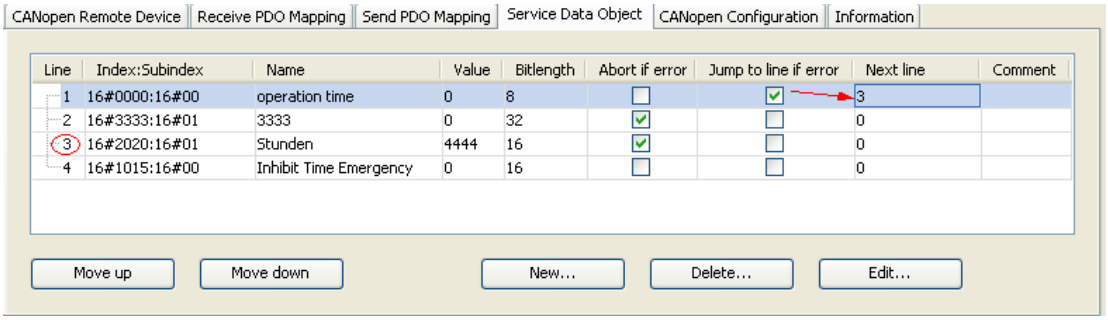


Abb. 491 Configure error handling, example

To modify the parameters of a SDO, which is already listed in the SDO configuration table, use button **Edit...** to get the **Select item from object dictionary dialog** where you can edit as described above for adding a SDO.

Button **Delete...** will remove the currently selected entry.

CANopen Configuration

In this dialog of a CANopen device configuration the parameters as described in the EDS-file are listed and their values might be modified by editing the respective field. This is the standard Device Editor dialog for parameters of a device.

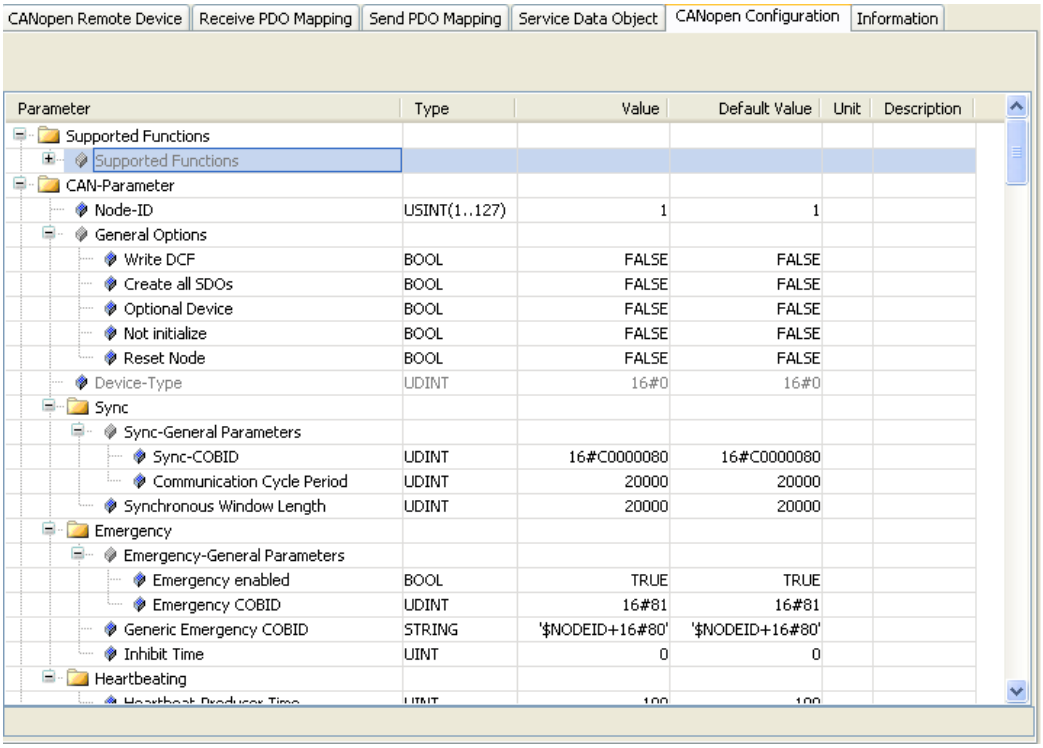


Abb. 492 CANopen Configuration dialog

CANopen I/O Mapping

This dialog is only available in the CANopen device configuration of a non-modular device. It is the standard Device Editor dialog for the configuration of the I/O-mapping of a module, that is for assigning IEC variables to input and output channels of the hardware.

Status

This tab of the configuration editor for an CANbus device or module provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog in the CANopen device or module configuration shows information on the device resp. module which is currently selected in the devices tree. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

CANopen module

CANopen-Module Configuration

This is the standard dialog of the device editor for handling the parameters of the currently selected CANopen module, which is part of a CANbus configuration.

CANopen-Module I/O Mapping

This is the standard dialog of the Device Editor for the I/O-mapping of a module.

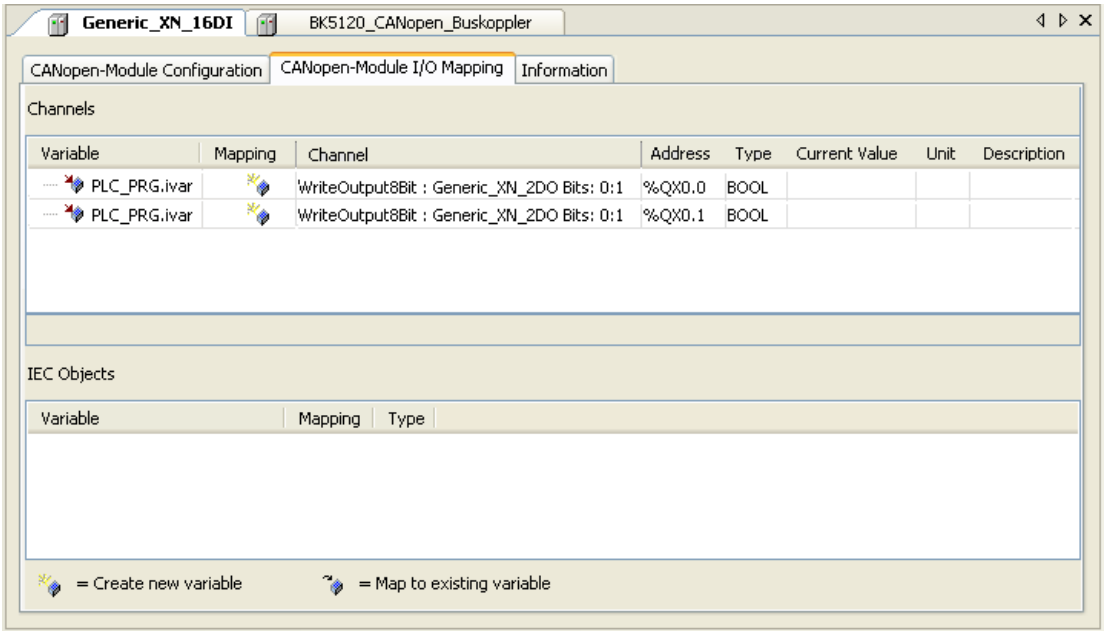


Abb. 493 Example of I/O Mapping dialog

If this dialog is available for a module, the "father"-module (CANopen device module) is a "modular device", which in contrast to a non-modular device has no own I/O-mapping configuration dialog.

If option Autoconfig PDO Mapping is activated, the I/O-mapping will be updated in all CANopen modules as soon as modules get added or removed below a modular CANopen device !

Status

This tab of the configuration editor for an CANbus device or module provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog in the CANopen device or module configuration shows information on the device resp. module which is currently selected in the devices tree. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

14.3 EtherCat Device Editor

The 'EtherCAT' plug-in provides dialogs for the configuration of EtherCAT Master and Slave modules inserted in the device tree of a project in the Devices view window. These dialogs will be added to the standard Device Editor dialogs.

The EtherCAT modules configuration initially is determined by the device description files of the used master and slave devices and might be modified by the user in the configuration editor dialogs. However, for standard applications and for the purpose of an as easy as possible handling, the configuration to a large extent can be automatized by choosing the "Autoconfig" mode.

Notice the possibility to access master and slaves and to modify configuration parameters also in operational mode by using master and slave instances as well as modules of the EtherCAT library.

General on EtherCAT

EtherCAT, the open real-time Ethernet network, allows real-time performance because it updates 1000 distributed I/Os in 30 µs or 100 axis every 100 µs using twisted pair or fiber optic cable. Line, tree or star topology are supported. There is a bus-master and slaves.

An Ethernet packet can be processed on the fly instead of getting received, then interpreted and copied as process data at every connection. The FMMU (fieldbus memory management unit) in each slave node reads the data addressed to it, while the telegram is forwarded to the next device. Similarly, input data is inserted while the telegram passes through. The telegrams are only delayed by a few nanoseconds. Addressing of the Ethernet terminals can be in any order because the data sequence is independent of the physical order. Broadcast, Multicast and communication between slaves are possible.

Accurate synchronization, which is for example needed in applications where several servo axes carry out coordinated movements simultaneously, is possible by accurate alignment of distributed clocks. Those in contrast to fully synchronous communication have a high degree of tolerance within the communication system.

Installing and Inserting EtherCAT devices

The combination of EtherCAT devices with a »PLC Designer« SP Win V3 requires the installation of the packet library WinPCap (for a free download visit for example winpcap.org).

In addition you have to enlarge the »PLC Designer« configuration file (...\GatewayPLC\»PLC Designer«SP.cfg) by the appropriate one of the following entries:

- Component.<subsequent number>=SysEthernet
- Component.<subsequent number>=CmpEt100Drv (to be used with IntelPro100)
- Component.<subsequent number>=CmpRTL81x9Mpd (to be used with RTL8139)
- Component.<subsequent number>=CmpRTL8169Mpd (to be used with Realtek RTL8169 or RTL8168(PCIe Version))

In order to be able to insert and configure EtherCAT devices as objects in the devices tree, the Master and Slave devices must be installed (which is automatically done by the standard »PLC Designer« setup) via the Device Repository dialog, using the **device description files** provided with the hardware. The device description file for the master (*.devdesc.xml) i.a. defines which slaves can be inserted. The descriptions for the slaves are also available as xml-files (file type: EtherCAT XML Device Description Configuration Files).

An EtherCAT Master can be added to the Devices tree via command 'Add Object' or 'Add Device'. Below a master one or several slaves might be inserted.

As soon as the master device is inserted in the devices tree, the required library IoDrvEtherCAT.library will automatically be included in the library manager. Furthermore an EtherCAT_Master task will be inserted in the Task Configuration of the currently active application. As usual you may add to this task the call of a POU, wherein you may set EtherCAT specific boolean variables to influence the behaviour of the EtherCAT configuration within the application:

Support of optional devices

The lack of EtherCAT devices, which are configured within the application, will cause an error during the start up of the field bus leading to an abortion of the stack. By setting
<instance name of Ethercat master>.StartConfigWithLessDevice := TRUE;
at the beginning of the first PLC cycle missing devices may then be treated as optional devices that do not influence the correct start-up of the field bus any longer.

Disabling of extra output message

To switch the outputs as fast as possible the EtherCAT master will send a message separately for each single task. If for example a connected drive synchronizes to the real time output data, the bus cylce task should be the only one to be allowed to set the outputs, as the multiple messages would disturb the synchronization. To disable extra task messages you have to set
<instance name of Ethercat master>.EnableTaskOutputMessage := FALSE;
once in the first PLC cycle.

Configuration dialogs

By default the Device Editor for the **EtherCAT Configuration** can be opened by a double-click or by using command 'Edit object' when the Master or Slave entry is selected in the devices tree.

Notice the Device Editor Options for general editor settings: For example generic dialogs might be set invisible.

Notice also, that the **buscycle task** is defined in the PLC Settings.

An EtherCAT configuration dialog always is entitled with the Master or Slave name and can provide tabs containing the following sub-dialogs:

For an EtherCAT Master:

- Master
- EtherCAT Configuration / Master
- EtherCAT I/O Mapping / Master
- Status
- Information

For an EtherCAT Slave:

- Slave
- FMMU/Sync
- Expert Process Data
- Process Data
- Startup parameters
- EtherCAT Configuration / Slave
- EtherCAT I/O Mapping / Slave
- Status
- Information

Access on EtherCAT configuration by the application:

Instances for the master and the slave will be created automatically and can be used in the application program for example for restart, stop, status check.

- Implicit Instance of the Master
- Implicit Instance of the Slave
- Concatenated List for Checking all Slaves

Further on the EtherCAT library provides function blocks for reading and writing particular parameters also during operation of the bus.

- Function blocks for "CAN over EtherCAT"

EtherCAT Master

Master

This tab of the configuration dialog for an EtherCAT Master provides the main Master settings as defined in the device description file.

EtherCAT masters do not require a dedicated communication processor and can be implemented without special active plug-in card by just using a passive **NIC card** or the **on-board Ethernet MAC** (Media Access Controller).

The insertion of an EtherCAT Master device automatically brings the insertion of the EtherCAT_Master task in the Task Configuration of the currently active application.

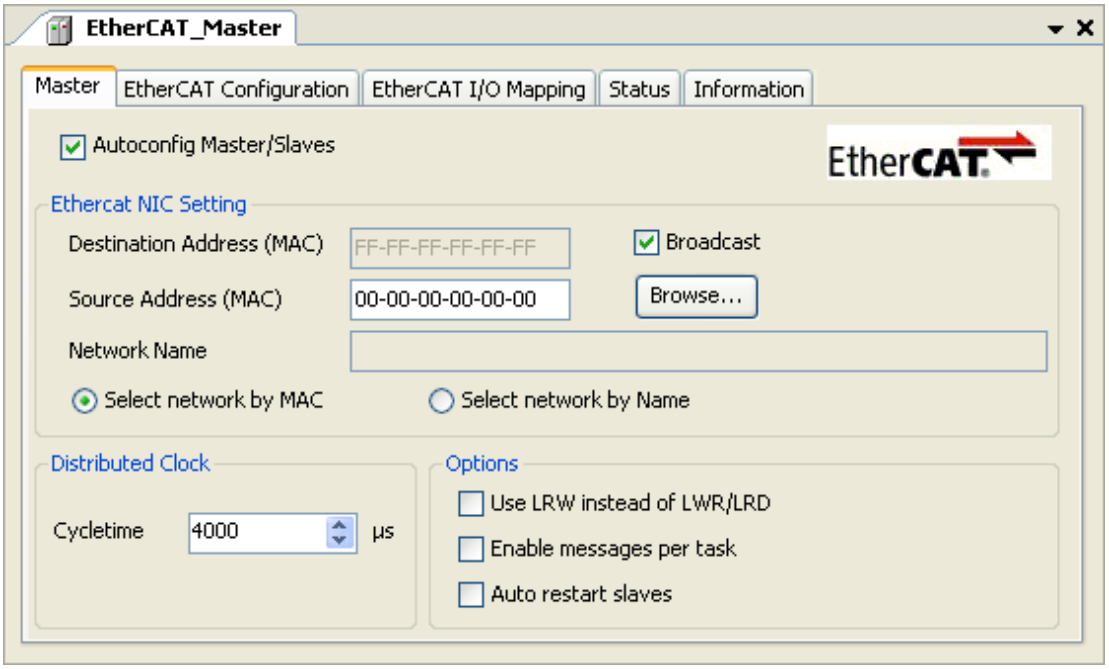


Abb. 494 EtherCAT Master configuration dialog

Autoconfig Master/Slaves: If this option is activated, the major part of the Master and Slaves configuration will be done automatically, basing on the description files and implicit calculations. In this case the FMMU/Sync settings dialog will not be available.

However despite the autoconfiguration mode of the master, for each particular slave an "expert mode" can be activated, allowing the manual modification of the automatically generated process data configuration.

Note: The Autoconfig mode is activated by default and usually sufficient and strongly recommended for standard applications. If the option is deactivated, all configuration settings for the Master and the Slave(s) will have to be done manually and expert knowledge is required. For the configuration of a slave-to-slave communication the autoconfig option has to be deactivated..

EtherCAT NIC Setting

Destination Address (MAC): MAC address of the participant of the EtherCAT network which should receive the telegrams. If option **Broadcast** is activated, no specific address must be entered.

Source Address (MAC): resp. **Network Name**: MAC address resp. name of the card, i.e. PLC (target system)

For this choose one of the following options

Select network by MAC

Select network by Name

Before setting one of the options consider the following: Each network adapter has an **unique MAC-ID**. So, if you now specify the network by MAC-ID, you will not be able to use this project also on another device. If you want to get your project independent from the device, better specify the network by its name.

In both cases the **Browse** button can be used to get the currently available target devices' MAC-IDs resp. names.

Distributed Clock

Master Cycle [µs]: Period of time after which a new data telegram is to be sent on the bus. If the "Distributed Clock" functionality is activated, the master cycle time will be transferred to the slave clocks. Thus an accurate synchronization of data exchange can be reached, which is particularly important in cases where spatially distributed processes require simultaneous actions (for example in applications where several servo axes carry out coordinated movements simultaneously). So a very precise network-wide timebase with a jitter of significantly less then 1 microsecond can be achieved.

Diagnostics Message: In online mode the startup run will be watched and the respective diagnostic messages will be displayed. If message "All slaves done" appears, the configuration is finished and the bus is running.

Options

Use LRW instead of LWR/LRD : Activation of this option enables slave-to-slave communication. Instead of a separate read (LRD) and write commands (LWR), combined read/write commands (LRW) will be used.

Enable messages per task : If this option is activated, read and write commands, that is handling input and output messages can be done with different tasks.

Auto restart slaves: If this option is activated, the master always will try to restart the slaves as soon as the communication aborts.

Master Setting

These settings can only be edited if the Autoconfig mode is deactivated (see above), otherwise they will be done automatically and will be invisible in this dialog.

Image In Address: First logical address of the first slave for input data.

Image Out Address: First logical address of the first slave for output data.

EtherCAT Configuration / Master

This tab of the configuration dialog for an EtherCAT Master provides all master parameters as defined by the device description file.

If the Autoconfig mode is activated these settings are done automatically basing on the given device descriptions and network topology. Normally they **have not to be modified** for standard applications.

The parameters are listed with **Parameter** name, data **Type**, currently set **Value**, **Default Value**, **Unit** and a short Description. The value field might be edited for configuration purposes. However if 'Autoconfig' mode is active, modifications will be without any effect!

See DeviceEditor / Configuration for a general description on the **Configuration** tab of an device editor .

Ethercat I/O Mapping / Master

This tab of the configuration editor for an EtherCAT Master provides the possibility to assign project variables to the EtherCAT outputs resp. inputs. Thus the PLC, which is connected to the EtherCAT Master, can be controlled by the application program.

For a description on how to do the mapping please see DeviceEditor / IO Mapping.

The automatically generated Instance of the Master, which can be used in the application, is shown in the lower part of the dialog (IEC Objects).

Status

This tab of the configuration editor for an EtherCAT device provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog is provided on a tab of an EtherCAT Master or Slave configuration dialog. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

EtherCAT Slave

Slave

This tab of the configuration editor for an EtherCAT Slave provides the main slave settings as defined in the device description file:

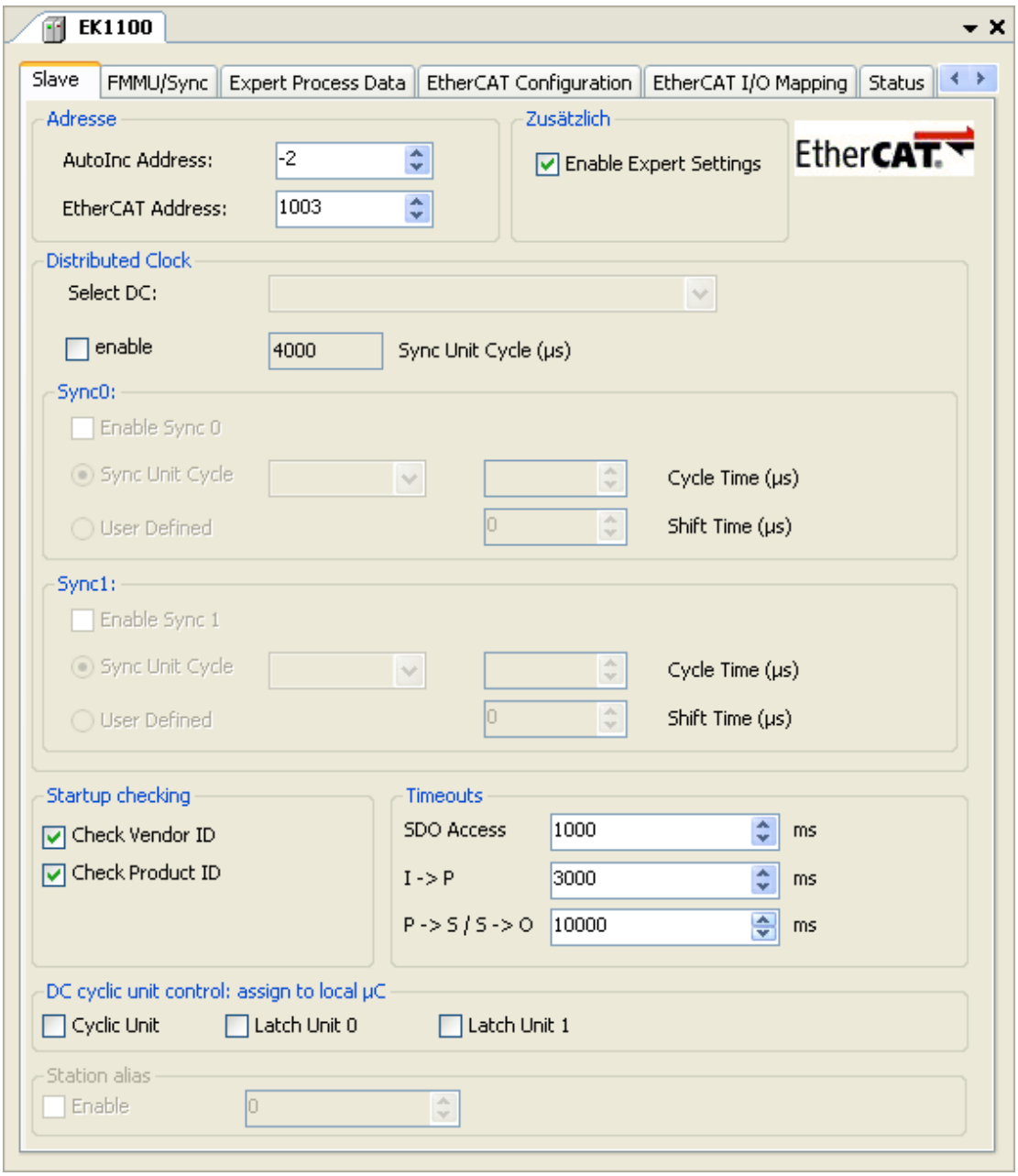


Abb. 495 EtherCAT Slave configuration dialog

Address

The following fields are only available, if the Autoconfig Master/Slaves option is not activated:

AutoInc Address: Autoincremental address (16-bit), defined by the position of the slave in the network. This address is only used during startup, when the master is assigning the EtherCAT addresses to the slaves. When for this purpose the first telegram runs through the slaves, each run-through slave increases its AutoInc Address by 1. The slave with address 0 finally will receive the data. Possible input for example "-8".

EtherCAT Address: Final address of the slave, assigned by the master during startup. This address is independent from the position in the network.

Distributed Clock

Select DC: The drop-down menu provides all settings for distributed clocks provided by the device description file.

enable: If the "Distributed Clock" functionality is enabled, the data exchange cycle time, displayed in the **Sync Unit Cycle (µs)** field will be determined by the Master cycle time. Thus the master clock can synchronize the data exchange within the network.

The settings for the handling of the synchronization unit(s) are slave-dependant:

Sync0:

Enable Sync 0: If this option is activated, the "Sync0" synchronization unit (Beckhoff) is used. A synchronization unit describes a set of process data which is exchanged synchronously.

Sync Unit Cycle: If this option is activated, the master cycle time, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The **Cycle Time (µs)** field shows the currently set cycle time.

User Defined: If this option is activated, the desired time in microseconds can be entered in the Cycle Time field.

Sync1:

Enable Sync 1: If this option is activated, the "Sync1" synchronization unit (Beckhoff) is used. A synchronization unit describes a set of process data which is exchanged synchronously.

Sync Unit Cycle: If this option is activated, the master cycle time, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The **Cycle Time (µs)** field shows the currently set cycle time.

User Defined: If this option is activated, the desired time in microseconds can be entered in the Cycle Time field.

Diagnostics

This section is visible only in online mode.

Current State The current state of the slave is displayed. Possible values: Init, Pre-Operational, Safe Operational, Operational.

If the state is "Operational", the slave configuration has been terminated correctly.

Additional

Enable Expert Settings: If this option is activated, additionally expert settings for Startup checking and Timeouts (see below) can be made and the dialog Expert Process Data will be available. Notice however, that for standard applications the autoconfiguration mode, which by default is activated in the master, is sufficient and thus any use of the "expert settings" is not necessary.

Startup checking

By default at startup of the system the **Vendor ID** and/or the **Product ID** will be checked against the current configuration settings. If a mismatch is detected, the bus will be stopped and no further actions will be executed. This serves to avoid the download of an erroneous configuration.

Here you can deactivate the options in order to switch off the check.

Timeouts

By default there is no timeout defined for the following actions. If you however want to watch, whether they exceed a certain time, specify this time here (in microseconds)

SDO Access: Sending SDO list at system startup.

I -> P: Switch from mode "Init" to mode "Preoperational".

P -> S / S-> O: Switch from mode "Preoperational" to "Safe Operational" resp. from "Safe Operational" to "Operational".

DC cycle unit control: assign to local µC

Choose the desired option(s) concerning the distributed clocks functions in order to define, which should be assigned to the local microprocessor. The control is done in register 0x980 in the EtherCAT slave: The possible settings: Cyclic Unit, Latch Unit 0, Latch Unit 1.

Station alias

Only if explicitly supported by the device description of the slave, this setting can be activated. It allows direct assignment of an alias address in order to get the slaves' address independent of its position within the bus.

FMMU/Sync

This dialog will only be provided on a tab of an EtherCAT Slave configuration editor, if the AutoConfig mode in the **master** is deactivated. It shows the FMMUs and Sync Managers of the slave as defined by the device description file. These settings may be reworked, for example for configuring a slave-to-slave communication.

Note: These are expert settings, which usually are not necessary for standard applications!

FMMU:

This table shows the Fieldbus Memory Management Units of the slave which are used for handling the process data. Each the mapping of the logical address (**GlobStartAddr**) on a physical address (**Ph. Start Address**) is defined. Bitwise mapping is possible. New units can be added resp. existing ones can be edited by the **Edit FMMU dialog**, to be opened via the **Add...**resp. **Edit...** button.

Sync Manager

This table shows the **Synchronization Manager(s)** of the slave. For each available Sync Manager **Type** (Mailbox In, Mailbox Out, Inputs, Outputs) i.a. the physical **Start Address**, the **Access** type, the **Buffer** and the physical address, where the **Interrupts** have to be sent to, are defined. New Sync Managers can be added resp. the existing ones can be edited in the dialog **Edit SyncMan**, which is to be opened by button **Add** resp. **Edit**.

Expert Process Data

This dialog will only be provided on a tab of an EtherCAT Slave configuration editor, if the Expert Settings mode for the slave is activated. It provides another and more detailed view of the Process Data which are also displayed in the Process Data tab.

Additionally here the download of the **PDO Assignment** and the **PDO Configuration** can be activated.

Process Data

This dialog is provided on a tab of an EtherCAT Slave configuration editor. It shows the **input and output process data** of the slave as described by the device description file.

The selected inputs (to be read) and outputs (to be written) of the device will be available in the I/O Mapping dialog as outputs and inputs for the PLC to which project variables might be mapped.

In order to modify the current selection, you must first perform a mouse-click on the checkbox before the currently selected data in order to cancel the selection. After that you can set another one.

Startup parameters

This dialog will only be provided on a tab of an EtherCAT Slave configuration editor, if the Expert Settings mode for the slave is activated.

Here for the particular slave you can define parameters for the device, which will be transferred by SDOs (Service Data Objects) at startup of the system. The dialog will be provided only if the device supports "CAN over EtherCAT". The object dictionary containing the necessary data objects either is included in the EtherCAT XML description file or it is available in an EDS-file which is referenced in the XML description file.

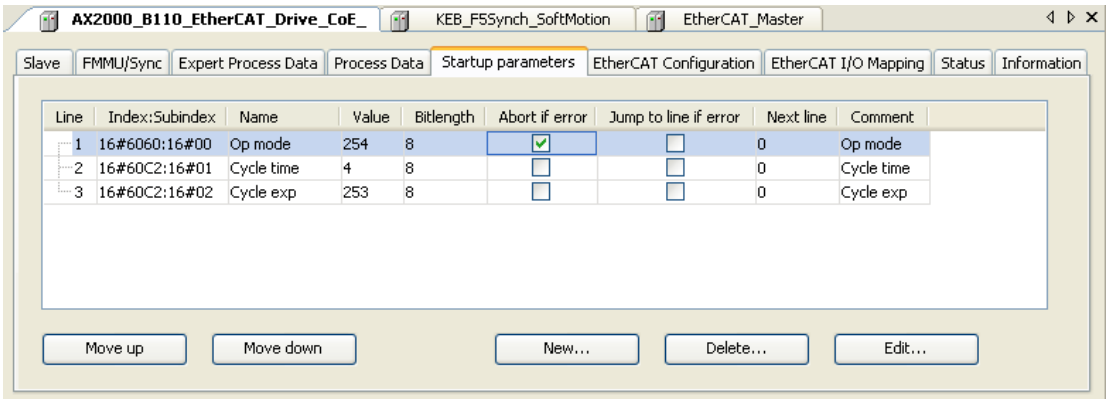


Abb. 496 Startup parameters dialog

CAN over EtherCAT

To add an SDO entry to the configuration table, use button **New...**

The dialog **Select item from object dictionary** will open which offers the available object dictionary entries as defined in the EDS-file.

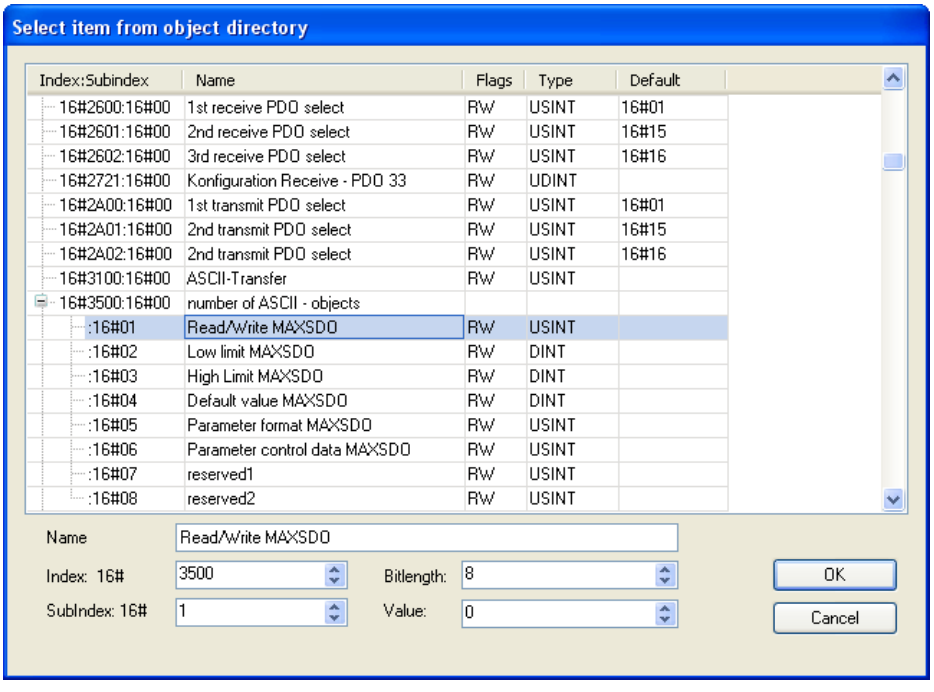


Abb. 497 Dialog - Select item from object dictionary

Before adding an SDO, you might modify its parameters by editing in the fields below the objects table. By defining new index/subindex entries you optionally might add a new object to the SDO, which is not described in the EDS-file. This is useful if the object dictionary is incomplete or not existing at all.

The order (top-down) in the SDO table represents the **sequence** in which the SDOs will be transferred to the module. By buttons **Move Up** and **Move down** the currently selected entry can be shifted within the list.

The **Value** and **Comment** fields also can be edited directly. For this purpose select the field and press <space> or perform another mouse-click.

You can define the behaviour in case of an error during SDO transfer by activating one of the following settings :

Abort if error: If an error is detected, the transfer will be terminated with error state.

Jump to line if error and **Next line**: If an error is detected, the transfer will be continued with the SDO specified in the line which is entered in field 'Next line' (line number, the line numbers are displayed in the **Line** column)

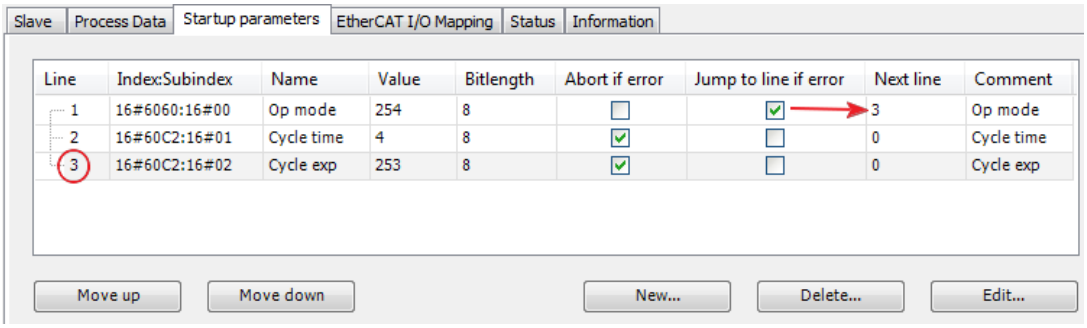


Abb. 498 Configure error handling, example

If supported by the device, the dialog will additionally provide the option to set Complete access. If it is activated, you will have the possibility to read/write not only single sub-indexes, but all subindices of one index of the object dictionary with just one (long) SDO. Enter an array of desired initialization values (for example "1,2,3,4" or "16#54,16#10". As long as complete access is activated, no subindex and bit length can be set, because the subindex always will be 0 and the length will be calculated automatically.

To modify the parameters of a SDO, which is already listed in the SDO configuration table, use button **Edit...** to get the **Select item from object dictionary dialog** where you can edit as described above for adding a SDO.

Button **Delete...** will remove the currently selected entry.

Servodrive over EtherCAT

To add an IDN entry to the configuration table, use button New...

The dialog **Select item from object dictionary** will open which offers the available object dictionary entries as defined in the XML-file.

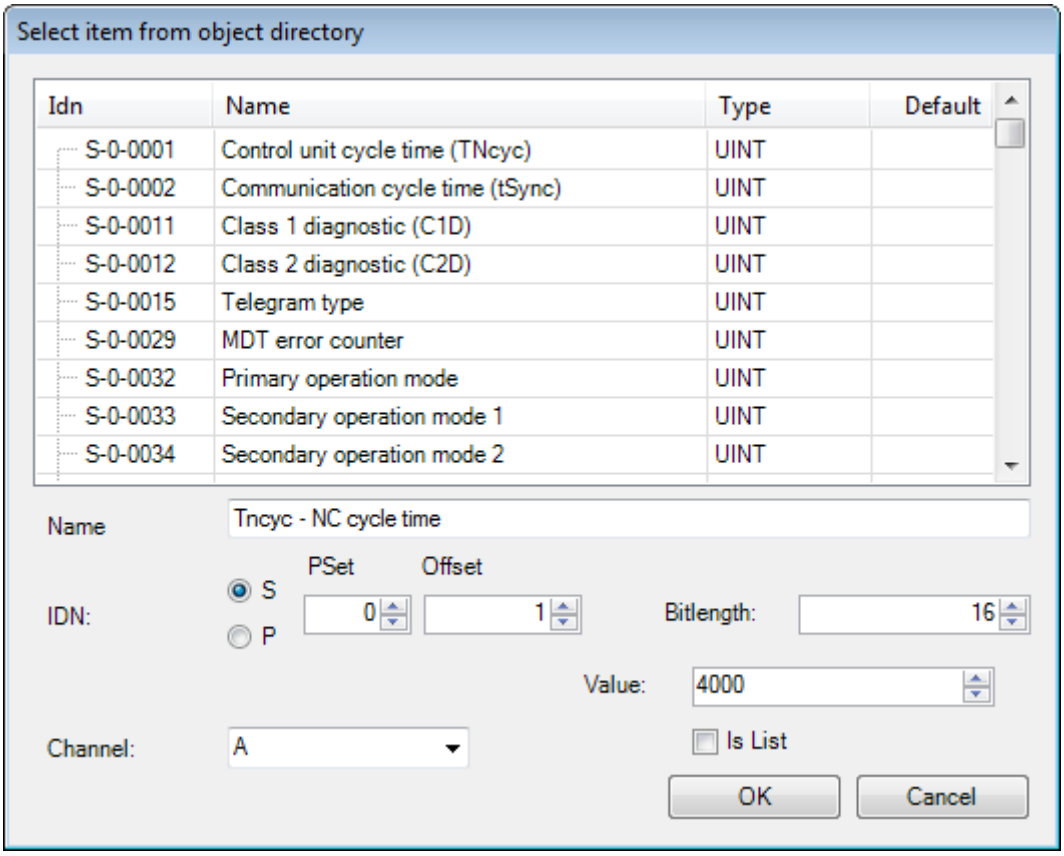


Abb. 499 Dialog 'Select item from object dictionary'

Before adding an IDN, you might modify its parameters by editing in the fields below the objects table. By defining new PSet/Offset entries you optionally might add a new object to the IDN, which is not described in the XML-file. This is useful if the object dictionary is incomplete or not existing at all.

The Channel box is shown automatically if the device has multiple subdevices. Example: A servo controller could drive two servomotors.

The order (top-down) in the IDN table represents the **sequence** in which the IDNs will be transferred to the module. By buttons Move Up and Move down the currently selected entry can be shifted within the list.

The Value and Comment fields also can be edited directly. For this purpose select the field and press [space] or perform another mouse-click.

You can define the behaviour in case of an error during IDN transfer by activating one of the following settings :

Abort if err: If an error was detected, the transfer will be terminated with error state.

Jump to line if error and Next line: If an error is detected, the transfer will be continued with the IDN specified in the line which is entered in field Next line (line number, the line numbers are displayed in the Line column)

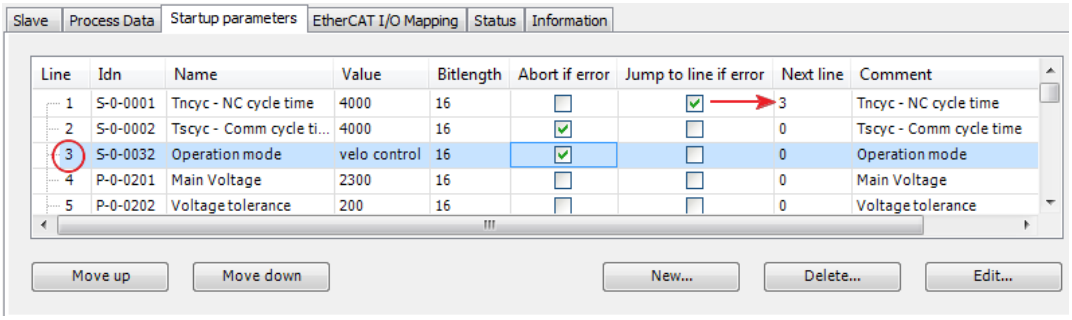


Abb. 500 Example: Configuration of error handling

To modify the parameters of a IDN which is already listed in the IDN configuration table, use button Edit... to get the **Select item from object dictionary dialog** where you can edit as described above for adding startup parameter.

Button Delete... will remove the currently selected entry.

Online

This dialog will only be provided on a tab of an EtherCAT Slave configuration editor, if the ⇘ expert settings mode for the slave is activated and the application is logged in to the device. It provides a view with status information of the Slaves and functions to transfer files to the Slaves over EtherCAT (FoE)

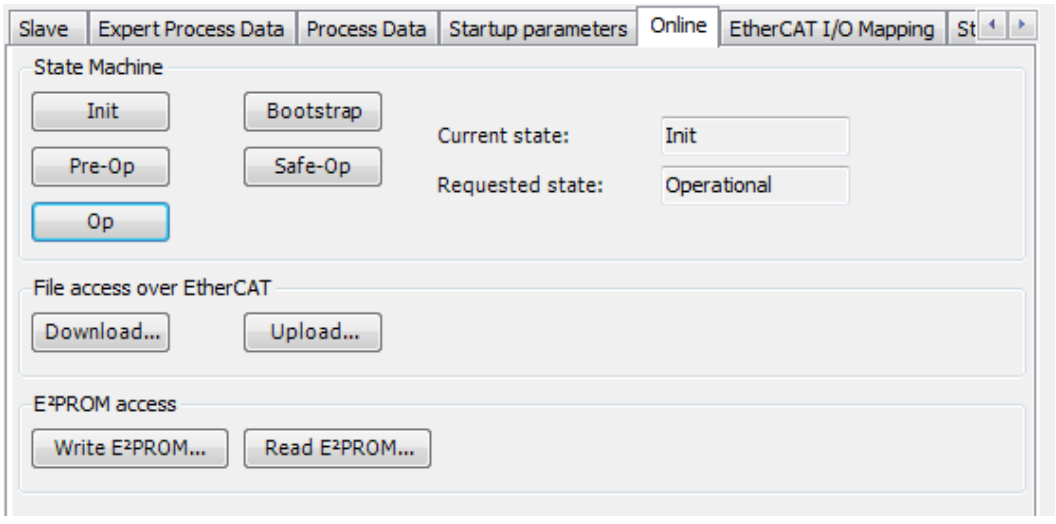


Abb. 501 Dialog for online file transfer

State Machine

The buttons Init, Pre-Op, Op and Safe-Op can be used for debugging purpose.

File access over EtherCAT

If you want transfer firmware files to or from the slave you have click on the Bootstrap button to switch the slave in 'Bootstrap Mode'. The Download and Upload of firmware files can be done with the corresponding buttons.

A dialog to save or open the firmware file will open. In this dialog a string and password is required to execute the file transfer. This information is provided by the slave device and documented in the datasheet of the slave.

E²PROM access

The slave configuration can be read from the E²PROM or write to the E²PROM. Here as well as for the firmware transfer a dialog to open or save files will open.

EtherCAT Configuration / Slave

This tab of the configuration editor for an EtherCAT Slave provides all slave parameters as defined by the device description file.

If the Autoconfig mode is activated in the master, these settings are done automatically basing on the given device descriptions and network topology. Normally they **have not to be modified** for standard applications.

The parameters are listed with **Parameter** name, data **Type**, currently set **Value**, **Default Value**, **Unit** and a short Description. The value field might be edited for configuration purposes. However if Autoconfig mode is active, modifications will be without any effect!

See DeviceEditor / Configuration for a general description on the **Configuration** tab of an device editor .

EtherCAT I/O Mapping

This tab of the configuration editor for an EtherCAT Slave provides the possibility to assign project variables to the EtherCAT outputs resp. inputs. Thus the PLC, which is connected to the EtherCAT slave, can be controlled by the application program.

For a description on how to do the mapping please see DeviceEditor / IO Mapping.

Status

This tab of the configuration editor for an EtherCAT device provides status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog is provided on a tab of an EtherCAT Master or Slave configuration dialog. If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version number, Categories, Order Number, Description, Image.

Access on EtherCAT Configuration via Application

Implicit Instance of the Master

For each EtherCAT Master implicitly an instance of type "IoDrvEtherCAT" will be created as soon as a master device has been inserted in the devices tree. The name of the instance is identical with the device name used in the tree. Information on the availability of this instance is displayed in the lower part of the I/O Mapping dialog.

The master instance can be used in the application for accessing the EtherCAT bus during runtime; for example for a restart of the master or for watching the state of particular slaves. For the latter see: Concatenated List for Checking all Slaves.

The following parameters can be used for this purpose according to syntax "<instancename>.<parameter>":

Input parameters:

xRestart	On a rising edge the complete master will be restarted. All configuration parameters will be re-loaded.
xStopBus	When this parameter is TRUE, the communication is stopped. No further EtherCAT telegrams will be sent. On most of the devices afterwards a restart is necessary, because they have gotten into error state.

Output parameter:

xConfigFinished	If this parameter is TRUE, the transfer of all configuration parameters has been finished successfully. The communication is running.
xDistributedClockInSync	If distributed clocks are used, the PLC will be synchronized with the first EtherCAT Slave for which the DC setting is activated. The output gets TRUE as soon as this synchronization has been finished successfully. This signal for example can be used to start SoftMotion function blocks not before the PLC is running in synchronized mode, because otherwise position jumps could occur. At the start of the PLC the output is FALSE and will change to TRUE after a few seconds. If the synchronicity gets lost due to any faults, the output will be reset to FALSE.
xError	This output will get TRUE if an error is detected during the start of the EtherCAT stack or if during operation the communication with the slaves gets interrupted because no more messages can be received (for example due to a cable break). The error cause can be made out with the help of the logging list resp. via the error string. Thereby "Ethercat_Master.LastMessage.Ethercat_Master" is the instance of the master.

Examples for use of the instance:

EtherCAT_Master.xRestart := xRestart;	cause restart of the master
EtherCAT_Master.xStopBus := xStop;	stop communication on the bus
EtherCAT_Master(); xFinish := EtherCAT_Master.xConfigFinished;	call the master in order to get information on the success of the configuration parameters download

Properties of the Master:

StartConfigWithLessDevice	This property can be used to influence the start behaviour of the stack. The EtherCAT stack normally stops as soon as for example five servo controller are configured in the project, but only three are connected. If however StartConfigWithLessDevice := TRUE in the first cycle, the stack nevertheless will try to start. Thus for example it is possible to set up an universal configuration of 10 servo controllers but to keep the actual number of connected controllers variable. Regard that the Vendor ID and Product IP of each particular slave will be checked anyway. If a mismatch is detected, the start of the stack will be stopped.
NumberActiveSlaves	This property returns the number of the actually connected slaves. If **StartConfigWithLessDevice** is TRUE, then the number of real devices can be determined.
EnableTaskOutputMessage	EtherCAT messages normally are sent by the bus cycle task and additionally by each task which is using slave outputs. In the bus cycle task all outputs are written and all inputs are read. In the other tasks the outputs will be transferred once more in order to get them immediately written to the respective slaves. Thereby it is tried to minimize the dead time until writing as far as possible. In association with distributed clocks this might result in problems in some devices, if for example servo controllers are not synchronized with the **Syncinterrupt**, but are using the point of time of writing for internal synchronization. In this case multiple write accesses might occur within one cycle. If **EnableTaskOutputMessage** is set to FALSE, only the bus cycle task will be used. Additional tasks will not effect further messages. Default: TRUE.
ConfigRead	If this property returns TRUE, the reading of the configuration has been finished and the user may edit the settings, for example in order to add self-defined SDOs.
LastMessage:	This property returns a string with the latest message of the EtherCAT Stacks. If the start has been done successfully, „All slaves done“ should be returned. The string is the same as used for the diagnostic message shown in the EtherCAT Master device editor in online mode

Alternatively, as far as supported by the device, the interface provided by the library IIoDrvBusControl.library can be implemented to provide access on a EtherCAT device from out the application.

Implicit Instance of the Slave

For each EtherCAT Slave implicitly an instance of type "ETCSlave" will be created as soon as the slave device is inserted in the devices tree.

The name of the instance is identic with the device name used in the devices tree. Information on the availability of this instance is displayed in the lower part of the I/O Mapping dialog.

The slave instance can be used in the application for getting, switching and checking the slave status during runtime. See also: Concatenated List for Checking all Slaves.

The following parameters can be used for this purpose according to syntax "<instancename>.<parameter>":

Input parameter:

xSetOperational	On a rising edge it will be tried to switch the slave back to the ETC_SLAVE_ OPERATIONAL mode.

Output parameter:

wState	The current state of the slave is returned. Possible values: 0: ETC_SLAVE_BOOT 1: ETC_SLAVE_INIT 2: ETC_SLAVE_PREOPERATIONAL 4: ETC_SLAVE_SAVEOPERATIONAL 8: ETC_SLAVE_OPERATIONAL

Status ETC_SLAVE_OPERATIONAL means that the configuration has been finished successfully. In case of errors during the configuration the device might fall back to previous states.

Alternatively, as far as supported by the device, the interface provided by the library IIoDrvBusControl.library can be implemented to provide access on a EtherCAT device from out the application.

Properties of the Slave:

VendorID	After the start of the EtherCAT stack, this property returns the Vendor ID read from the device.
ConfigVendorID	This property reads the Vendor ID from the configuration.
ProductID	After the start of the EtherCAT stack this property returns the Product ID read from the device.
ConfigProductID	This property reads the Product ID from the configuration.
SerialID	After the start of the EtherCAT stack this property contains the serial number of the device.

If in the expert settings the check of the Vendor resp. Product ID is activated, the starting of the stack will stop as soon as there is a mismatch between VendorID and ConfigVendorID resp. between ProductID and ConfigProductID.

Concatenated List for Checking all Slaves

For each EtherCAT Master and EtherCAT Slave implicitly an instance of a function block will be created, which for example may be used to watch the state of particular slaves. For this purpose the instance must be called in the application program and the slave state can be read from wState. In order to simplify the programming, all masters and slaves can be retrieved via concatenated lists. Thus all slaves can be checked by a simple "WHILE" loop.

For the master as well for a slave there are the properties **NextInstance** and **LastInstance**. Herewith a pointer on the next resp. previous slave is returned. For the master additionally the property **FirstSlave** is available, providing a pointer on the first slave.

The following example shows how to check all slaves:

Example

Declaration:

```
pSlave: POINTER TO ETCSlave;
```

Program:

```
pSlave := Ethercat_Master.FirstSlave;
WHILE pSlave <> 0 DO
pSlave^();
IF pSlave^.wState = ETC_SLAVE_STATE.ETC_SLAVE_OPERATIONAL THEN
;
END_IF
pSlave := pSlave^.NextInstance;
END_WHILE
```

First via Ethercat_Master.FirstSlave the first slave of the master is retrieved.

In the WHILE loop the respective instance is called and thereby wState is determined. Then the state can be checked. With pSlave^.NextInstance the pointer on the next slave gets retrieved. At the end of the list the pointer gets Null and the loop will be exited.

IODrvEtherCAT library function blocks for "CAN over Ethercat"

Function blocks for CAN over EtherCAT

The EtherCAT library IODrvEtherCAT.library, which automatically will be included in the project when an EtherCAT configuration is supported, i.a. contains function blocks for reading and writing parameters. Thus the particular parameters can be checked and modified also in online mode.

Multiple function blocks can be active at the same time. Internally the particular requests will be handled in a queue and get processed successively.

See the following modules:

ETC_CO_SdoRead (reading a parameter which might be longer than 4 Bytes)

ETC_CO_ SdoRead4 (reading a parameter which is not longer than 4 Bytes)

ETC_CO_SdoReadDWord (reading a parameter and storing the value to a DWORD)

ETC_CO_SdoRead_Access (reading all entries)

ETC_CO_SdoRead_Channel (reading slave parameters)

ETC_CO_SdoWrite (writing a parameter which might be longer than 4 Bytes)

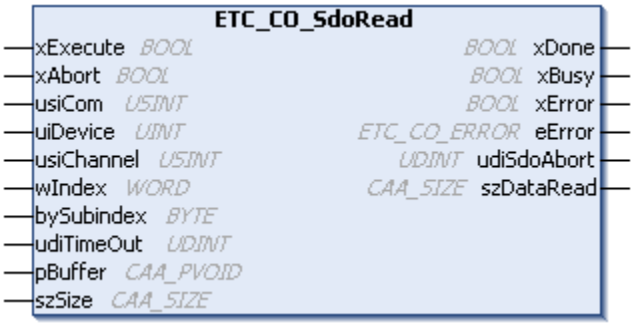
ETC_CO_SdoWrite4 (writing a parameter which is not longer 4 Bytes)

ETC_CO_SdoWriteDWord (writing parameter values given directly in a DWORD)

ETC_CO_SdoWriteAccess (writing slave parameters)

ETC_CO_SdoRead

This function block, which is provided with the library IODrvEtherCAT.library, serves to **read EtherCAT slave parameters**. In contrast to ETC_CO_SdoRead4 also parameters **longer than 4 Bytes** will be supported. The parameter to be read is specified by Index and Subindex as used in the object dictionary.



Input parameters:

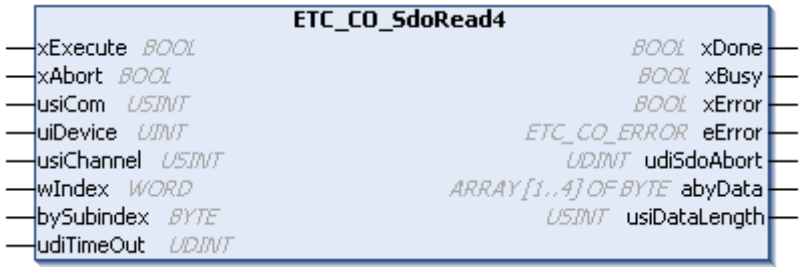
xExecute	On a rising edge at this input the reading of the slave parameter will be started. In order to get the internal channel deallocated, the instance must be called at least once by "xExecute:= FALSE".
xAbort	If this parameter is TRUE, the current read process will be aborted.
usiCom	Number of the EtherCAT Master: If only one EtherCAT Master is used, usiCom is "1". In case of multiple masters use "1" for the first, "2" for the second, etc.
uiDevice	Physical address of the slave. If the autoconfiguration mode in the master is deactivated , an own address can be set in the slave. That arbitrarily chosen address must be entered here. If the autoconfiguration mode is activated, the first slave will get address "1001". The current address of a slave can be checked in the Slave configuration dialog in field „EtherCAT address“.
usiChannel	Reserved for future extensions. Currently not used.
wIndex	Index of the parameter in the object dictionary.
bySubIndex	Subindex of the parameter in the object dictionary.
udiTimeout	Here you can set a timeout in milliseconds. If the reading of the parameter has not been executed within this time, an error message will be generated.
pBuffer	Pointer on a data buffer in which the data will be stored after a successful parameter transfer.
szSize	Size of the data buffer (see above: pBuffer) in Bytes.

Output parameters:

xDone	This output will get TRUE as soon as the reading of the parameter has been finished successfully.
xBusy	This output is TRUE as long as the reading has not yet been finished.
xError	This output will get TRUE, if an error is detected. eError will show the cause.
eError	This output (type ETC_CO_ERROR) shows the cause of the detected error, which is indicated by xError. For example "ETC_CO_TIMEOUT" in case of an timeout.
udiSdoAbort	In case of an error detected in the device, this output will provide more detailed information on the error.
szDataRead	Number of read Bytes; maximum szSize (see input parameters).

ETC_CO_SdoRead4

This function block, which is provided with the library IODrvEtherCAT.library, serves to **read EtherCAT slave parameters**. In contrast to ETC_CO_SdoRead only parameters not longer than **4 Bytes** will be supported. The parameter to be read is specified by Index and Subindex as used in the object dictionary.



Input parameters:

xExecute	On a rising edge at this input the reading of the slave parameter will be started. In order to get the internal channel deallocated, the instance must be called at least once by "xExecute:= FALSE".
xAbort	If this input is TRUE, the current read process will be aborted.
usiCom	Number of the EtherCAT Masters: If only one EtherCAT Master is used, usiCom always is "1". In case of multiple masters use "1" for the first, "2" for the second, etc.
uiDevice	Physical address of the slave. If the autoconfiguration mode in the master is deactivated , an own address can be set in the slave. That arbitrarily chosen address must be entered here. If the autoconfiguration mode is activated, the first slave will get address "1001". The current address of a slave always can be checked in the Slave configuration dialog in field „EtherCAT address“.
usiChannel	Reserved for future extensions. Currently not used.
wIndex	Index of the parameter in the object dictionary.
bySubIndex	Subindex of the parameter in the object dictionary.
udiTimeout	Here you can set a timeout in milliseconds. If the reading of the parameter has not been executed within this time, an error message will be generated.

Output parameters:

xDone	This output will get TRUE as soon as the reading of the parameter has been finished successfully.
xBusy	This output is TRUE as long as the reading has not yet been finished.
xError	This output will get TRUE, if an error is detected. eError will show the cause.
eError	This output (type ETC_CO_ERROR) shows the cause of the detected error, which is indicated by xError. For example "ETC_CO_TIMEOUT" in case of an timeout.
abyData	This is an array of 4 Bytes to which the read parameter data will be copied. If 1 Byte have been read, this will be stored in index 1 of the array. Values of 2 resp. 4 Bytes will be copied to the array according to Intel Byteorder.
usiDataLength	Number of read bytes (1,2,4).

ETC_CO_SdoReadDWord

This function block, which is provided with the library IODrvEtherCAT.library, like function block ETC_CO_SdoRead4 serves to **read EtherCAT slave parameters**. However the read data will be not be copied to an array but to a DWORD (dwData). If byte swapping is necessary, this will be done automatically. Thus the read value can be directly used for further processing.

ETC_CO_SdoRead_Access

This function block, which is provided with the library IODrvEtherCAT.library, like function block ETC_CO_SdoRead serves to **read EtherCAT slave parameters**. However by the additional input xCompleteAccess (BOOL) it allows to read the complete index with all entries. For this purpose xCompleteAccess must be TRUE and bySubIndex must be 0.

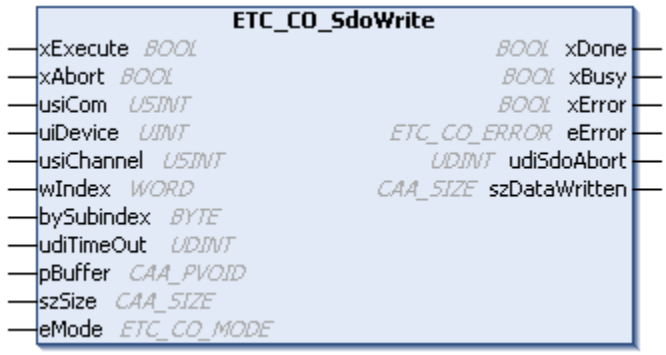
ETC_CO_SdoRead_Channel

This function block, which is provided with the library IODrvEtherCAT.library, like function block ETC_CO_SdoRead_Access serves to **read all EtherCAT slave parameters**.

However there is an additional input **byChannelPriority (BYTE)** for specifying channel and priority within the CoE mailbox message. The first 6 bits of the byte specify the channel, the last 2 bits specify the priority.

ETC_CO_SdoWrite

This function block, which is provided with the library IODrvEtherCAT.library, serves to **write EtherCAT slave parameters**. In contrast to ETC_CO_SdoWrite4 also parameters **longer than 4 Bytes** will be supported. The parameter to be written is specified by Index and Subindex as used in the object dictionary.



Input parameters:

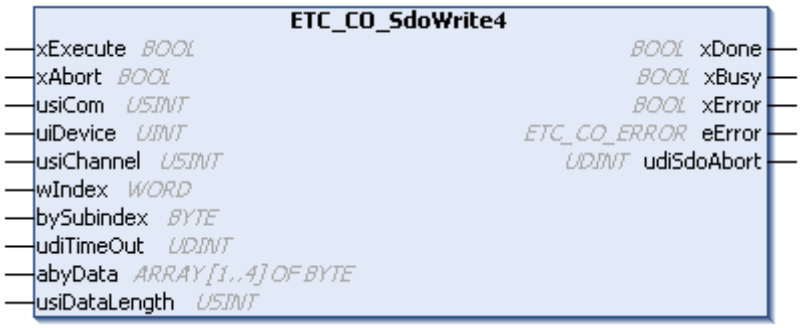
xExecute	On a rising edge at this input the writing of the slave parameter will be started. In order to get the internal channel de-allocated, the instance must be called at least once by "xExecute:= FALSE".
xAbort	If this input is TRUE, the current write process will be aborted.
usiCom	Number of the EtherCAT Masters: If only one EtherCAT Master is used, usiCom always is "1". In case of multiple masters use "1" for the first, "2" for the second, etc.
uiDevice	Physical address of the slave. If the autoconfiguration mode in the master is deactivated , an own address can be set in the slave. That arbitrarily chosen address must be entered here. If the autoconfiguration mode is activated, the first slave will get address "1001". The current address of a slave always can be checked in the Slave configuration dialog in field „EtherCAT address“.
usiChannel	Reserved for future extensions. Currently not used.
wIndex	Index of the parameter in the object dictionary.
bySubIndex	Subindex of the parameter in the object dictionary.
udiTimeout	Here you can set a timeout in milliseconds. If the writing of the parameter has not been executed within this time, an error message will be generated.
pBuffer	Pointer on a the buffer which contains the data to be written.
szSize	Size of the data to be written, in Bytes.
eMode	This input (enumeration ETC_CO_MODE) defines the number of bytes to be written: Possible values are: ETC_CO_AUTO, ETC_CO_EXPEDITED or ETC_CO_SEGMENTED. Usually the AUTO-mode is used, because then automatically that mode, which matches the data length, will be used.

Output parameters:

xDone	This output will get TRUE as soon as the writing of the parameter has been finished successfully.
xBusy	This output is TRUE as long as the writing has not yet been finished.
xError	This output will get TRUE, if an error occurs . eError will show the cause.
eError	This output (type ETC_CO_ERROR) shows the cause of the detected error, which is indicated by xError. For example "ETC_CO_TIMEOUT" in case of an timeout.
udiSdoAbort	In case of an error detected in the device, this output will provide more detailed information on the error.
szDataWritten	Number of written Bytes; will be set to szSize after successful writing.

ETC_CO_SdoWrite4

This function block, which is provided with the library IODrvEtherCAT.library, serves to **write EtherCAT slave parameters**. In contrast to ETC_CO_SdoWrite only parameters not longer than **4 Bytes** will be supported. The parameter to be written is specified by Index and Subindex as used in the object dictionary.



Input parameters:

xExecute	On a rising edge at this input the writing of the slave parameter will be started. In order to get the internal channel de-allocated, the instance must be called at least once by "xExecute:= FALSE".
xAbort	If this input is TRUE, the current write process will be aborted.
usiCom	Number of the EtherCAT Masters: If only one EtherCAT Master is used, usiCom always is "1". In case of multiple masters use "1" for the first, "2" for the second, etc.
uiDevice	Physical address of the slave. If the autoconfiguration mode in the master is deactivated , an own address can be set in the slave. That arbitrarily chosen address must be entered here. If the autoconfiguration mode is activated, the first slave will get address "1001". The current address of a slave always can be checked in the Slave configuration dialog in field „EtherCAT address“.
usiChannel	Reserved for future extensions. Currently not used.
wIndex	Index of the parameter in the object dictionary.

bySubIndex	Subindex of the parameter in the object dictionary.
udiTimeout	Here you can set a timeout in milliseconds. If the writing of the parameter has not been executed within this time, an error message will be generated.
abyData	This array of 4 Bytes contains the data to be written. The data must be stored according to Intel Byteorder.
usiDataLength	Number of bytes to be written (1,2,4).

Output parameters:

xDone	This output will get TRUE as soon as the writing of the parameter has been finished successfully.
xBusy	This output is TRUE as long as the writing has not yet been finished.
xError	This output will get TRUE, if an error is detected. eError will show the cause.
eError	This output (type ETC_CO_ERROR) shows the cause of the detected error, which is indicated by xError. For example "ETC_CO_TIMEOUT" in case of an timeout.
udiSdoAbort	In case of an error detected in the device, this output will provide more detailed information on the error.

ETC_CO_SdoWriteDWord

This function block, which is provided with the library IODrvEtherCAT.library, like function block ETC_CO_SdoWrite4 serves to **write EtherCAT slave parameters**. However the data to be written data will not be given as an array but as a DWORD (dwData). If byte swapping is necessary, this will be done automatically. Thus the value to be written can be specified directly.

ETC_CO_SdoWriteAccess

This function block, which is provided with the library IODrvEtherCAT.library, like function block ETC_CO_SdoWrite serves to **write EtherCAT slave parameters**.

However by the additional input **xCompleteAccess (BOOL)** the complete index with all entries can be written. If for this purpose xCompleteAccess is set to TRUE, bySubIndex must be 0. A further input, **byChannelPriority (BYTE)** can be used to specify the respective data element (channel and priority) in the CoE Mailbox message.

14.4 Profibus DP Device Editor

Hardware configuration corresponding to the **Profibus DP standard** is supported by the '**Device Editor Profibus**' plug-in .

A Profibus DP system consists of one or several **masters** and their **slaves**. Each slave is provided with a parameter set by its master and supplies data on request of the master. At the initialization of the bus system each master parameterizes the slaves which are assigned to it by the configuration. In a running bus system the master sends and/or requests data to/from the slaves.

In order to be able to insert and configure Profibus devices as objects in the devices tree, the **device description files** which are provided together with the hardware, must be imported via the Device Repository dialog. Device description files for Profibus modules are provided in different languages and by default have extension **GS<country identifier>**. The country identifier for example is "E" for English and "D" for German.

Below a master there can be inserted one or several slaves. Below a Profibus device appropriate I/O-modules can be inserted. Inserting of an object is done via command Add device resp. Insert device or Plug device. Also notice the possibility to temporarily disable a device from participating in network communication.

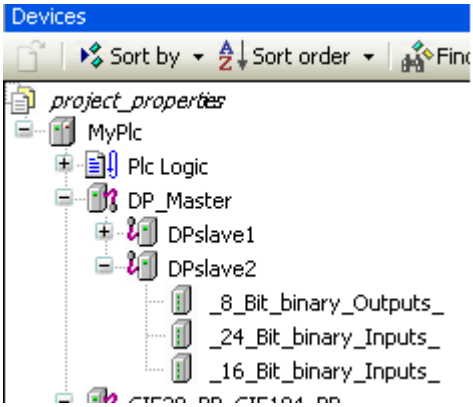


Abb. 502 Example of DP configuration

If a Profibus DP device is selected in the devices tree, the **Device Editor dialog** at least will provide tabs for the 'DP Parameters' and the 'Information' dialogs. The '...Configuration' dialog will only be available if activated in the Device options. The 'I/O Mapping' dialog will only be available for input/output modules, the 'NetX Configuration' dialog only for NetX fieldbus masters.

Thus the following dialogs might be available:

- NetX configuration of a DP Master of a NetX fieldbus
- DP Parameters of a DP Master
- DP Parameters of a DP Slave
- DP Parameters of a DP-Module
- Profibus DP Configuration of a DP Master or DP Slave
- DP-Module Configuration of an input/output module for a DP device
- DP-Module I/O Mapping of an input/output module for a DP device
- Status
- Information

DP Parameters of a DP Master

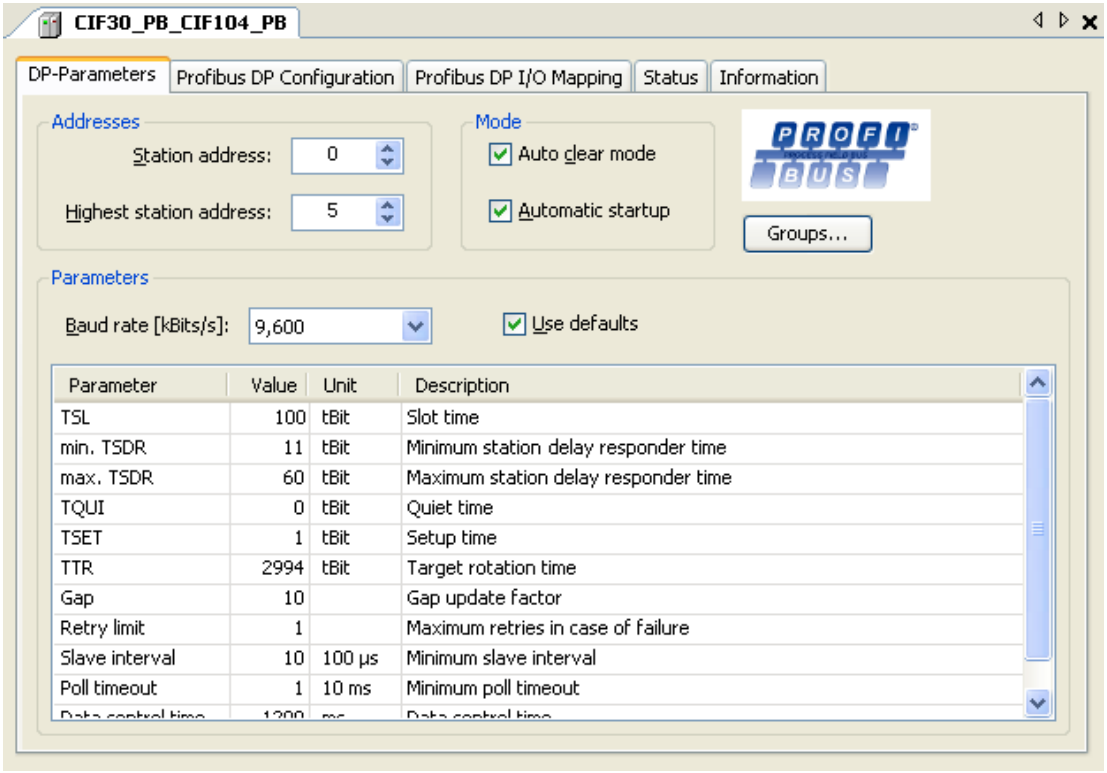


Abb. 503 DP Parameters dialog for a DP Master

The DP Parameters of a DP Master are a standard set of parameters to be configured for each Profibus master. Basically they are defined by the device description file (GS?-file), but might be edited in the DP Parameters dialog by the user.

Addresses

Station address: The allowable range is from 0 to 126. Each device newly inserted on a bus line is automatically provided the next higher address. (note: Address 126 is the default DP slave address). Manual entry is possible; addresses are tested for duplication.

Highest station address: The highest station address (HSA) assigned on the bus is displayed. Here, a lower address can also be entered in order to narrow the GAP range (that is, the address range within which the search for newly-active devices is carried out).

Mode

Auto clear mode: If this option is activated, in error case the outputs of the slaves will be switched to the save state and the master will change from "operate-" to "clear-" state.

Automatic startup: If this option is not activated, the master must be started manually. However it **depends on the driver**, whether the setting is evaluated.

Parameters

These bus parameters describe the timing of the communication.

Baud rate [kBits/s]: The entries given by the device description file (GS?-file) are available here for selection, but only a transmission rate supported by all slaves can be entered.

Use defaults: Activate this option, if the values displayed in the parameter table should be reset to their defaults corresponding to the currently set baud rate.

The parameter table shows for each **Parameter** the **Value**, **Unit** and a short **Description**. The following parameters might be described and can be modified by a click on a selected value field:

TSL [tBit]	Slot time; Maximum time during which the master waits, after sending a request message, for the receipt of the first character of the slave's reply message
min. TSDR **[tBit]**	Minimum station delay responder time; minimum reaction time after which a station on the bus may reply (min. 11 tBit)
max. TSDR **[tBit]**	Maximum station delay responder time; maximum time span within which a slave must reply.
TQUI **[tBit]**	Quiet time; idle period which must be taken into account during conversion of NRZ (Non Return to Zero) signals to other codings (switchover time for repeater)
TSET **[tBit]**	Setup time
TTR **[tBit]**	Target rotation time; token cycle time setting; projected time interval in which a master should receive the token. Result of the sum of the token stop times of all masters on the bus.
Gap	Gap update factor; number of bus cycles after which the master's GAP (address range from its own bus address to the address of the next active station) is searched for an additional, newly inserted active station.
Retry limit	Maximum number of retries (request attempts) by the master in case of failure, that is if it has not received a valid response from the slave.
Slave interval **[µs]**	Minimum slave interval, that is time span between two bus cycles in which the slave can process a request from the master (time basis 100ms). The value entered here must be checked against the respective specifications in the slave's GSD-file.
Poll timeout **[ms]**	Minimum poll timeout; time after which the master's reply by a master-master communication must be retrieved by the requester (Class 2 DP master) (time basis 1 ms).
Data control time **[ms]**	Data control time; Time in which the master reports its status to the slaves assigned to it. At the same time, the master monitors whether at least one data exchange each has taken place with the slaves within this period, and updates the Data_Transfer_List.

tBit: Time unit for transfer of a bit on the PROFIBUS; Reciprocal value of the transmission rate; for example 1 TBit at 12MBaud=1/12.000.000 Bit/sec=83ns

Groups: This button opens the dialog for defining the global **Group Properties**.

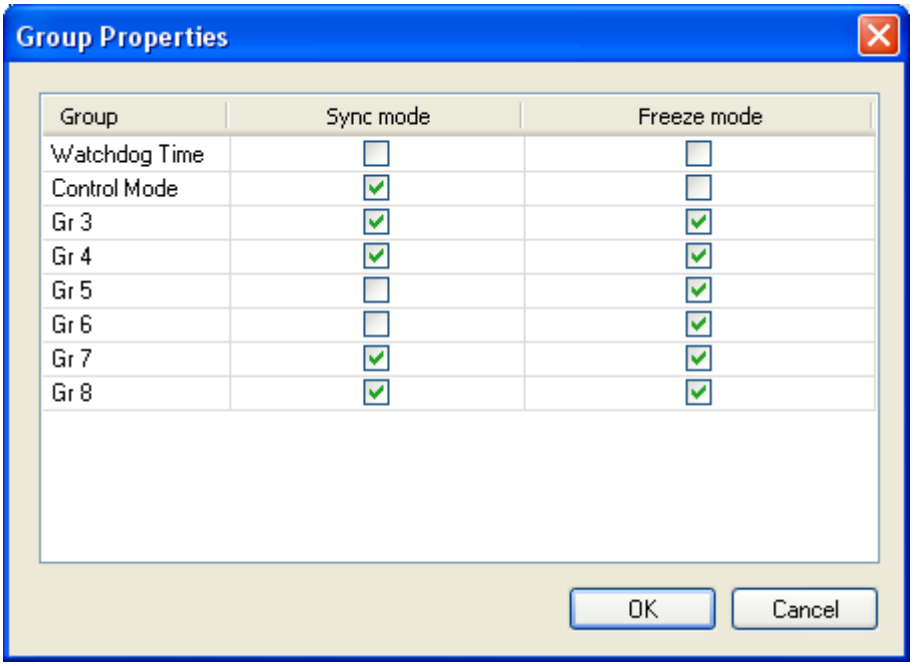


Abb. 504 Group Properties dialog

Up to eight groups can be set up with individual combinations of Freeze mode and/or Sync mode properties, that is activation of Freeze and/or Sync mode for data exchange. The slaves then can be assigned to one or various groups (see DP Parameters of a DP Slave). This enables the master to synchronize the data exchange. By a global control command (Freeze, Sync) it can instruct all slaves of a group at a time.

With a **Freeze** command the master instructs a slave or a group to „freeze" all inputs in their current state and to transfer these data in the subsequent data exchange action. With a **Sync** command the slaves are instructed to synchronously forward all data, which will be received from the master during the subsequent data exchange action, to the outputs.

The **group names** can be edited. For this purpose select the name in the **Group** column and press the <Space> bar to get the string editable. To activate or deactivate the Freeze and Sync option for a group, perform a mouse-click on the checkbox in the respective column. An "x" in the checkbox means that the property is activated.

DP Parameters of a DP Slave

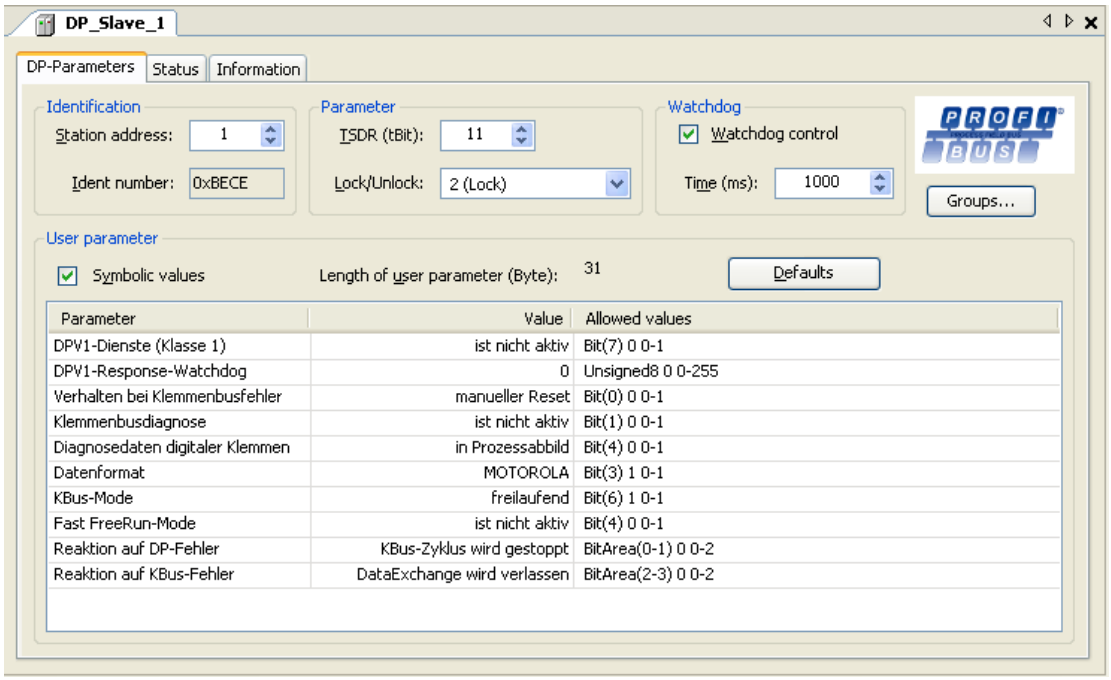


Abb. 505 DP Parameters dialog for a DP Slave

The DP Parameters of a DP Slave in contrast to that of the master are not a standard set but individually defined in the device description file for the particular device. They might be edited in the DP Parameters subdialog of the device editor by the user.

Identification

Station address: see DP Master

Ident number: Unique ident number assigned by the PNO for this device type. Allows unambiguous reference between a DP slave and the corresponding GS?-file.

Parameter

TSDR (tBit): Time Station Delay Responder. Reaction time, that is the earliest time after which the slave is permitted to respond to the master. (min. 11 TBit);

TBit: Time unit for transfer of a bit on the PROFIBUS; Reciprocal value of the transmission rate; for example 1 TBit at 12MBaud=1/12.000.000 Bit/sec=83ns

Lock/Unlock: Slave is locked or released to other masters:

0 (TSDR unlock):	min.TSDR and slave-specific parameters may be overwritten
1 (Will be unlocked):	Slave released to other masters
2 (Lock):	Slave locked to other masters, all parameters are accepted
3 (Unlock):	Slave released to other masters

Watchdog:

Watchdog control: If this option is activated, the watchdog time as defined below will apply. If the slave has not been accessed by the master within this time, it will be reset to its initialization state.

Time (ms): Watchdog time, relevant in case the watchdog control is activated.

User parameters

"User parameters" in extension to the base DP parameters (see above) are various individually parameters of a DP slave. If such are defined in the device description file (GS?-file), they will be displayed here in a table.

The parameter table shows for each **Parameter** the parameter name, the real resp. symbolic **Value** (see above, option Symbolic values) and the **Allowed values** as defined in the device description file (GS?-file). The parameter values entered in the 'Value' column can be edited by a click on a selected value field.

Symbolic values: If symbolic names are also specified for the parameters in the device description file (GS?-file), this option can be activated to get the symbolic values displayed instead of the real values in column 'Value' in the parameter table.

Length of user parameters (Byte): Sum of the data length of all user parameters as defined in the device description file.

Defaults: Activating this button will set the values displayed in the parameter table back to their defaults.

Groups

This button opens the Group Properties dialog. Here the slave can be assigned to one or more of the eight possible groups which serve for synchronized data exchange between master and slave.

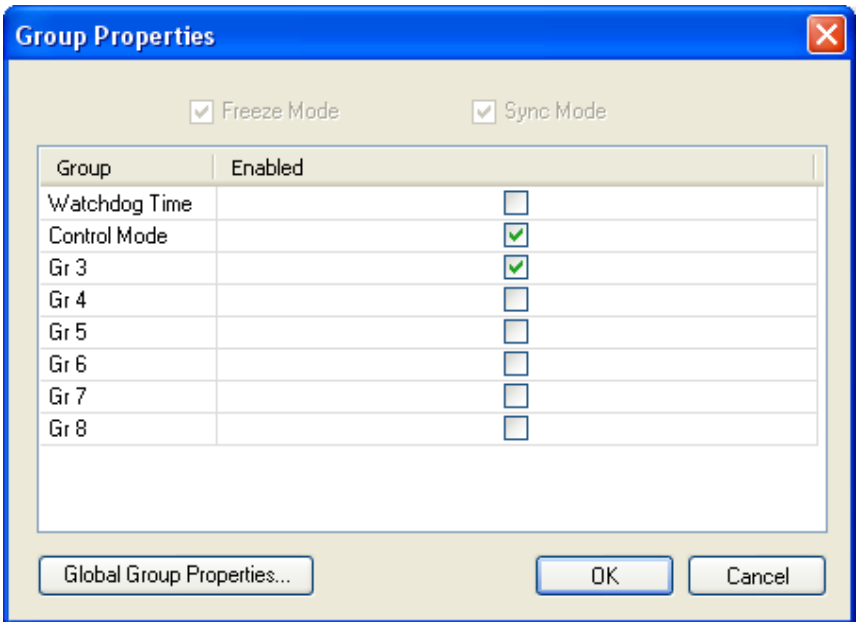


Abb. 506 Group Properties dialog

The global group properties (**Sync. Mode** and/or **Freeze Mode**) are defined in the DP Master configuration (see DP parameters of the DP master, Group properties). This configuration dialog can also be opened from here via the **Global Group Properties** button.

A slave device can only be assigned to those groups whose properties it supports. The concerned properties of each slave (Sync. Mode / Freeze Mode) are displayed above the table, the supported modes being checked.

To assign the slave to a group, click on the checkbox in the **Enable** column of the respective group to set a checkmark. To remove the slave from a group, in the same way you can remove the checkmark.

DP Parameters of a DP-Module

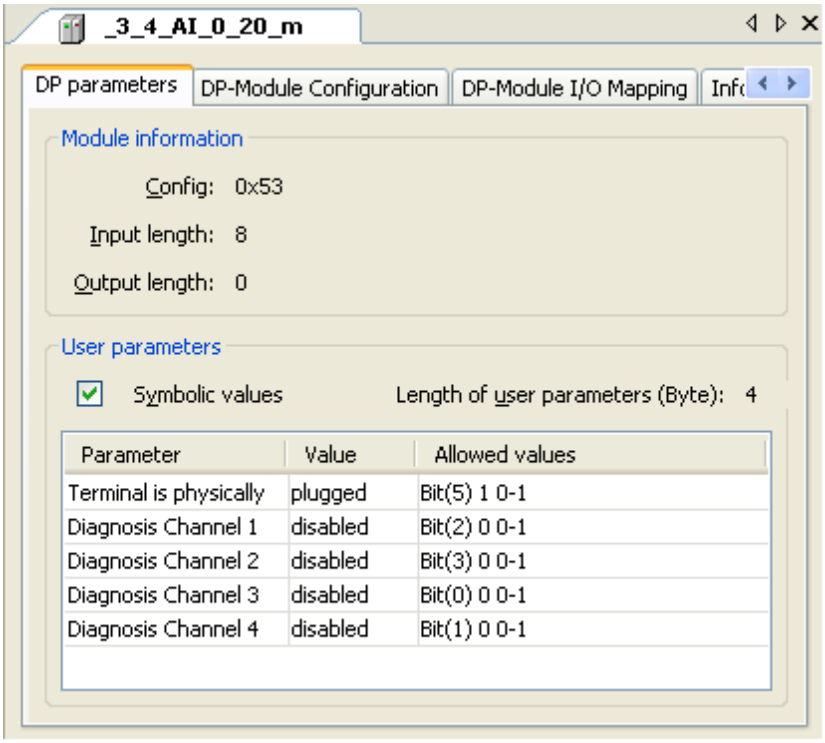


Abb. 507 DP parameters dialog for a DP-Module

The DP parameters dialog of an input/output-module in a Profibus configuration in the Module information section describes the settings for the **Config** (module description coding according to PROFIBUS standard) and the length of the input and output data (**Input length**, **Output length)** in bytes as defined in the device description file (GS?-file).

Further on - if provided with the device description - a set of device-specific **User parameters** is displayed in a table like it is for the DP Parameters of a DP Slave. The option **Symbolic values** allows to switch the display in column 'Value' between symbolic and real values and the complete **Length of user parameters** as given by the device description is indicated in Bytes. The parameter values entered in the 'Value' column can be edited by a click on a selected value field.

Profibus DP Configuration

This dialog is only available in the device editor for a Profibus Master resp. Slave device, if option 'Show generic device configuration views' is activated. It corresponds to the **Configuration** dialog provided by the Device editor also for other device modules.

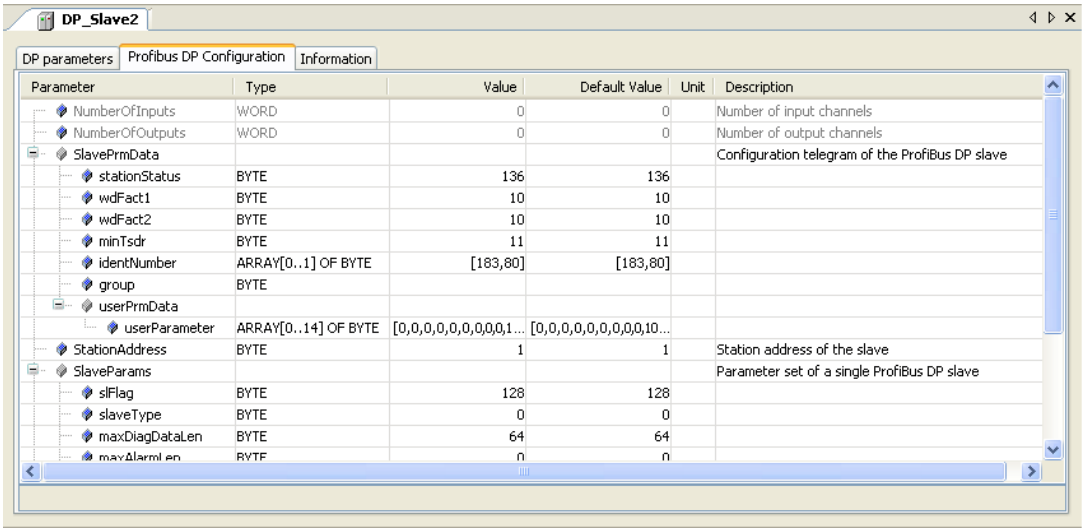


Abb. 508 Profibus DP Configuration dialog

The device-specific master or slave parameters defined by the manufacturer in the device description file (extra, not in GS?-file) will be listed in a table and might be edited there if allowed by the description file. Non-editable parameters are displayed grey-colored.

DP-Module Configuration

This dialog is only available in the device editor of a Profibus module, if option 'Show generic device configuration views' is activated. It corresponds to the Configuration dialog provided by the Device editor also for other device modules.

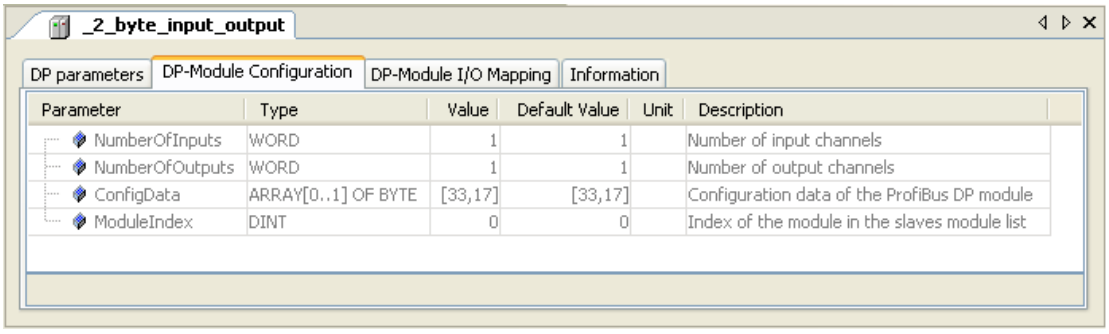


Abb. 509 DP-Module Configuration dialog

DP-Module I/O Mapping

This dialog serves to map the inputs and outputs of a Profibus device to project variables. The handling corresponds to that of the mapping dialog provided by the Device editor also for other device modules.

Profibus DP I/O Mapping Master

This dialog serves to map the inputs and outputs of a Profibus device to project variables. The handling corresponds to that of the mapping dialog provided by the Device editor also for other device modules.

NetX configuration

In case of a NetX fieldbus this dialog is available as one of the tabs in the device editor of the master, for selecting the NetX chip (slot for the card) and the communication channel of this chip. The name of the setting used in the dialog and the possible settings provided by the selection lists are defined by the device description.

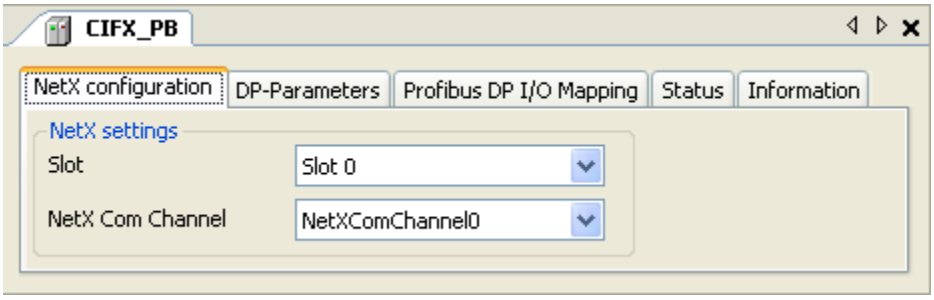


Abb. 510 NetX configuration dialog, example

Slot: Slot to be used. In case of PCI cards with NetX chip the slot numbers usually correspond to the PCI card numbers.

NetX Com Channel: Channel on the card to be used for the communication. A NetX board may have up to four communication channels for different fieldbusses.

Regard in case of a Profibus **NetX** configuration the possibility to check the hardware for the **currently available bus participants** by calling method **GetLiveList** (provided by the automatically included IoDrvCifXProfibus.library). On a rising edge at input xEnable the method requests a current list of all participants connected to the Profibus. The operation has been terminated when output wOpState is HIL_DONE or HIL_ERROR. The method can return the following values: 00 = station not available, 01 = inactive station, 03 = active station.

Status

This dialog is provided on a tab of the **Device** dialog (Device Editor). It shows status information (for example "Running", "Stopped") and device-specific diagnostic messages, also on the used card and the internal bus system.

Information

This dialog is generically provided on a tab of the device editor dialog and shows some general information on the device.

Interfaces associated to Profibus DP master and slave

The libraries listed below provide interfaces for ProfibusDP devices. A driver can implement one of these interfaces in order to equip the user with the related functionality.

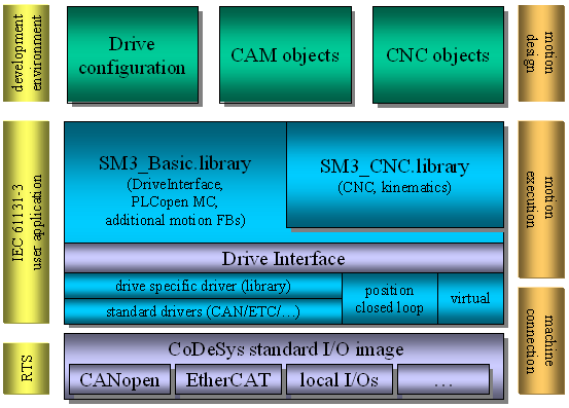
- IIoDrvBusControl.Library
- IIoDrvDPV1C1.Library
- IIoDrvDPV1C2.Library
- IIoDrvProfibus.Library

14.5 SoftMotion Device Editor

14.5.1 Softmotion concept and survey on components

SoftMotion allows you to realize movements efficiently and with no need of detailed know-how: elementary single-axis movements or CAMs as well as complex motions in several dimensions. Mainly applications focusing not only on the motion functionality, but also on sequence and process control or auxiliary functions, fall into the scope of SoftMotion application within the development environment of »PLC Designer«. Whereas the program logic is completely handled by the PLC program, the pure motion information is executed by library functions.

»PLC Designer« SoftMotion is a software package that serves as development and runtime environment for motion controllers. In addition to the standard features of »PLC Designer« and »PLC Designer« SP the following, motion controller specific software parts are contained:



- The **Drive Configuration** provides an editor to import the structure and the configuration of the drive-hardware into the »PLC Designer« user interface. Thereby the functions of the Drive Interface libraries will create IEC data structures as abstract representation of the drives. Automatically, that is without any additional effort by the IEC-programmer, the Drive Interface will communicate with the drives and take care of the topicality of the drive data structures as well as of the transfer of updated data.
  To control the drives the IEC program can access the abstract data structures either by use of standard modules of the SoftMotion libraries (SM3_Basic.library, SM3_CNC.library) or by use of modules created by the IEC programmer for this purpose. Thereby, the target values are prescribed cyclically; this means, that in each IEC task cycle target values (for position, velocity, acceleration etc.) are calculated once and transferred from the Drive Interface to the drives.
  The possibility to "instruct" drives (for example to prescribe only a final position, so that the drive will move on its own initiative and report its arrival) is not provided. The reason is, that, while an instruction is executed, the central controller has no influence on the drives. Therefore it is not possible to for example coordinate the moving of multiple axes.
- The **CAM editor** is an independent plug-in of »PLC Designer« to be integrated into the programming interface. CAMs are used to control multi-axes drives. Within the editor a CAM can be implemented graphically or in table form. Therefrom a global data structure instance containing the description of the object will be created implicitly and forwarded to the application, where it can be accessed by the appropriate POUs.
- The CNC editor is an independent plug-in of »PLC Designer«. It allows to program multidimensional motions, which can be transferred and controlled via the drive interface which does the communication with the drive-hardware. The editor works abutted to the CNC language DIN66025, synchronously in a graphical and a text editor. Beside that the table editor can be used instead of the text editor with a similar range of function. Basically up to 9-dimensioned motions can be realized, whereby three dimensions will be interpolated not linearly. Thus in three dimensions lines, circles, parables, ellipses and splines can be programmed, the other directions are interpolated just linearly. For each path, which has been designed, »PLC Designer« automatically creates a data structure, which is available in the IEC program.
- The library **SM3_Basic.library** is a basic component that has to be included in all SoftMotion applications. It consists in the following components:

  - PLCopen Function Blocks according to the PLCopen standard, which allow to easily realize the controlling of a single axis motion as well as the synchronized motion of two axes. Beside library elements for status check, parameterizing and general operating there are function blocks that actuate axis with respect to defined speed and acceleration parameters. If two axes shall be synchronized, one axis serves as master and controls the second axis (slave) according to a certain prescript. This prescript for example can be a CAM using appropriate POUs to link the slave to the master axis. Further on there are function blocks, which allow electronic gear or phase shifts.
  - Several proprietary motion control FBs
  - DriveInterface basic driver FBs (AXIS_REF, AXIS_REF_VIRTUAL, AXIS_REF_MAPPING)
  - Utility functions: for example for file service or error report
  - Function blocks for the implementation of specific drive drivers: the basic driver FBs can be extended by more specific drivers that can communicate with a certain drive type.
- The library **SM3_CNC.library** is based on the SM3_Basic.library. Beside some function blocks for kinematical transformations it provides all POUs necessary for creating, executing and visualizing CNC motions.
- The **Drive Interface** is the part of the SM3_Basic.library taking charge of the communication between the IEC program and the drives. It provides the possibility to extend the basic driver FBs to specific driver types. These specific driver FBs (named AXIS_REF_*) are typically combined in a separate library that is referred to by the corresponding device description of the device. Additionally, the drive interface includes bus specific libraries to handle the I/O mapping.

Portability

Except for particular drivers, which are serving hardware components directly, all SoftMotion runtime components are programmed in IEC 61131-3. Hence, maximum platform independence is reached.

Note: The 'Communication Settings' and 'Applications' tabs concern the communication with the PLC and are provided by the 'Device Communication Editor' plug-in.

14.5.2 The Softmotion Drive Interface

The Softmotion Drive Interface is a standardized interface allowing to include, to configure and to address the abstract image of a drive hardware within the IEC program. The mapping of diverse hardware onto a general interface not only allows an easy changing of drives and reuse of IEC programs, but also facilitates the communication between the PLC and the drives or between the drives themselves. It is the interface that links the drive to its representation within the application (AXIS_REF) and that is responsible for updating and transferring the motion data necessary for the drive control.

The Drive Interface consists of the following components:

- the device description of SoftMotion devices to be represented in the device tree
- libraries referred by the device description that extend or overload the basic AXIS_REF FB according to the requirements of the specific drive type
- libraries that include FBs for an acyclic reading/writing of data and for wrapping basic functions of the field bus stack

The import of the basic libraries to the library manager results from the use of a **SoftMotion PLC** providing a **SoftMotion General Drive Pool** as well. The drive devices may then be added in two different ways:

- **Configuration as free drive device**
  These drive devices are not fixed to another device in the object tree, but collected in the SoftMotion General Drive Pool each Softmotion PLC is equipped with. Virtual drives for example are free drive devices, as they are not linked to another field bus device; also position controlled drives can be free devices, as there is no obligatory choice for the field bus device providing their in- and outputs.

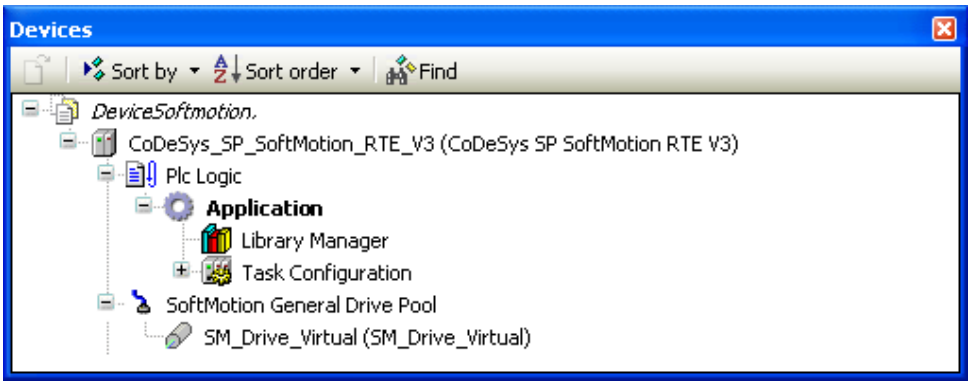


Abb. 511 Exemplary configuration with free drive device

- **Configuration as connected drive device**
  These drive devices stand in a one-to-one correlation with another device representing the drive within the field bus topology. The icons associated to these drive devices are augmented by little signs that shall indicate the related field bus type. Any standard servo drive for example is connected to the controller via a field bus.

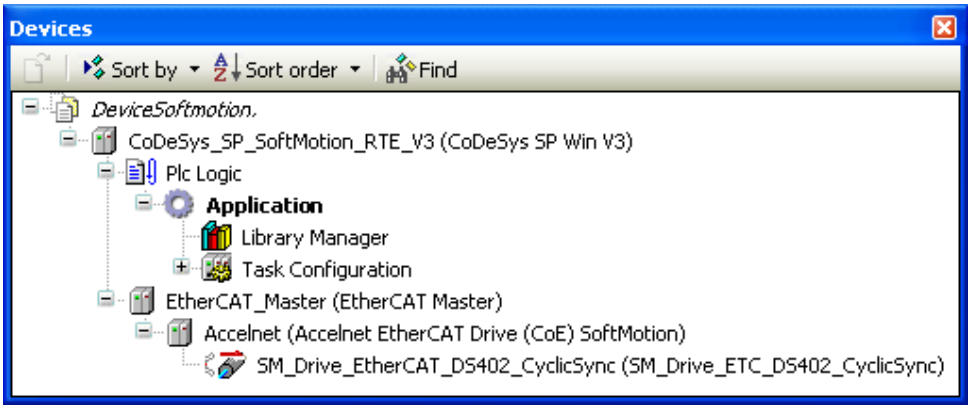


Abb. 512 Exemplary configuration with connected drive device

14.5.3 Configuration within the Device Softmotion Editor

Depending on the specific type of the device the associated Device Softmotion Editor will include different tabs for its configuration. You will find described in detail those that are part of the basic equipment of the Softmotion Editor:

- Softmotion Drive: Basic
- Softmotion Drive: Scaling/Mapping
- Status
- Information

Softmotion Drive: Basic

The first tab of the Device Softmotion Editor provides several boxes for the configuration of the basic settings for the inserted device.

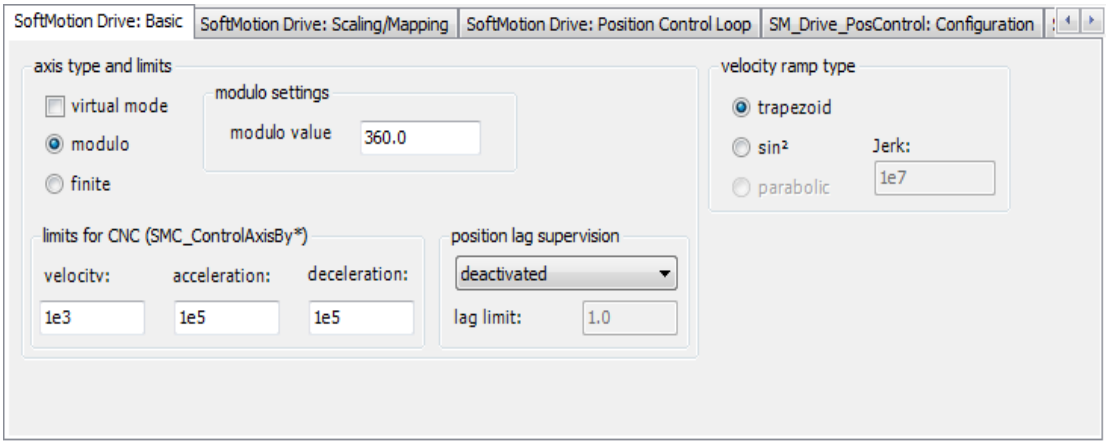


Abb. 513 Device Softmotion Editor - SoftMotion Drive: Basic

axis type and limits:

virtual mode: If you activate the checkbox "virtual mode", the drive will be replaced by a simulation similar to a virtual drive device. Consequently this checkbox will always be activated for virtual drives. In case of connected drive devices this has no influence on the field bus devices that will work as usual, but without receiving messages from or sending messages to the real device.

You may chose between two types of axis: **linear** or **rotatory** (modulo). Depending on this choice the small box on the right will be entitled

- software limits (possible setting for linear drives): activate the checkbox, if you want to limit the position values by a lower (**negative**) or upper (**positive**) bound.
- modulo settings (obligatory setting for rotatory drives): you have to define the modulo range.

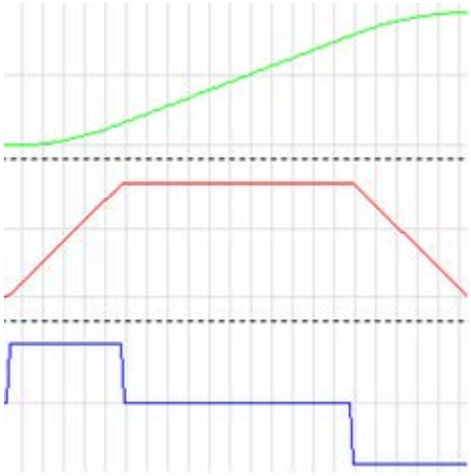
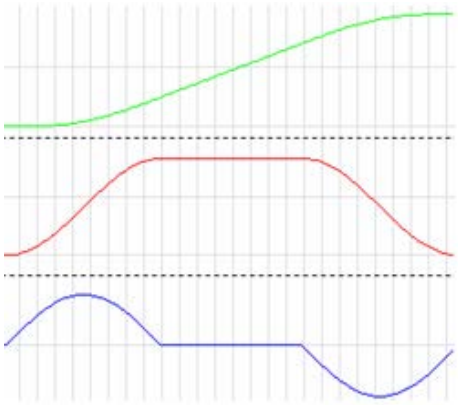
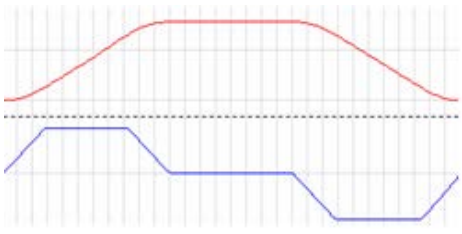
limits for CNC (SMC_ControlAxisBy) :
in this sector you may define limit values (for velocity, acceleration and deceleration) used by the library modules SMC_ControlAxisBy* in order to detect jumps (for example SMC_ControlAxisByPos).

velocity ramp type:

The velocity ramp type defines the velocity profile for the velocity-generating one-axis and the master/slave-modules. The associated libraries will provide up to three different choices: **trapezoid** results in a trapezoid velocity profile (with constant acceleration in each section), **sin²** results in a sin²-velocity profile (with continuous acceleration), **parabolic** results in a parabolic velocity profile (with continuous piecewise acceleration profile).

For the two latter modes the **Jerk** has to be defined in addition.

The following images demonstrate the effect of the different ramp types; the position is displayed in green, the velocity in red and the acceleration in blue.

trapezoid The velocity is piecewise linear, but continuous, whereas the piecewise constant acceleration exhibits jumps.	
sin² The sharp edges of the velocity function are smoothed (by using a sin²-function instead of a line), so that the jumps in the acceleration are eliminated. As the motion sequence is clearly defined, in general the jerk cannot be limited by the user. His limitation for the jerk is valid only in case that the drive has an acceleration different from 0 at the beginning. Then, with respect to the jerk limit, the drive will be run to zero, before the actual movement will be started. Compared to the trapezoid velocity profile decelerating will take more time now.	
parabolic (not yet implemented) The acceleration has a continuous, piecewise linear profile, whose gradient is limited by the jerk. Therefore the velocity has a continuous, piecewise parabolic profile. Actually this is the only profile the jerk can be limited for.	

Softmotion Device: Scaling/Mapping

This page provides the adjustment of the physical device to its representation within the application.

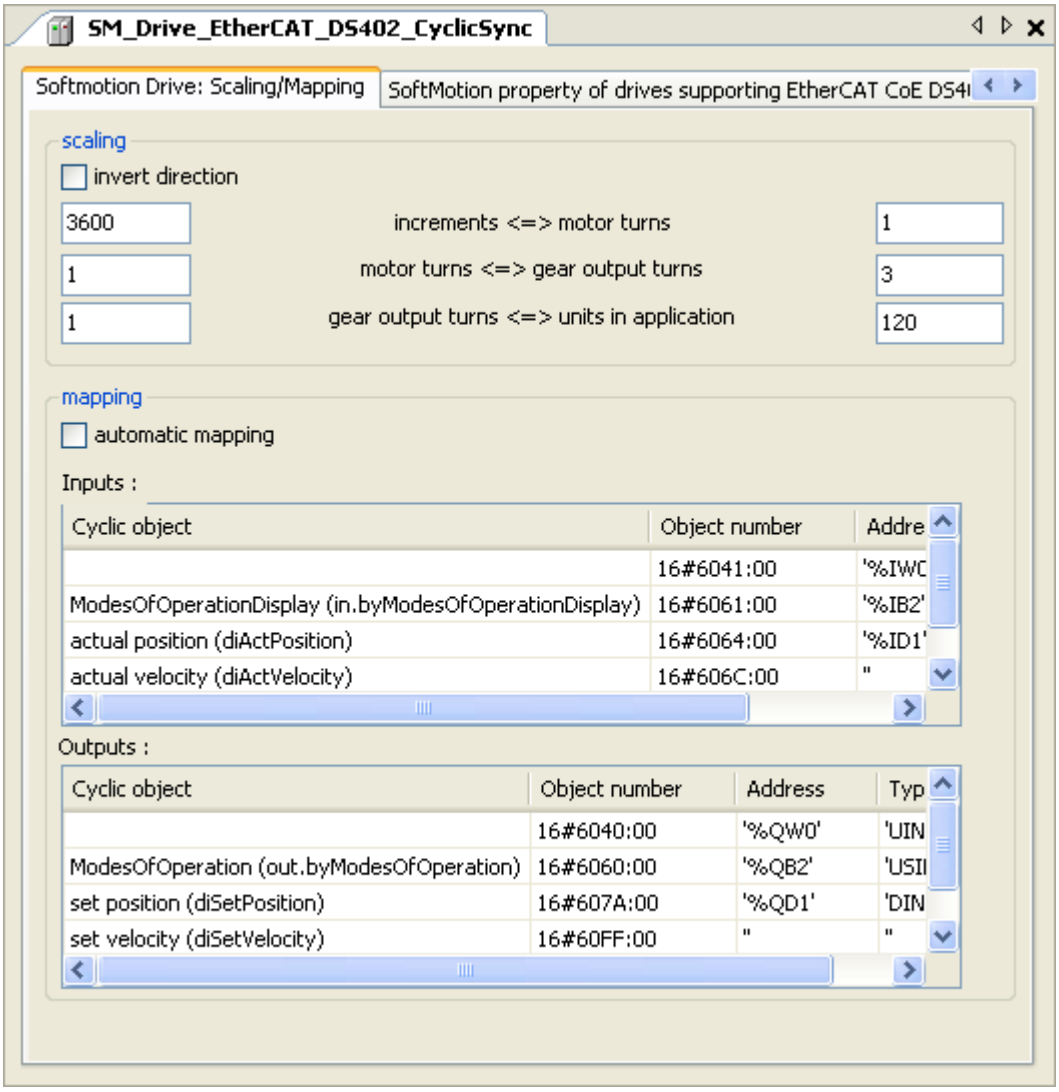


Abb. 514 Device Softmotion Editor - Softmotion Device: Scaling/Mapping

scaling:

You have to enter the multiplying factor defining the conversion between the integer position values received from the device (left entry in the first line) and the technical units used in the IEC program (right entry in the third line). Additionally, by editing the further parameter values, a drive can be taken in to account.
In the exemplary configuration a drive creating 3600 increments for one rotation is scaled such that the technical units used within the application are measured in radian.

mapping:

Using the **automatic mapping** the IEC parameters concerning the drive will be automatically mapped on the appropriate in-/and outputs of the device. After deactivating this checkbox you may edit the mapping manually by modifying address or type of the in- or outputs within the displayed parameter list, which has been adapted from the device description file.

Information

This page displays information on the specific device read out from the device description file.

14.5.4 Configuration Drive Logical

The SoftMotion Drive Logical is a virtual drive under a master device or an encoder.

The axis of the logical drive reflects the movements of its master axis while being a separate instance. This allows to execute the following instructions on it without influencing the master axis: MC_SetPosition, MC_TouchProbe (may be used on the logical axis under the precondition that the touch probe channel is not used by the master axes nor by any other logical axis of this master axis) and MC_AbortTrigger.

The axis type AXIS_REF_LOGICAL_SM3 is part of the library SM3_Basic.The instance of the type is configured with help of the SoftMotion drive logical editor:

- SoftMotion Drive: Logical
- SM_Drive_Logical: Configuration
- SM_Drive_Logical: I/O Mapping
- Status
- Information

SoftMotion Drive: Logical

The first tab of the SoftMotion Drive Editor provides some boxes for configuration.

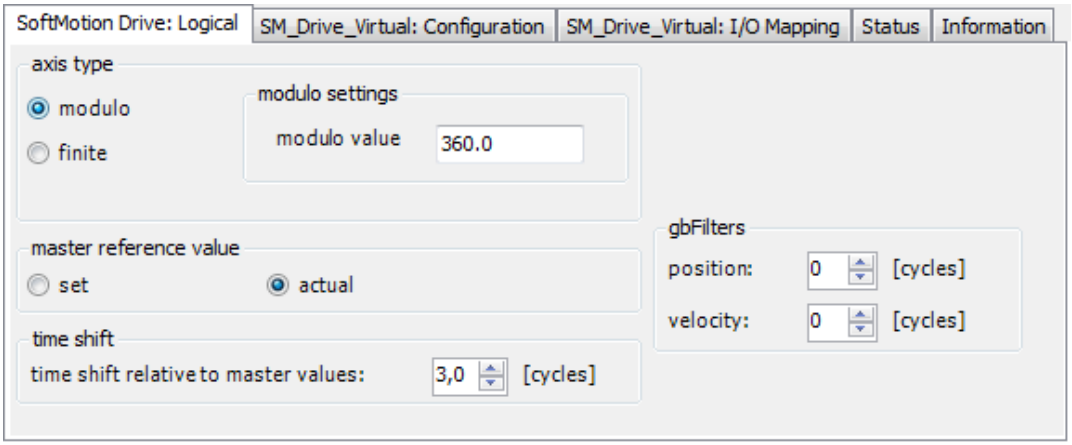


Abb. 515 SoftMotion Drive: Logical, modulo axis

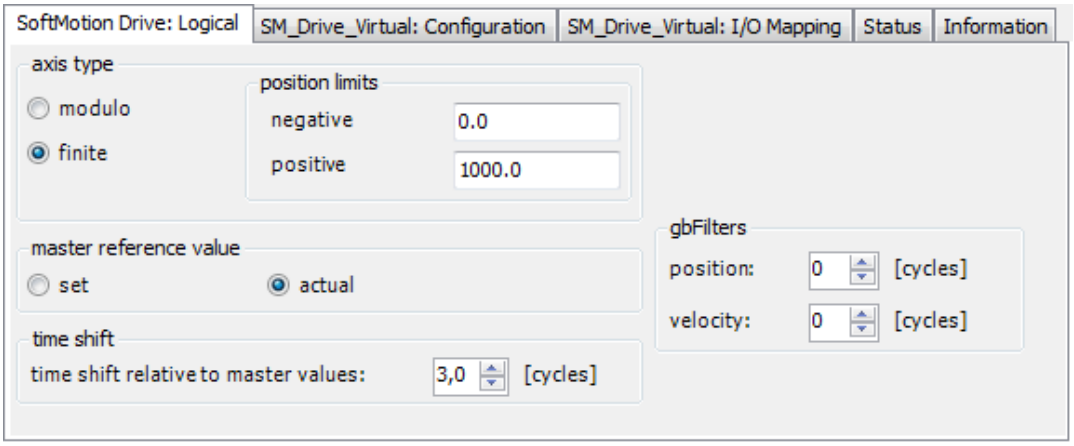


Abb. 516 SoftMotion Drive: Logical, finite axis

axis type:

If the axis type is set to **modulo**, then **modulo settings** are visible with the possibility to set the **modulo value**.

If the axis type is set to **finite**, the negative and positive position limits can be set.

master reference value:

With this radio box you can choice between **set** and **actual** master reference value. Depending on this input either the set value or the actual values of the master are used as input for the logical axis and the shift compensation.

time shift:

Here the number of cycles, which have to be extrapolated, can be set. The number of cycles defines the time gap, who the set values have to be extrapolated.

input data filter:

The filter calculates the moving average of values. Up to the selected number of cycles the associated values are used for the calculation of the average position resp. velocity. Smoothed values are the result. The range is 0 to 9. If the number of cycles is set to 0, the latest value is used as filtered position or velocity. Otherwise a calculated value is used.

SM_Drive_Logical: Configuration

Here the instance of the data structure of the logical axis is listed in tabular view with all the configured settings.

SM_Drive_Logical: I/O Mapping

No mapping is possible.

Status

Here the status information of the logical drive is visible.

Information

This page displays some information on the logical drive read out from the device description file.

14.5.5 Survey on special use cases

The library SM3_Basic.library includes some higher level function blocks that interact with the drive not via the standard set/act position interface, but give commands or modify settings of the drive by reading/writing parameters of the AXIS_REF. Employing these function blocks the following use cases may be handled properly:

Reinitialization of a drive

A modification of drive parameters (like scaling factors, position period of rotary drives etc.) during runtime necessitates a reinitialization of the drive. Calling the function block SMC3_ReinitDrive with a rising edge at input Execute will reset the parameter wCommuncationState of the drive and repeat the complete start-up process for the drive.

Enabling/Disabling a drive

The AXIS_REF parameters bRegulatorOn and bDriveStart control the power stage and the brake of a drive; they can be influenced by use of the function block MC_Power.

To enable the power stage the parameter bRegulatorOn has to be set; by bRegulatorRealState the drive reports the current state of this process. If supported by the drive device, a reset of bRegulatorOn during a move will suspend the torque immediately .

The variable bDriveStart determines whether the drive is allowed to move or not; by bDriveStartRealState the current state of this parameter is reported. If supported by the drive, a rest of the parameter bDriveStart during a move will decelerate the drive according to the configured ramp; thereafter the drive will keep its position as long as bRegulatorOn remains set.

Handling of mechanical brakes

If supported by the drive, the variable eBrakeControl may be (re)set by the function block SMC3_BrakeControl. Three different modes are possible (SMC3_BrakeSetState: SMC_BRAKE_AUTO, SMC_BRAKE_OPEN, SMC_BRAKE_CLOSE).

The variable bBrakeClosedRealState reports the current state of the brake.

From SoftMotion point of view the control as well as the state of the mechanical brake neither depend on the state of the drive nor on the ongoing motion.

Switching between Controller modes

The function block SMC_SetControllerMode enables the user to switch between different controller modes. The variables byControllerMode resp. byRealControllerMode of AXIS_REF represent the desired resp. the actual state of the controller. Possible values are:

1	Torque/power control
2	Velocity control
3,0	Position control
4	Current control

After writing the variable byControllerMode the actual state will be checked as long as it equals the desired control mode.

Homing (drive controlled)

SoftMotion offers two possibilities to home a drive: one is controlled by the drive, the other by the controller.

The latter one is executed by calling the function block SMC_Homing. Thereby, the reference signal has to be part of the IO image of the controller. During the homing operation the drive remains in a position control mode.
In general, this method is not as precise as the drive controlled homing, as the position latch (when touching the reference switch) is based on the actual positions received from the drive; hence its accuracy is limited by the communication cycle.

By contrast at a drive controlled homing the reference switch is wired to the drive itself, which is able to evaluate the position more quickly and therefore with a higher precision. During the drive controlled homing process, the drive is set to a different control mode, where it doesn't follow the controllers set values, but is controlled by internal procedures. As soon as the homing method is finished, the drive will be reset to its original control mode.

The function block MC_Home starts the drive controlled homing procedure by setting fReference and bStartReference. Additionally the state of the AXIS_REF (wState) is set to "homing". As soon as the drive passes in the homing procedure (what means that it does no longer obey the set values of the controller), its driver will set the parameter bStartReferenceRealState to TRUE. To indicate the finishing of the homing procedure the driver will reset the values of bStartReference and bStartReferenceRealState to FALSE .

Latching (drive controlled)

SoftMotion provides two possibilities to latch the actual position of a drive: the more precise one is to wire the signal instructing the drive to latch its position directly to the drive; the second possibility is to use a controller input for measuring the actual position. The first method is much more precise (as there are less delays for transmitting the signals), but needs the drive to support this feature. Moreover, in general window capturing is supported only by the second method. Nevertheless the capturing inside the drive is the standard method.

The function block MC_TouchProbe designed for latching will communicate with the driver via the data structure aCaptDesc (being part of the AXIS_REF parameters); this data structure contains all information on captured positions, start and stop of capturing and valid positions of latching windows.

14.5.6 Example Applications

Application Examples

Please regard that »PLC Designer« SoftMotion is a product liable for costs and licenses, which is not unlocked on all »PLC Designer«-compatible controllers. Referring to this please contact the vendor of your controller.

If you are using »PLC Designer« SP WinV3 or »PLC Designer« SP RTE, please make sure to have started the SoftMotion variant of the runtime system. In case of »PLC Designer« SP WinV3 you will find this in the start menu -> Programs -> 3S »PLC Designer« -> »PLC Designer« SP Win V3 -> »PLC Designer« SP SoftMotion Win V3 as a test version, for »PLC Designer« SP SoftMotion RTE you need a special setup.

The examples described in the following use the »PLC Designer« SP SoftMotion Win V3 runtime system, but can be switched to another SoftMotion controller via the "Update device“ functionality. Further on the examples use virtual drives which might be replaced by real drives of the same name.

See the following programming examples:

- Single Axis Motion Control
- Single Axis Motion Control with Visualization Template
- Drive Control via CAM and Virtual Time Axis
- Changing CAMs
- CNC program with direct creation of the OutQueue
- CNC program decoding online with use of variables
- CNC program with preprocessing in online

Single Axis Motion Control

(See the corresponding sample project PLCopenSingle.pro)

This example shows how a drive can be controlled via modules matching the PLCopen standards:

In the device tree a virtual drive has been inserted below **SoftMotion General Drive Pool**:

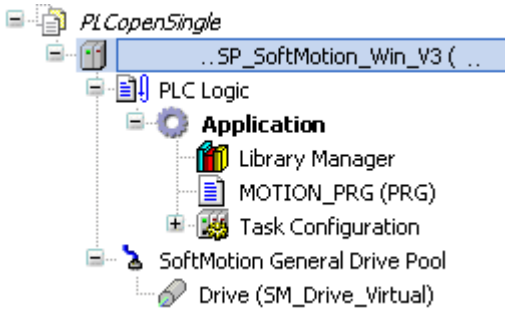


Abb. 517 Device Tree, Drive inserted

By a double-click on the new device **Drive** the following configuration dialog will open:

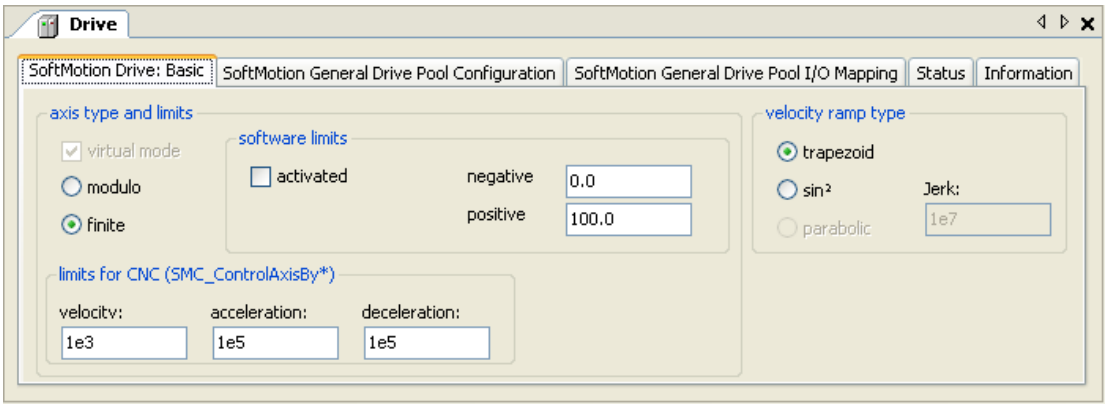


Abb. 518 SoftMotion Drive: Basic, Dialog in offline mode

In the **Task Configuration** set up the call of program **MOTION_PRG**, which will create a movement on the given axis.

See in the following how to create this program:

Add a program named MOTION_PRG, implementation language **ST**, and declare the following variables:

```
PROGRAM MOTION_PRG
VAR
iStatus: INT;
Power: MC_Power;
MoveAbsolute: MC_MoveAbsolute;
p:REAL:=100;
END_VAR
```

Program a state machine, which in the initial state activates control for the axis:

```
CASE iStatus OF
0:
Power(Enable:=TRUE, bRegulatorOn:=TRUE, bDriveStart:=TRUE, Axis:=Drive);
IF Power.Status THEN
iStatus := iStatus + 1;
END_IF
```

In the subsequent state by use of module MC_MoveAbsolute the axis will be driven to position p:

```
... (* continuation of the program shown above *)
1:
MoveAbsolute(Execute:=TRUE, Position:= p, Velocity:=100, Acceleration:=100,
Deceleration:=100, Axis:=Drive);
IF MoveAbsolute.Done THEN
MoveAbsolute(Execute:=FALSE, Axis:=Drive);
iStatus := iStatus + 1;
END_IF
```

In the last step the axis will be moved back to position 0:

```
... (* continuation of the program shown above *)
2:
MoveAbsolute(Execute:=TRUE, Position:= 0, Velocity:=100, Acceleration:=100,
Deceleration:=100, Axis:=Drive);
IF MoveAbsolute.Done THEN
MoveAbsolute(Execute:=FALSE, Axis:=Drive);
iStatus := 1;
END_IF
```

Subsequently there is a jump back to state 1. This effects a pendular movement between the positions p and 0.

This program can be built and loaded to the controller after that has been started and the correct address has been configured in the communication settings dialog.

When the **Drive** configuration dialog is opened in **online mode**, the axis movement will be displayed:

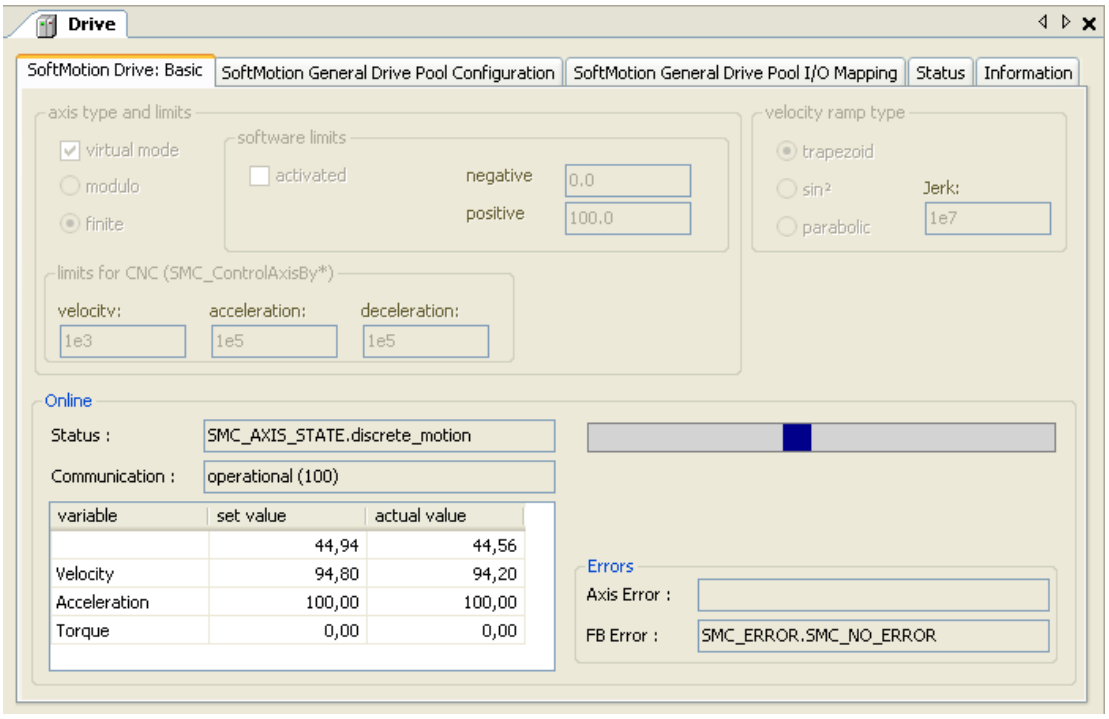


Abb. 519 SoftMotion Drive: Basic, Dialog in online mode

Single Axis Motion Control with Visualization Template

(*See the corresponding sample project PLCopenSingle2.project*)

Instead of ST also each other IEC programming language might be used, as for example CFC in the current example. This example serves to demonstrate the start and interrupt mechanism of the function modules. Further on the different start modes for module for rotatory axis can be tried out.

Configure the **Task Configuration** and insert a **drive** like done in example Single Axis Motion Control, but now use a **rotatory** drive with period 360. The program MOTION_PRG is written in **CFC** and contains only three calls of instances of the modules **MC_Power** (needed for activating the axis), **MC_MoveAbsolute** and **MC_MoveVelocity**:

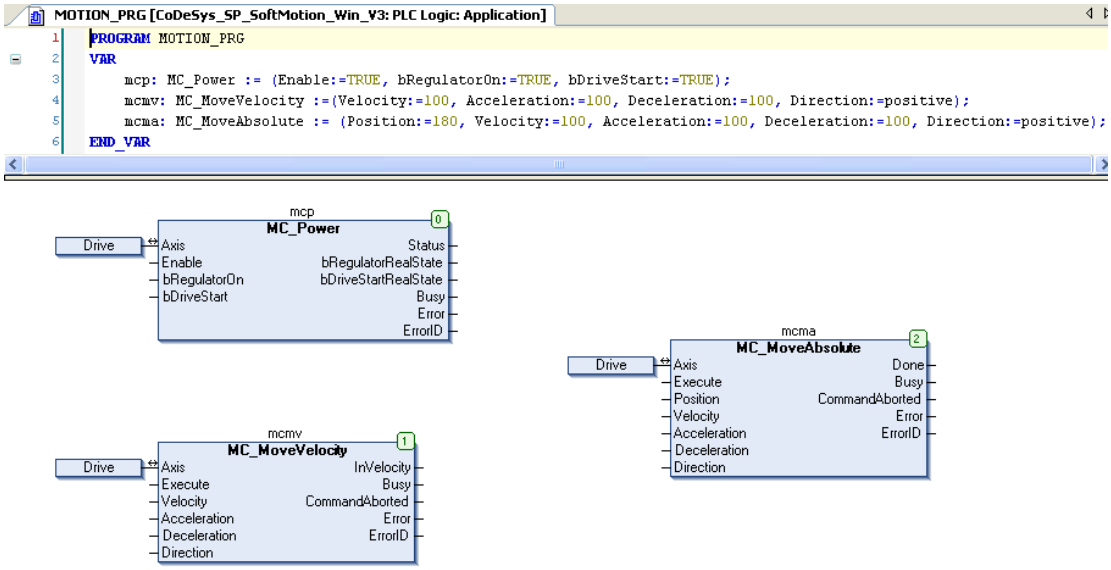


Abb. 520 Program MOTION_PRG

It is recommended to initialize the inputs of the modules, because then it is not necessary to reinsert the values each time when starting this test application.

Additionally we create an **operation visualization**. We use the **visualization templates** provided by the libraries and via the frame concept we connect them with the function block instances:

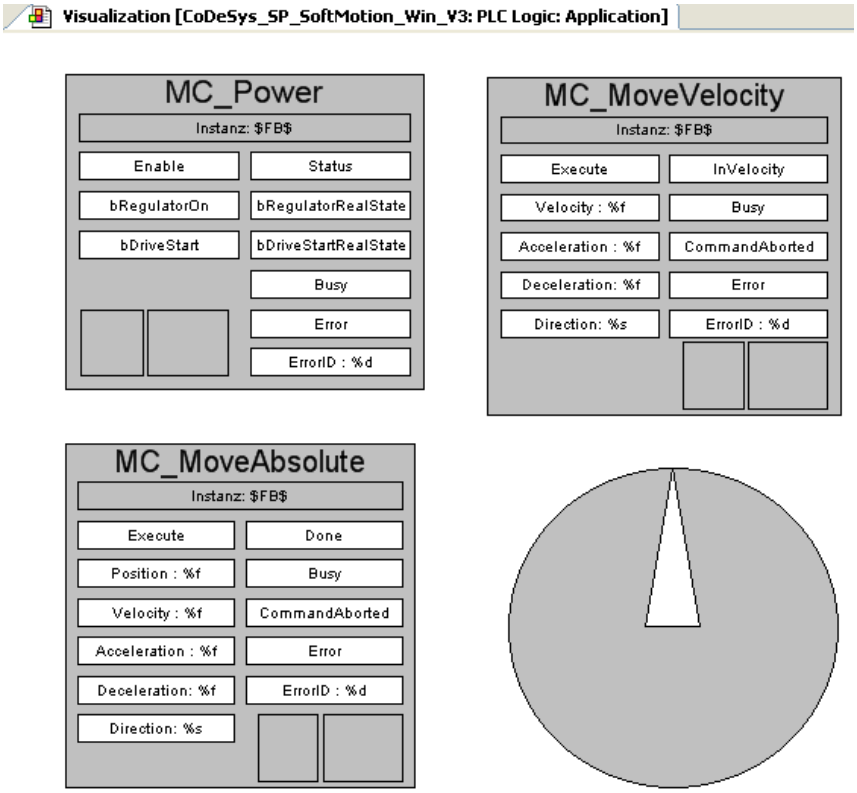


Abb. 521 Visualization, frames with visualization templates

Now we can **compile** the project without errors, then **log in** to the controller and **start** the application. By activating input 'Execute' of **MoveVelocity** the drive will start rotating. Activating 'Execute' of **MoveAbsolute** will position the drive to the set position, whereby it will be rotated in positive direction, according to the setting **Direction: positive**. This will cause an interrupt of module MoveVelocity.

Play with the modules by direct editing the modules within the visualisation: test various velocities and accelerations and also test the direction modes (positive/negative/current/shortest/fastest) of MoveAbsolute.

Drive Control via CAM and Virtual Time Axis

(*See the corresponding sample project PLCopenMulti.project*)

The following example demonstrates how a **periodic CAM** can be realized on a **linear drive**. Additionally it shows the **tappets** function.

1. In the **device tree** create any periodic **CAM** (named "Example" in the sample project) which refers to a master axis covering a range between 0 and 10 sec and which at least contains one inverting **tappet** with ID1; for example:

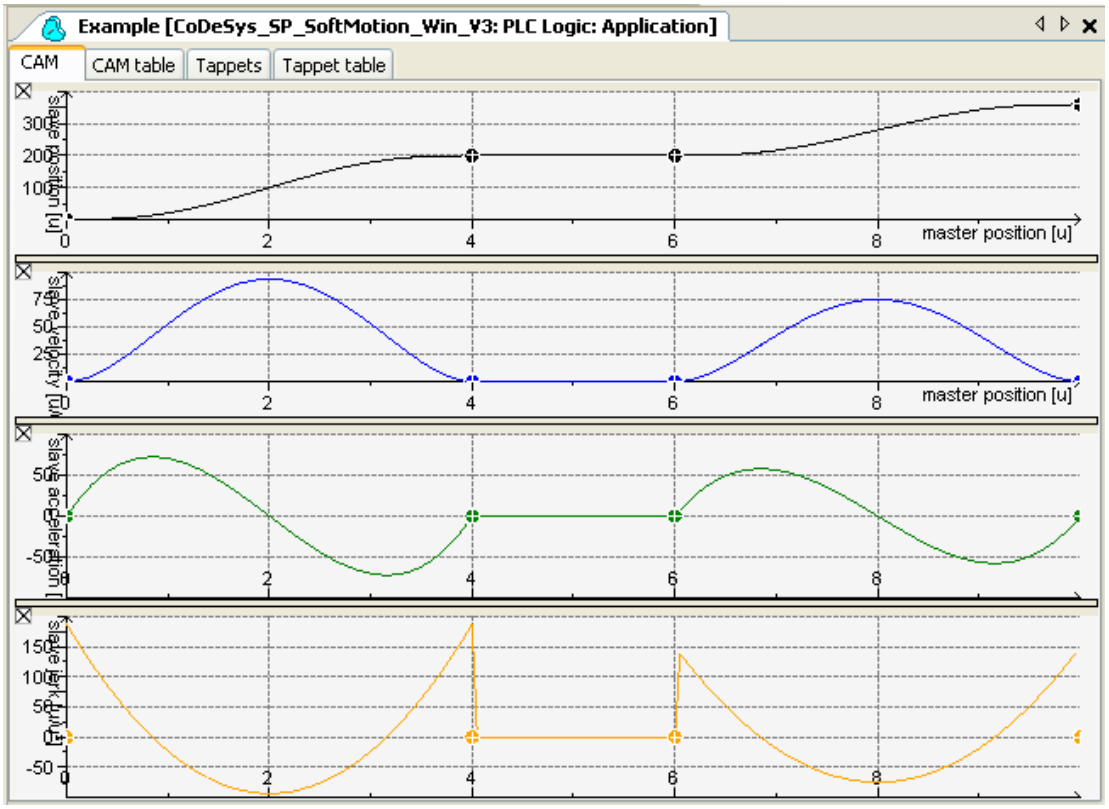


Abb. 522 CAM "Example", CAM configuration

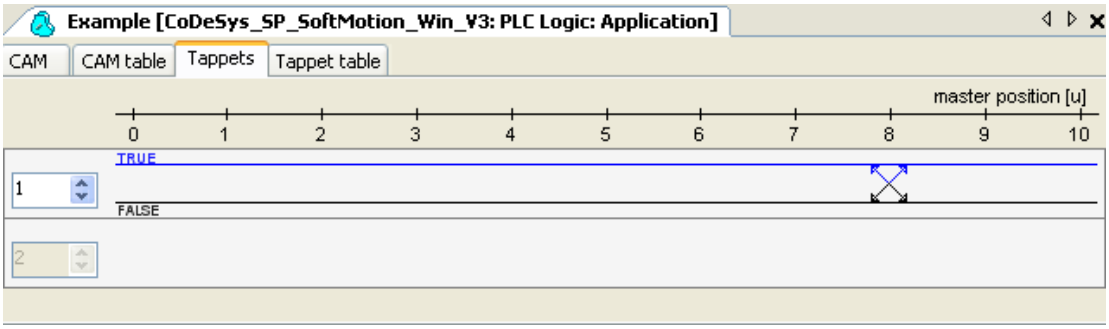


Abb. 523 CAM "Example", Tappets configuration

2. In the device tree define two **modular drives** with names "Drive" (period 360) and "Virtual“ (period 10):

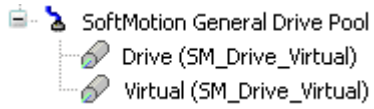


Abb. 524 Device Tree with two modular drives

3. Create program **MOTION_PRG** in **FBD**, containing the calls of the following modules:

```
PROGRAM MOTION_PRG
VAR
power1, power2: MC_Power;
TableSelect: MC_CamTableSelect;
CamIn: MC_CamIn;
Tappet: SMC_GetTappetValue;
MoveVirtual: MC_MoveVelocity;
END_VAR
```

After the **Power** modules (power1, power2) for the slave axes first the **MoveVelocity** module will be called, moving the virtual master. **TableSelect** will select the desired CAM, and **CamIn** will realize it. The **Tappet** module checks the position of the tappet. Due to the fact that the tappet is configured 'inverted', it will switch every 10 seconds.

Now you can compile the program and start it on the controller.

In order to control the target resp. current position, create a **visualization**, which will help to check the particular modules and the position of the axes.

Regard, that the master of the CAM not only can be a virtual time axis, but of course any desired **AXIS_REF** data structure. For drives which the control is currently activated, the target values will be regarded, for drives which currently are not controlled, the current values.

Changing CAMs

(*See the corresponding sample project PLCopenMulti.CAM.project*)

This example shows how a CAM movement with **two alternating CAMs** can be realized. It has been programmed in **ST** and performs the same actions like shown in example Drive Control via Virtual Time Axis. At the end of the first CAM the **MC_CamIn** module sets the output bEndOfProfile, which will cause that the each the other **MC_CamTableSelect** will be used and restarted together with MC_CamIn.

CNC Example 1: Direct Creation of the OutQueue

See the corresponding sample project "CNCdirect.project" basing on the configuration file "softmotion.cfg"!

1. Creation of the NC program in the CNC-Editor:

We create an example program, which is running between x out of [0,100] and y out of [0,100]. Additionally we define the velocities and accelerations for the path and set two witch points on the path. E.g.:

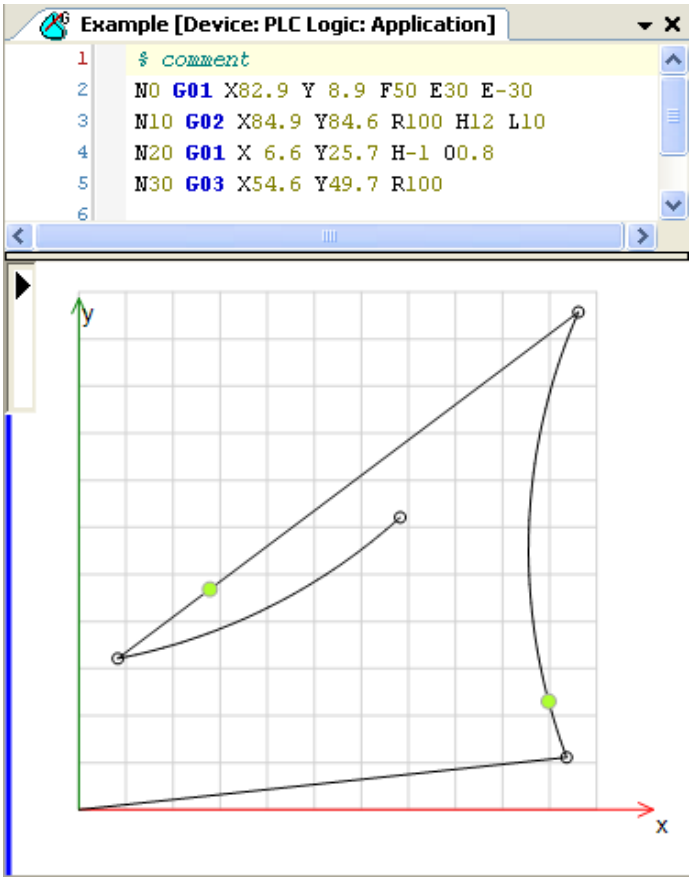


Abb. 525 CNC program of the application CNCdirect

As compile mode we choose **SMC_OutQueue**.

2. Drive Interface, PLC configuration:

Define a drive structure with 2 linear drives; the maximum velocity etc. is to be set.

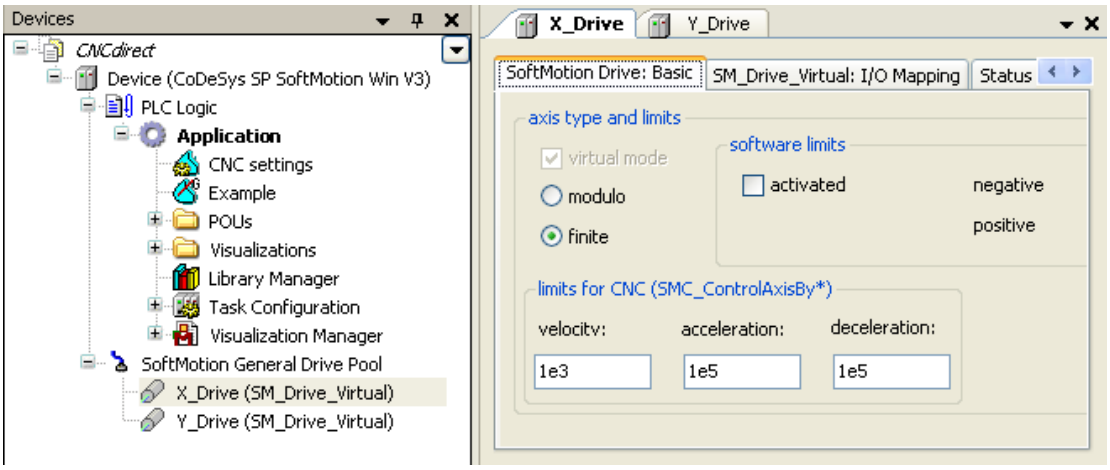


Abb. 526 Device tree of CNCdirect

3. Creation of the IEC program:

First we have to activate the drives via the MC_Power module.

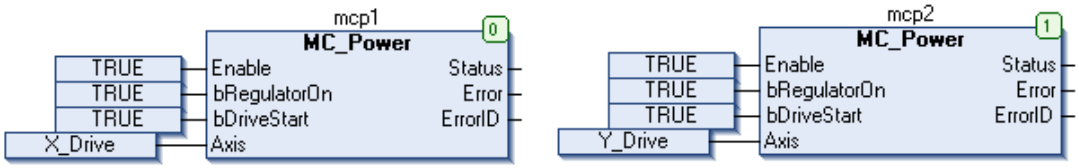


Abb. 527 MCpower function blocks from the Ipo object

A further important element is the SMC_Interpolator module. As input *poqDataIn* it gets the address of the CNC program. Besides that the IEC- task cycle time must be written to *dwIpoTime*. You can enter it either as a constant value at input dwIpoTime, or you use variable dwCycle of the axis group structure (PLC configuration), which has the advantage, that – if you modify the cycle time of the task – automatically the correct time will be used as input for the Interpolator.

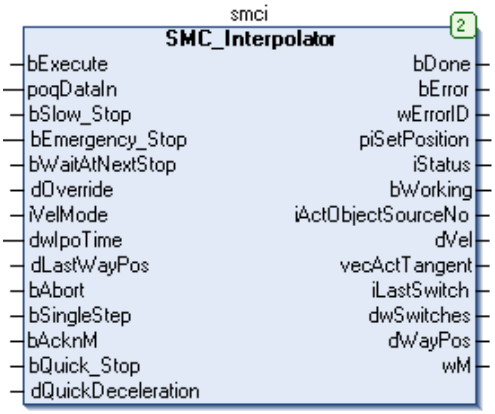


Abb. 528 Function block SMC_Interpolator from Ipo object

We want to control a portal system with this example program. For this reason we insert an instance of the backward and forward transformation modules from library SM_Trafo.lib. The forward transformation module as inputs gets the drives (the Z-drive is allocated with a otherwise not used variable *dummy* of type AXIS_REF); the backward module must get the target position of the *Interpolator*:

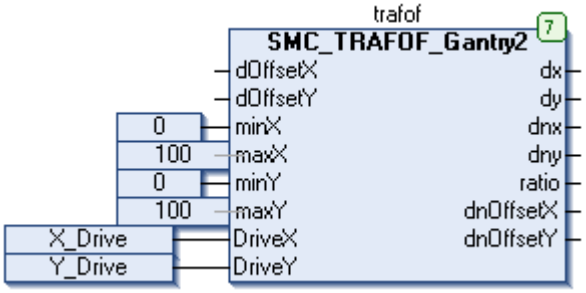


Abb. 529 Forward transformation module

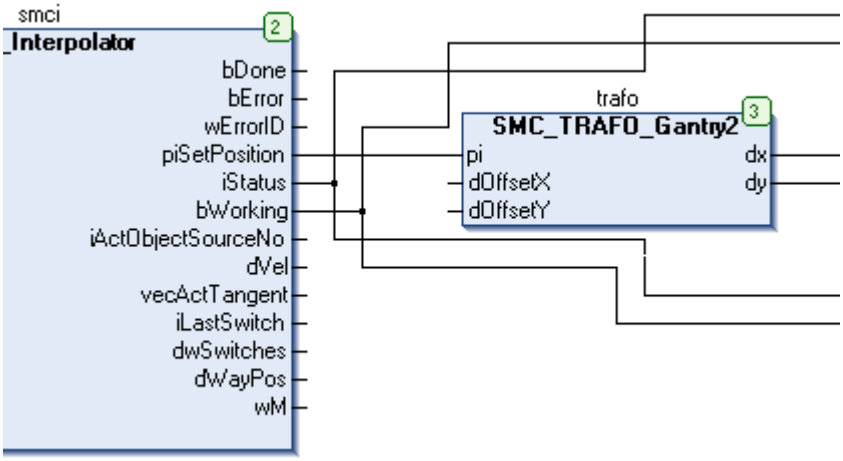


Abb. 530 Backward transformation module

The outputs of the module, that means the axes coordinates, now must be written to the drives. For that the function blocks *SMC_ControlAxisByPos* are used. Due to the fact that our application does not guarantee continuous outputs of the Interpolator (e.g. the path ends at a point different to that where it starts), we should activate gap avoiding functions (bAvoidGaps, fGapVelocity, fGapAcceleration, fGapDeceleration), we should connect the *StopIpo*-output with the *bEmergency_Stop* of the Interpolator and we should connect the Interpolator-output *iStatus* with the corresponding inputs of the axes control modules.

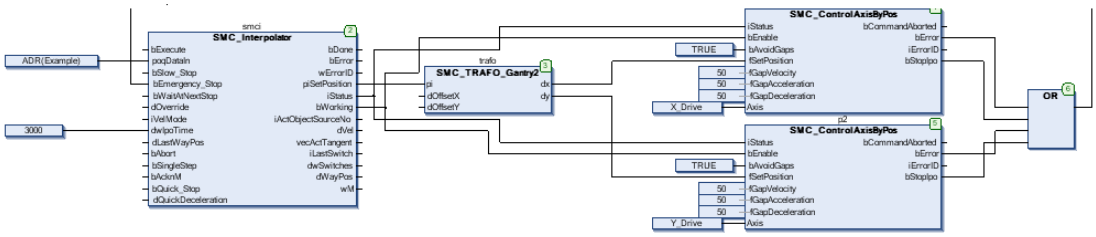


Abb. 531 Connections of the needed function blocks of the Ipo object

Please regard during programming in CFC the correct order of the elements !

4. Creating the operation and test interface:

Create a new visualization and insert two visualization elements of type 'Visualization'. The first one is the Interpolator template, the second one is the Transformation template. They get linked to the corresponding function block instances (here: Ipo.smci resp. Ipo.trafof) via the placeholder functionality.

5. Starting

The program now can get compiled without errors and can be started. It will execute the CNC program, as soon as the Execute-input of the Interpolators gets set. After having been processed completely, an other rising edge will cause a new run.

Regard the function of the path switches, which are also displayed in the visualization of the Interpolator module.

CNC Example2: Decoding online with use of variables

See the corresponding sample project CNConline.project!

1. Creating the CNC program in the CNC-Editor:

Like in the previous example we create a CNC program, but now we use two global variables g_x and g_y.

Example:

```
VAR_GLOBAL
g_x: REAL:=100;
g_y:REAL:=50;
END_VAR
```

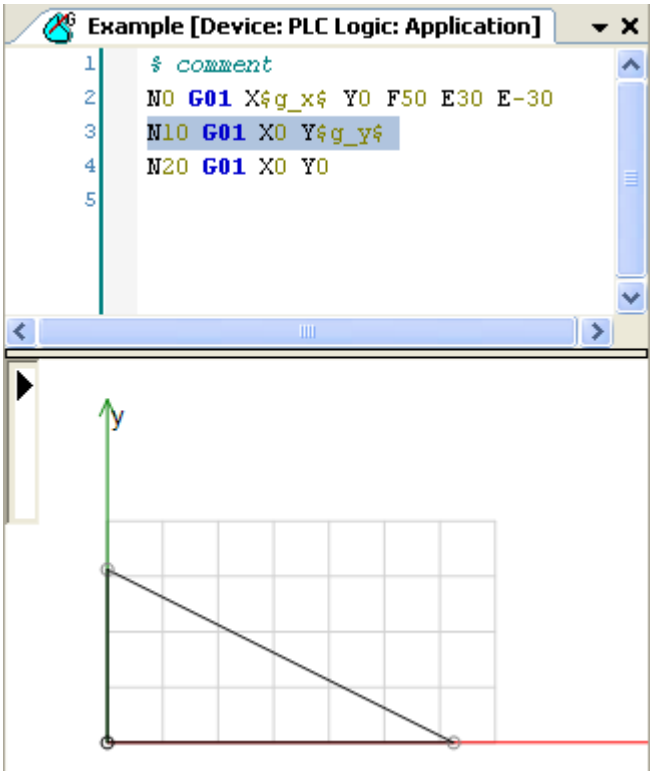


Abb. 532 Example object

We choose compilation mode SMC_CNC_REF because we are using variables in our CNC program.

2. Drive Interface, PLC configuration:

The drive structure is the same as described in the previous example.

3. Creating the IEC program:

Due to the fact that we have chosen a different compile mode for the current example, we have to do the decoding and path-preprocessing in the IEC program. This time-consuming process must not be done in time with the Interpolator (reason: at each call of the Decoder a path object is created, which typically can be used for many Interpolator calls), thus the Interpolator often is swapped out to a task of lower priority, which is called more rarely:

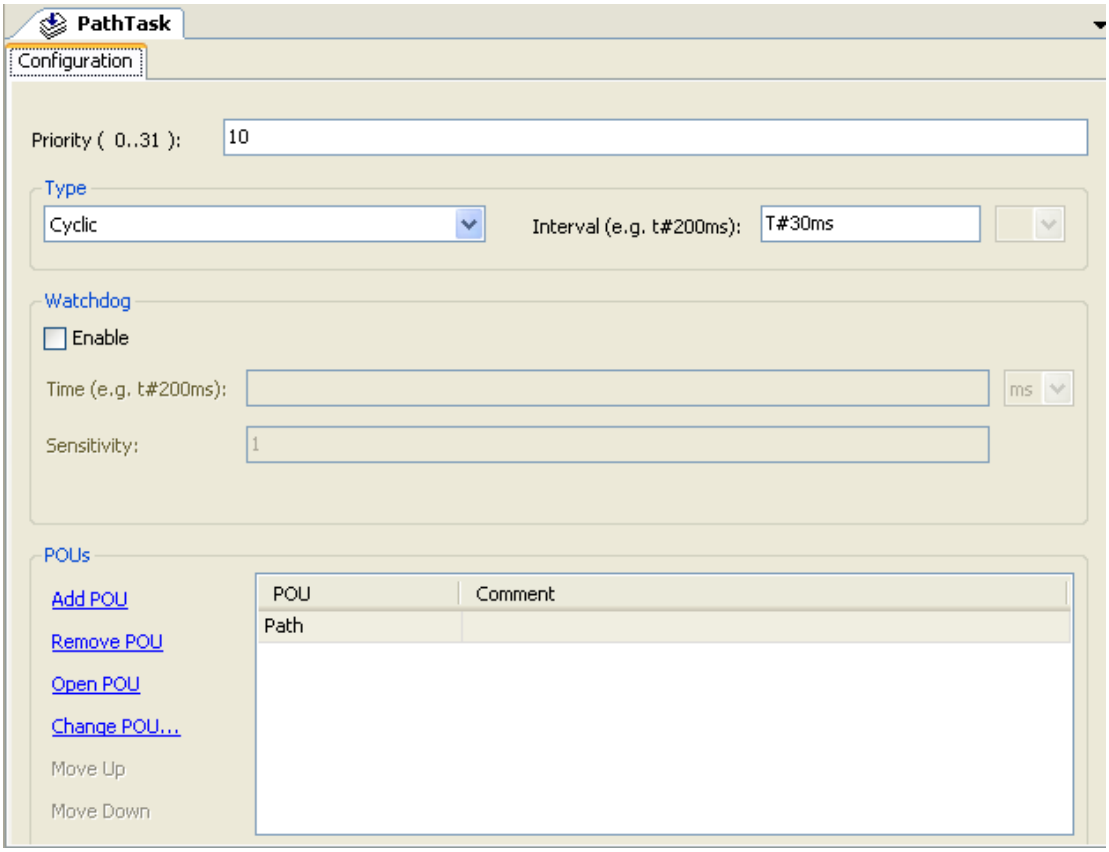


Abb. 533 Path task object

The underlying mechanism: In the slow task initially about one GEOINFO-object will be created per cycle, which is stored in the OUTQUEUE-structure of the Decoder module. As soon as this OUTQUEUE is filled, the modules of the slow task will pause until the OUTQUEUE isn't full any longer, this means until the fast task has processed the first GEOINFO-object and removed it from the OUTQUEUE. In this case the modules of the slow task get active again and re-fill the OUTQUEUE-structure. In the fast task per each cycle one path position point of that OUTQUEUE-structure, to which the *DataIn* parameter of the Interpolator is currently pointing, will be calculated and processed. Due to the fact that a GEOINFO-object generally consists of several position points, it will take several cycles until the first GEOINFO-object has been processed completely and will be removed by the Interpolator. Since the processing of a GEOINFO-object obviously takes more cycles than the creating, the slow task actually can be called more rarely than the fast. However, the task times must be defined in a way which makes sure that in the last OUTQUEUE of the slow task always enough GEOINFO-objects are available, so that no data-underrun can occur. A data-underrun would result if the Interpolator does not get any GEOINFO-objects from DataIn while the end of the path has not yet been reached. In this case the Interpolator will brake and stop unitl new data elements are available.

In program Path the decoding of the NC program and the velocity check are done:

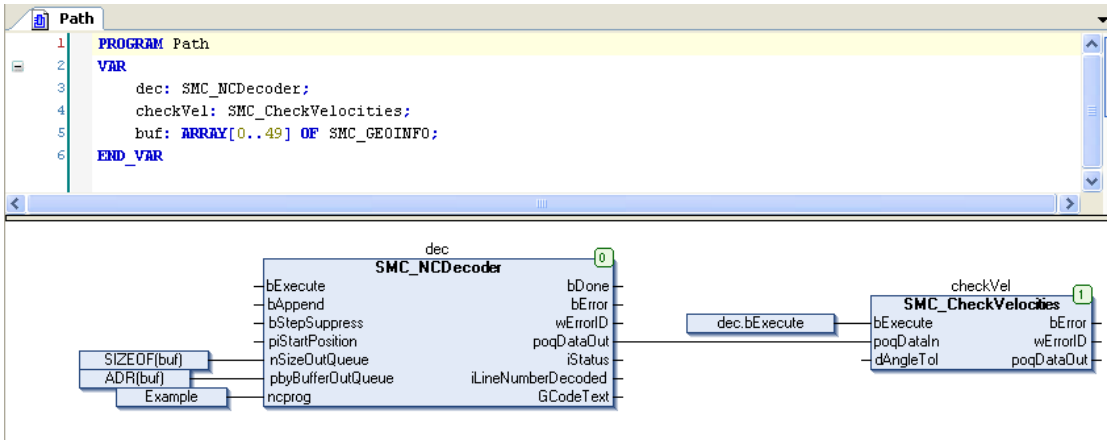


Abb. 534 Path object

The interpolating part of the IEC program nearly remains at it was, except that the data input of the Interpolator is not the CNC program name (ADR(Example)), but the OutQueue-output of the path-preprocessing modules (here: g_ncDecoder.poqDataOut).

4. Creating the operation and test interface:

For a visualization of the previous example it is useful to add templates of the new modules (SMC_NCDecoder and SMC_CheckVelocities). Besides that the global variables g_x and g_y should be editable, in order to be able to check their function later at start-up.

5. Start up:

The program now can be compiled without errors and started. It will process the CNC program as soon as the Execute-inputs of the Decoder and Interpolator have been set. If you change the values of the global variables, those will be read at a re-start of the Decoder and the path will be adjusted accordingly. Also regard the function of the Append-input of the Decoder.

CNC Example 3: Path preprocessing online

See the corresponding sample project CNCprepro.project!

We want to extend the previous example by a path-preprocessing function: The corners of the program should be smoothed by splines. This is done by the SMC_SmoothPath module. The CNC program must be embraced by the words G51/G50:

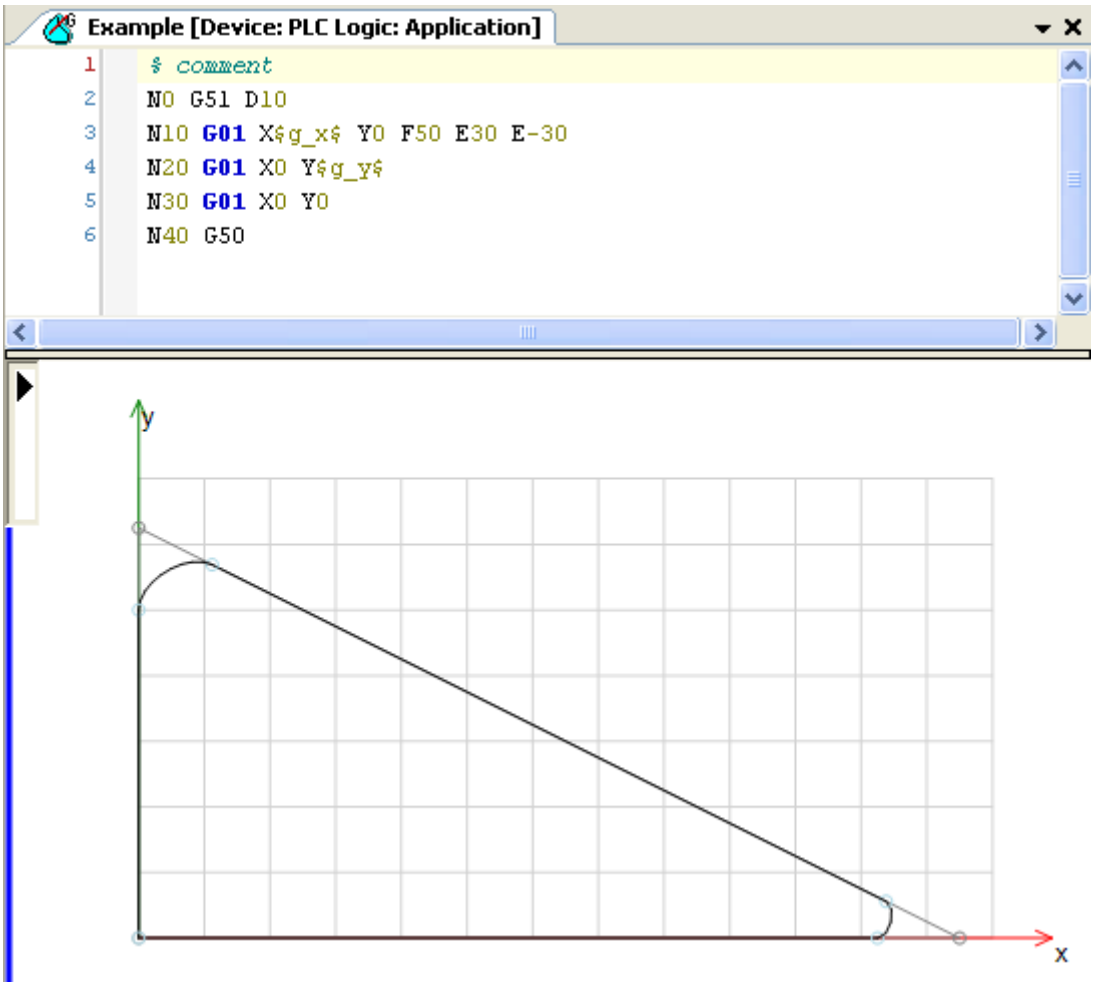


Abb. 535 Example object of the CNCprepro.project

If we did not use variables, we could compile the program as it is as a Queue and could directly pass it on to the Interpolator. But using variables, we must do the decoding and rounding off in the program. For this purpose we define a new POU of type SMC_SmoothPath and call it after the Decoder:

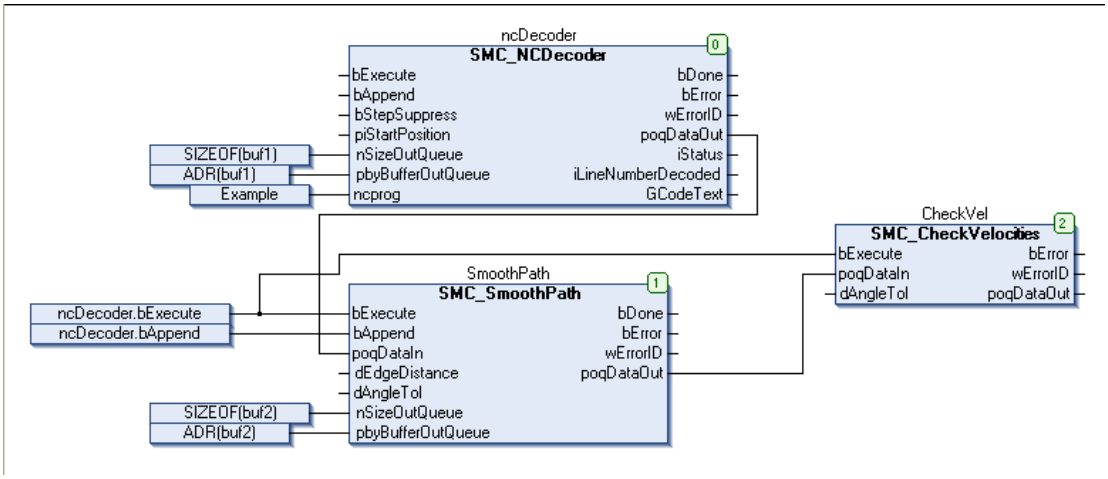


Abb. 536 Path object of the CNCprepro.project

The data input of the Interpolator module as always must be connected to the output poqDataOut of the CheckVelocities module.

This program can get compiled without errors and will not stop - contrary to the previous one - in the corners of the NC program, because the corners now have been smoothed by the path-preprocessing module.

CNC Example 4: CNC by means of table editor

See the corresponding sample project **CNC_table.projec**t.

Here the functionality of the sample project **CNCprepro.project** is programmed by means of the table editor of the CNC Editor. Unlike in **CNCprepro** in **CNC_table** no IEC variables are used, so the parameters are set with fix values. Otherwise the functionality is identical.

CNC Example 5: CNC from file

See the corresponding sample project **CNC_File.projec**t.

With this example you can comprehend the reading in of a ASCII file with G-code program in IEC code. This makes sense when the G-code program is very large.

Here the functionality of the project is like the functionality in the **CNCOnline.project**, but he G-Code program is read form an ASCII file and no IEC variables are used. The parameters are set with fix values. The processing order is like there in the **CNC_Online.project**, too.

All in all you can find following differences:

1. additional virtual drive (Z axis)
2. function block **smoothpath** is not needed
3. G-Code is read from file
4. preprocessing is online (see **CNC-Online.project**)
5. additional function block **ReadNCFile**
6. Visu template available

15 Command Line Options

15.1 Starting PLC Designer.exe

When starting PLC Designer.exe via a command line, the options listed in the contents tree of the current online help can be used.

--culture

This command line option can be added when starting codesys.exe in order to get the user interface provided in one of the available languages,.

Syntax:

--culture=<Culture>

where <Culture> is something like "de", "en", "fr", "it", "es", "zh-CHS", etc.

Example:

PLC Designer.exe --culture=en

for starting the English user interface.

16 Programming Reference

16.1 Declaration

The variables of a project are to be declared manually in the declaration editor or via the Auto Declare dialog. See the further related help pages referring for example on the various categories of variables (local, global, input, output etc.), the initialization, the use of pragmas, init method etc.

16.1.1 Variables Declaration

The declaration of a variable can be done in the declaration part of a POU or via the Auto Declare dialog, as well as in a DUT or GVL editor.

The "sort" (in the declaration dialog it is named "scope") of the variable(s) to be declared is specified by the keywords embracing the declaration of one or several variables. The common variable declaration for example is embraced by "VAR" and "END_VAR".

The variable type keywords may be supplemented by attribute keywords, like for example "RETAIN" (VAR_INPUT RETAIN).

The declaration of a variable must match the following rules:

Syntax:

<Identifier> {AT <address>}:<Type> {:=<initialization>};

The parts in braces {} are optional.

Identifier: The identifier is the name of a variable. The items listed in the following in each case must be regarded when defining an identifier.

- It must not contain spaces or special characters.
- It is not case-sensitive, which means that for example "VAR1", "Var1" and "var1" are all the same variable.
- The underscore character is recognized in identifiers (for example, "A_BCD" and "AB_CD" are considered two different identifiers), but an identifier must not have more than one underscore character in a row.
- The length of the identifier as well as the meaningful part of it, are unlimited.
- The rules listed in the following text box concerning multiple use must be regarded:

Multiple use of identifiers (namespaces):

- An identifier must not be used duplicate locally.
- An identifier must not be identical to any keyword.
- Globally an identifier can be used plurally, thus a local variable can have the same name as a global one. Within a POU in this case the local variable will have priority.
- A variable defined in a Global Variables List can have the same name as a variable defined in another Global Variables List. In this context notice the following IEC 61131-3 extending features, which have not been available in »PLC Designer« V2.3:

1. Global scope Operator: An instance path starting with "." opens a global scope. So, if there is a local variable, for example "ivar", with the same name as a global variable, ".ivar" refers to the global variable.
2. The name of a Global Variables List can be used as a namespace for the included variables. So variables can be declared with the same name in different Global Variable Lists and can be accessed specifically by preceding the variable name with the list name.
   Example:
   globlist1.ivar := globlist2.ivar; (* ivar from globlist2 is copied to ivar in globlist1 *)
3. Variables defined in a Global Variables List of an included library can be accessed according to syntax <library namespace>.<name of Global Variables List>.<variable>". See below for namespaces of libraries.
   Example:
   globlist1.ivar := lib1.globlist1.ivar (* ivar from globlist1 in library lib1 is copied to ivar in globlist1 *)

- For a library also a namespace is defined, when it gets included via the Library

- Manager. So you can access a library module or variable by <library namespace>.<modulename|variablename>. Notice that in case of nested libraries the namespaces of all libraries concerned have to been stated successively. Example: If Lib1 was referenced by Lib0, the module fun being part of Lib1 is accessed by "Lib0.Lib1.fun":

  ivar := Lib0.Lib1.fun(4, 5); (* return value of fun is copied to variable ivar in project *)
  As far as the checkbox "Publish all IEC symbols to that project as if this reference would have been included there directly." has been activated within the properties of the referenced library Lib, the module fun may also be accessed directly via "Lib0.fun".

AT <address>: The variable can directly be linked to a definite address using the keyword AT.

In function blocks you can also specify variables with incomplete address statements. In order that such a variable can be used in a local instance, there must be an entry for it in the 'Variable Configuration'.

Type: valid Data Type, optionally extended by a ":=<initialization>".

Optionally pragma instructions can be added in the declaration part of an object, in order to affect the code generation for various purposes.

Note:
Pay attention to the possibility of an automatic declaration.
For faster input of the declarations, use the shortcut mode.

16.1.2 Recommendations on the naming of identifiers

Identifiers are defined at the declaration of variables (Variable names), user-defined data types and at the creation of POUs (functions, function blocks, programs). In addition to the items to be regarded anyway when defining an identifier you might consider to follow some recommendations in order to make the naming as unique as possible.

See for the naming of identifiers for the following:

- Variable names
- Variable names in »PLC Designer« V3.x Libraries
- User-defined data types (DUT)
- User defined data types (DUTs) in »PLC Designer« V3.x Libraries
- Functions, Function blocks, Programs (POU), Actions
- POUs in »PLC Designer« V3.x Libraries
- Visualization names

(1) Variable names

The naming of variables in applications and libraries as far as possible should follow the **Hungarian notation**:

For each variable a meaningful, short description should be found, the **base name**.

The first **letter** of each **word** of a base name should be a capital letter, the others should be small ones (Example: FileSize).

Before the base name, corresponding to the data type of the variable, **prefix(es)** are added in small letters.

See in the table some information and the recommended prefixes on the particular data types:

Data type	lower limit	upper limit	Information content	Prefix	Comment
BOOL	FALSE	TRUE	1 Bit	x*	
				b	reserved
BYTE			8 Bit	by	Bit string, not for arithm. operations
WORD			16 Bit	w	Bit string, not for arithm. operations
DWORD			32 Bit	dw	Bit string, not for arithm. operations
LWORD			64 Bit	lw	not for arithm. operations
SINT	-128	127	8 Bit	si	
USINT	0	255	8 Bit	usi	
INT	-32.768	32.767	16 Bit	i	
UINT	0	65.535	16 Bit	ui	
DINT	-2.147.483.648	2.147.483.647	32 Bit	di	
UDINT	0	4.294.967.295	32 Bit	udi	
LINT	-2^{63}	$2^{63} - 1$	64 Bit	li	
ULINT	0	$2^{64} - 1$	64 Bit	uli	
REAL			32 Bit	r	
LREAL			64 Bit	lr	
STRING				s	

Data type	lower limit	upper limit	Information content	Prefix	Comment
TIME				tim	
TIME_OF_DAY				tod	
DATE_AND_TIME				dt	
DATE				date	
ENUM			16 Bit	e	
POINTER				p	
ARRAY				a	

* pointedly for BOOLean variables x is chosen as prefix, in order to differentiate from BYTE and also in order to accommodate the perception of an IEC-programmer (see addressing "%IX0.0").

Examples:

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

In **nested** declarations the prefixes are attached to each other in the order of the declarations:

Example:

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

Function block instances and **variables of user-defined data types** as a prefix get a shortcut for the FB- resp. data type name (for example: sdo).

Example:

```
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 constants (c) start with prefix c and an attached underscore, followed by the type prefix and the variable name.

Example:

```
VAR CONSTANT
  c_uiSyncID: UINT := 16#80;
END_VAR
```

For **Global Variables** (g) and **Global Constants** (gc) an additional prefix + underscore is attached to the library prefix:

Examples:

```
VAR_GLOBAL
  CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
  CAN_gc_dwExample: DWORD;
END_VAR
```

(2) Variable names in »PLC Designer« V3.x Libraries

Basically see above (1) Variable names, with the following exception: global variables and constants do not need a library prefix, because using the namespace takes the function of a prefix.

Example:

g_iTest: INT; (declaration)

CAN.g_iTest (implementation, call in an application program)

(3) User-defined data types (DUT)

Structure: The name of each structure data type consists of a library prefix (Example: CAN), an underscore and a preferably short expressive description (for example: SDOTelegram) of the structure. The associated prefix for used variables of this structure should follow directly after the colon.

Example:

```
TYPE CAN_SDOTelegram :    (* prefix: sdo *)
STRUCT
  wIndex:WORD;
  bySubIndex:BYTE;
  byLen:BYTE;
  abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Enumerations start with the library prefix (Example: CAL), followed by an underscore and the identifier in capital letters..

Notice that in previous versions of»PLC Designer« ENUM values > 16#7FFF have caused inconsistencies, because they did not get converted automatically to INT values. For this reason ENUMs should be defined with correct INT values.

Example:

```
TYPE CAL_Day :(
CAL_MONDAY,
CAL_TUESDAY,
CAL_WEDNESDAY,
CAL_THIRSDAY,
CAL_FRIDAY,
CAL_SATURDAY,
CAL_SUNDAY);
```

Declaration:

```
eToday: CAL_Day;
```

(4) User defined data types (DUTs) in »PLC Designer« V3.x Libraries

DUT names in »PLC Designer« V3.x libraries do not get a library prefix, because using the namespace takes the function of a prefix. Also enumeration components are defined without library prefices.:

Example (in library with namespace CAL):

```
TYPE Day :(
MONDAY,
TUESDAY,
WEDNESDAY,
THIRSDAY,
FRIDAY,
SATURDAY,
SUNDAY);
```

Declaration:

```
eToday: CAL.Day;
Use in application:
IF eToday = CAL.Day.MONDAY THEN
```

(4) Functions, Function blocks, Programs (POU), Actions

The names of functions, function blocks and programs consist of the **library prefix** (Example: CAN), an **underscore** and an expressive short **name** (e.g: SendTelegram) **of the POU**. Like with variables always the first letter of a word of the POU name should be a capital letter, the others should be small letters. It is recommended to compose the name of the POU of a **verb** and a **substantive**.

Example:

```
FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *)
```

In the declaration part a short description of the POU should be provided as a **comment**. Further on all inputs and outputs should be provided with comments. In case of function blocks the associated prefix for set-up instances should follow directly after the name.

Actions get no prefix; just actions which should be called only internally, that is by the POU itself, start with **prv_**.

Each function - for the reason of compatibility with previous»PLC Designer« versions - must have at least one parameter. **External functions** must not use structures as return values.

(6) POUs in »PLC Designer« V3.x Libraries

POU names in »PLC Designer« V3.x libraries do not get a library prefix, because using the namespace takes the function of a prefix. For creating method names the same rules apply as for actions. Possible inputs of a method should get english comments. Also a short description of a method should be added in its declaration. Interface names should start with letter "I", for example: ICANDevice

(7) Visualization names

Note: Currently just avoid to name a visualization like another object in the project, because this would cause problems in case of visualization changes.

16.1.3 Variables Initialization

The **default-initialization** value is 0 for all declarations, but user defined initialization values can be added in the declaration of each variable and data type.

The **user defined initialization** is brought about by the assignment operator "**:=**" and can be any valid ST expression. Thus constant values as well as other variables or functions can be used to define the init value. The programmer just has to make sure that a variable used for the initialization of another variable is already initialized itself.

Examples for valid variable initializations:

VAR

var1:INT := 12;	(* Integer variable with initial value of 12*)
x : INT := 13 + 8;	(* init value defined by an expression with constants *)
y : INT := x + fun(4);	(* init value defined by an expression containing a function call; be aware of the order in this case! *)
z : POINTER TO INT := ADR(y);	(* not described by the IEC61131-3: init value defined by a address function; Be careful in this case: the pointer will not be initialized during online change! *)

END_VAR

see also: Arbitrary expressions for variable initialization

Note: As from compiler version 3.3.2.0 variables of global variables lists are always initialized before local variables of a POU.

16.1.4 Arbitrary expressions for variable initialization

A variable can be initialized with any valid ST expression. It is possible to access other variables out of the same scope, and it is possible to call functions. However the programmer has to assure that a variable used for initialization of another variable is already initialized.

Examples for valid variable initializations:

```
VAR
x : INT := 13 + 8;
y : INT := x + fun(4);
z : POINTER TO INT := ADR(y); (* be careful: the pointer will not be initialized during online change! *)
END_VAR
```

16.1.5 Declaration Editor

The declaration editor is a text or tabular editor used for the declaration of variables. Usually it is provided in combination with the language editors.

See Declaration Editor for detailed information.

16.1.6 Autodeclaration Dialog

It can be defined in the Options dialog, category 'Text Editor' / 'Editing', that an **Auto Declare** dialog should open as soon as a not yet declared string is entered in the implementation part of an editor and the <ENTER> key is pressed. This dialog will support the declaration of the variable.

The dialog also can be opened explicitly by command 'Auto Declare', which by default is available in the Edit menu, or by <Shift>+<F2>. If an already declared variable is selected before, its declaration can be edited in the dialog.

16.1.7 Shortcut Mode

The declaration editor as well as other text editors where declarations are done, support the "shortcut mode".

This mode is activated when you end a line of declaration with **<Ctrl>+<Enter>** and allows to use shortcuts instead of completely typing the declaration.

The following shortcuts are supported:

- All identifiers up to the last identifier of a line will become declaration variable identifiers
- The type of declaration is determined by the last identifier of the line. In this context, the following will apply:

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 type has been established through these rules, automatically BOOL is the type and the last identifier will not be used as a type (see below: example 1).
- Every constant, depending on the type of declaration, will turn into an initialization or a string (see below: examples 2 and 3).
- An address (as in %MD12) is extended by the "AT" attribute (see below: example 4).
- A text after a semicolon (;) becomes a comment (see below: example 4.).
- All other characters in the line are ignored (see below, e.g., the exclamation point in example 5.).

Examples:

	Shortcut	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;

16.1.8 AT Declaration

In order to link a project variable directly with a definite address you can enter this address in the declaration of the variable. Regard that the assignment of a variable to an address also can be done in the mapping dialog of a device in the PLC configuration (device tree).

Syntax:

<identifier> AT <address> : <data type>;

Keyword **AT** must be followed by an valid address. See the Address help page for further information and consider possible overlaps in case of byte addressing mode.

This declaration allows to assign a meaningful name to an address. Any changes concerning an incoming or outgoing signal may only be done in a single place (for example in the declaration).

Notice the following when choosing a variable to be assigned to an address:

- Variables requiring an input cannot be accessed by writing. The compiler intercepts this with error.
- AT declarations only can be used with local or global variables, not however with input and output variables of POUs.
- AT declarations must not be used in combination with VAR RETAIN or VAR PERSISTENT !
- If AT declarations are used with structure or function block members, all instances will access the same memory location, which corresponds to "static variables" in classic programming languages like e.g. C.
- The memory layout of structures is determined by the target too.

Examples:

```
counter_heat7 AT %QX0.0: BOOL;
lightcabinetimpulse AT %IX7.2: BOOL;
download AT %MX2.2: BOOL;
```

Note: If boolean variables are assigned to a Byte, Word or DWORD address, they occupy one byte with TRUE or FALSE, not just the first bit after the offset!

16.1.9 Keywords

Keywords are to be written in uppercase letters in all editors.

The following strings are reserved as keywords, i.e. they cannot be used as identifiers for variables or POUs:

- ABS
- ACOS
- ACTION (only used in the Export Format)
- ADD
- ADR
- AND
- ANDN
- ARRAY
- ASIN
- AT
- ATAN
- BITADR
- BOOL
- BY
- BYTE
- CAL
- CALC
- CALCN
- CASE
- CONSTANT
- COS
- DATE
- DINT
- DIV
- DO
- DT
- DWoRD
- ELSE
- ELSIF
- END_ACTION (only used in the Export Format)
- END_CASE
- END_FOR
- END_FUNCTION (only used in the Export Format)
- END_FUNCTION_BLOCK (only used in the Export Format)

- END_IF
- END_PROGRAM (only used in the Export Format)
- END_REPEAT
- END_STRUCT
- END_TYPE
- END_VAR
- END_WHILE
- EQ
- EXIT
- EXP
- EXPT
- FALSE
- FOR
- FUNCTION
- FUNCTION_BLOCK
- GE
- GT
- IF
- INDEXOF
- INT
- JMP
- JMPC
- JMPCN
- LD
- LDN
- LE
- LINT
- LN
- LOG
- LREAL
- LT
- LTIME
- LWORD
- MAX
- MIN
- MOD
- MOVE

- MUL
- MUX
- NE
- NOT
- OF
- OR
- ORN
- PARAMS
- PERSISTENT
- POINTER
- PROGRAM
- R
- REFERENCE
- READ_ONLY
- READ_WRITE
- REAL
- REPEAT
- RET
- RETAIN
- RETC
- RETCN
- RETURN
- ROL
- ROR
- S
- SEL
- SHL
- SHR
- SIN
- SINT
- SIZEOF
- SUPER
- SQRT
- ST
- STN
- STRING
- STRUCT

- SUPER
- SUB
- TAN
- THEN
- TIME
- TO
- TOD
- TRUE
- TRUNC
- TYPE
- UDINT
- UINT
- ULINT
- UNTIL
- USINT
- VAR
- VAR_ACCESS (only used very specifically, depending on the hardware)
- VAR_CONFIG
- VAR_EXTERNAL
- VAR_GLOBAL
- VAR_IN_OUT
- VAR_INPUT
- VAR_OUTPUT
- VAR_STAT
- VAR_TEMP
- WHILE
- WORD
- WSTRING
- XOR
- XORN

Additionally all conversion operators as listed in the input assistant are handled as keywords.

16.1.10 Local Variables - VAR

Between the keywords **VAR** and **END_VAR** all local variables of a POU are declared. These have no external connection; in other words, they can not be written from the outside.

Regard the possibility of adding an attribute to "VAR".

Example:

```
VAR
 iLoc1:INT; (* 1. Local Variable*)
END_VAR
```

16.1.11 Input Variables - VAR_INPUT

Between the key words **VAR_INPUT** and **END_VAR** all variables are declared that serve as input variables for a POU. That means that at the call position, the value of the variables can be given along with a call.

Regard the possibility of adding an attribute to "VAR_INPUT".

Example:

```
VAR_INPUT
 iIn1:INT (* 1. Inputvariable*)
END_VAR
```

16.1.12 Output Variables - VAR_OUTPUT

Between the key words **VAR_OUTPUT** and **END_VAR** all variables are declared that serve as output variables of a POU. That means that these values are carried back to the POU making the call. There they can be answered and used further.

Regard the possibility of adding an attribute to "VAR_OUTPUT".

Example:

```
VAR_OUTPUT
 iOut1:INT; (* 1. Output variable*)
END_VAR
```

Output variables in functions and methods:

According to IEC 61131-3 draft 2, functions (and methods) can have additional outputs. Those must be assigned in the call of the function like this:

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

The return value of the function *fun* will be written to the e.g. locally declared variables *loc1* and *loc2*.

16.1.13 Input and Output Variables - VAR_IN_OUT

Between the key words **VAR_IN_OUT** and **END_VAR** all variables are declared that serve as input and output variables for a POU.

Note: With variables of this type the value of the transferred variable is changed ("transferred as a pointer", Call-by-Reference). That means that the input value for such variables cannot be a constant. For this reason, even the VAR_IN_OUT variables of a function block can not be read or written directly from outside via <functionblockinstance><in/outputvariable>.

Example:

```
VAR_IN_OUT
 iInOut1:INT; (* 1. Inputoutputvariable *)
END_VAR
```

16.1.14 Global Variables - VAR_GLOBAL

„Normal" variables, constants, external or remanent variables that are known throughout the project can be declared as global variables.

Note: A variable defined locally in a POU with the same name as a global variable will have priority within the POU.

Note: As from compiler version 3.2.0.0 global variables always will be initialized before local variables of POUs.

The variables have to be declared locally between the keywords VAR_GLOBAL and END_VAR. Regard the possibility of adding an attribute to "VAR_GLOBAL".

A variable is recognized as a global variable by a preceding dot: for example ".iGlobVar1".

For detailed information on multiple use of variable names, the global scope operator "." and name spaces see: Global Scope Operator.

Use Global Variables lists (GVLs) to structure and handle global variables within a project. A "GVL" can be added via the Add Object command.

16.1.15 Temporary Variables - VAR_TEMP

This feature is an **extension to the IEC 61131-3 standard**.

VAR_TEMP declarations are only possible within programs and function blocks. Those variables are also only accessible within the body of the program or function block. VAR_TEMP declared variables are (re)initialized every time the POU is called.

The variables have to be declared locally between the keywords VAR_TEMP and END_VAR.

16.1.16 Static Variables - VAR_STAT

This feature is an **extension to the IEC 61131-3 standard**.

Static variables can be used in function blocks, methods and functions. They have to be declared locally between the keywords VAR_STAT and END_VAR and will be initialized at the first call of the respective POU.

Static variables are only accessible within the scope in which they are declared (like a static variable in C), but like a global variable do not loose its value after the POU is left. For example in a function they might be used as a counter for the number of function calls.

Regard the possibility of adding an attribute to "VAR_STAT".

16.1.17 External variables - VAR_EXTERNAL

These are global variables which are "imported" into the POU.

They have to be declared locally between the keywords VAR_EXTERNAL and END_VAR and in the global variable list. The declaration must exactly match the global declaration. If the global variable does not exist, an error message will appear.

Note: In CoDeSys it is not necessary to define variables as external. These keyword is provided in order to maintain compatibility to IEC 61131-3.

Example:

```
VAR_EXTERNAL
 iVarExt1:INT; (* 1st external variable *)
END_VAR
```

16.1.18 Attribute Keywords for Variable Types

The following attribute keywords can be added to the declaration of the variables "type" in order to specify the scope:

RETAIN: See remanent variables of type RETAIN

PERSISTENT: See remanent variables of type PERSISTENT

CONSTANT: See constants

16.1.19 Remanent variables - RETAIN, PERSISTENT

Remanent variables can retain their value throughout the usual program run period.

They are declared as "Retain Variables" or even more stringent as "Persistent Variables". For each case a separate memory area is used.

The declaration determines the degree of "resistance" of a remanent variable in the case of resets, downloads or a reboot of the PLC. In applications mainly the combination of both remanent flags will be required (see below Persistent Variables).

Note: If a V2.3-project is opened, the declarations of Retain variables will maintain effective, the declarations of persistent variables however must be reworked, see below: an own persistent global variables list must be created!

Note: Do not use the "AT Declaration" in combination with "VAR RETAIN" or "VAR PERSISTENT !

See in the following:

Retain variables

Persistent variables

Synoptic table on the behaviour

Retain variables

Variables declared as "Retains" will be kept PLC-dependently but typically in a separate memory area. They get the keyword "**RETAIN**" in their declaration in a POU resp. a in a global variable list.

Example:

```
VAR RETAIN
 iRem1 : INT; (* 1. Retain variable*)
END_VAR
```

Retain variables are identified by the keyword **RETAIN**. These variables maintain their value even after an uncontrolled shutdown of the controller as well as after a normal switch off and on of the controller (resp. at the online command Reset warm <application>. When the program is run again, the stored values will be processed further. A concrete example would be an piece-counter in a production line, that recommences counting after a power failure.
All other variables are newly initialized, either with their initialized values or with the standard initializations.
Contrary to Persistent variables Retain Variables are reinitialized at a new download of the program.

Application example: A piece counter in a production machine, which should continue counting after a power fail.

Retain variables however are re-initialized at 'Reset origin <application>' and - in contrast to persistent variables - at 'Reset cold <application>' and an application download.

The Retain property can be combined with the Persistent property. See below, the synoptic table.

Note:
If a local variable in a program is declared as VAR RETAIN, then exactly that variable will be saved in the retain area (like a global retain variable)

If a local variable in a function block is declared as VAR RETAIN, then the complete instance of the function block will be saved in the retain area (all data of the POU), whereby only the declared retain variable will be handled as a retain.

If a local variable in a function is declared as VAR RETAIN, then this will be without any effect. The variable will not be saved in the retain area ! If a local variable is declared as PERSISTENT in a function, then this will be without any effect also !

Persistent variables

Persistent variables are identified by keyword "PERSISTENT" (**VAR_GLOBAL PERSISTENT**). They get only re-initialized at a Reset origin <application>. In contrast to Retain variables they maintain their values after a download. An application example for persistent variables would be a counter for operating hours, which should continue counting even after a power fail or download. See below the synoptic table.

Persistent variables are handled like described in the following - and thus **different from »PLC Designer« V2.3**:

Persistent variables ONLY can be declared in a **special global variables list** of object type 'Persistent Variables', which is assigned to an application. There might be only ONE such list per application.

Note: As from V3.3.0.1 a declaration with "VAR_GLOBAL PERSISTENT" effects the same as a declaration with " VAR_GLOBAL PERSISTENT RETAIN" or "VAR_GLOBAL RETAIN PERSISTENT".

Like Retain variables the Persistent variables get stored in a separate memory area.

Example:

```
VAR GLOBAL PERSISTENT RETAIN
 iVarPers1 : DINT; (* 1. Persistent+Retain Variable App1 *)
 bVarPers : BOOL;  (* 2. Persistent+Retain Variable App1 *)
END_VAR
```

Note: Currently only global persistent variables are possible.

The target system must provide a **separate memory area** for the persistent variables list of each application.

At each **re-load of the application** the persistent variables list on the PLC will be checked against that of the project. The list on the PLC inter alia is identified by the application name. In case of **inconsistencies** the user will be prompted to **re-initialize** all persistent variables of the application. Inconsistency might result from renaming or removing or other modifications of the existing declarations in the list.

Note: So carefully consider any modifications in the declaration part of the persistent variables list and the effect of the resultantly requested re-initialization !

New declarations only can be added at the end of the list. At a download those will be detected as "new" and will not demand a re-initialization of the complete list.

Synoptic table on the behaviour of remanent variables

x = Value gets maintained - = Value gets initialized after Online command	VAR	VAR RETAIN	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset warm <application>	-	x	x
Reset cold <application>	-	-	x
Reset origin <application>	-	-	-
Download <application>	-	-	x
Online Change <application>	x	x	x
Reboot PLC	-	x	x

16.1.20 Constants - CONSTANT, Typed Literals

Constants are identified by the key word **CONSTANT**. They can be declared locally or globally.

Syntax:

```
VAR CONSTANT
 <Identifier>:<Type> := <initialization>;
END_VAR
```

Example:

```
VAR CONSTANT
 c_iCon1:INT:=12; (* 1. Constant*)
END_VAR
```

See the Operands description for a listing of possible constants. See there also regarding the possibility of using typed constants :

Typed Literals

Basically, in using IEC constants, the smallest possible data type will be used. If another data type must be used, this can be achieved with the help of typed literals without the necessity of explicitly declaring the constants. For this, the constant will be provided with a prefix which determines the type.

This is written as follows:

Syntax:

<Type>#<Literal>

<Type> specifies the desired data type; possible entries are all simple data types. The type must be written in uppercase letters.

<Literal> specifies the constant. The data entered must fit within the data type specified under <Type>.

Example:

```
iVar1:=DINT#34;
```

If the constant cannot be converted to the target type without data loss, an error message is issued:

Typed literals can be used wherever normal constants can be used.

Constants in online mode

As long as the default setting Replace Constants (Compile Options) is active, **constants** are indicated by an symbol preceding the value in the Value column of the Declaration or Watch window in online mode. In this case they cannot be accessed by e.g. forcing or writing.

16.1.21 Variables configuration - VAR_CONFIG

The "Variables configuration" can be used to map **function block variables** on the process image, that is on the device I/Os, without the need of specifying the definite address already in the declaration of the function block variable. The assignment of the definite address in this case is done centrally for all function block instances in a **global VAR_CONFIG list**.

For this purpose you can assign **"incomplete" addresses** to the function block variables declared between the key words VAR and END_VAR. These addresses are to be identified with an asterisk.

Syntax:

<identifier> AT %<I|Q>* : <data type>;

Example of the use of not completely defined addresses:

```
FUNCTION_BLOCK locio
VAR
 xLocIn AT %I*: BOOL := TRUE;
 xLocOut AT %Q*: BOOL;
END_VAR
```

Here two local I/O-variables are defined, a local-In (%I*) and a local-Out (%Q*).

Then the **final definition** of the addresses is to be done in the "Variables Configuration" in a global variables list:

For this purpose add a 'Global Variable list' object (GVL) to the Devices window by the Add Object command. There enter the declarations of the instance variables with the definite addresses between the key words **VAR_CONFIG** and **END_VAR**.

The instance variables must be specified by the complete instance path through which the individual POUs and instance names are separated from one another by periods. The declaration must contain an address whose class (input/output) corresponds to that of the incompletely specified address (%I*, %Q*) in the function block. Also the data type must agree with the declaration in the function block.

Syntax:

<instance variable path> AT %<I|Q><location> : <data type>;

Configuration variables, whose instance path is invalid because the instance does not exist, are denoted as errors. On the other hand, an error is also reported if no definite address configuration exists for an instance variable assigned to an incomplete address.

Example for a Variable Configuration

Assume that the following definition for function block locio - see the example above - is given in a program:

```
PROGRAM PLC_PRG
VAR
locioVar1: locio;
locioVar2: locio;
END_VAR
```

Then a corrected variable configuration would look this way:

```
VAR_CONFIG
PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;
PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

Note: Changes on directly mapped I/Os are immediately shown in the process image, whereas changes on variables mapped via VAR_CONFIG are not shown before the end of the responsible task.

16.1.22 Declaration and Initialization of User Defined Data Types

Besides the standard data types also user defined types might be used.

For information on declaration and initialization see the pages on DUT (Data Unit Type).

16.1.23 Extensible Functions - PARAMS

(not yet implemented)

As an extension to the IEC 61131-3 standard it is possible to declare functions or methods with a variable number of input parameters of the same type.

Syntax:

<extensible_declaration> : = <function name> : PARAMS OF <data type>

Example:

```
FUNCTION funHelp : INT
VAR_INPUT
pint : PARAMS OF <base data type>;
END_VAR
```

Implicitly pint points to a structure containing the components Count of type INT and Params of type POINTER TO <base data type>.

When calling funHelp it is either possible to pass an array of arbitrary length of type <base data type>, or a variable number of <base data type>-expressions, or mix both.

Example:

```
x := funHelp(1, 2, 3);
y := funHelp(intArray);
z := funHelp(1, intArray, 2);
```

A function may only contain one PARAMS-input, and this input must be the last input of the function. There is no syntax for explicitly assigning an PARAMS-input within the function call.

16.1.24 FB_init, FB_reinit Methods

FB_init

The method "FB_init" replaces the INI operator used in »PLC Designer« V2.3. It is a special method for a function block which can be declared explicitly but also and in any case is available implicitly. Thus in any case it can be accessed for each function block.

The init method contains **initialization code** for the function block as declared in the declaration part of the function block. If the init method is declared explicitly, the implicit initialization code will be inserted into this method. The programmer can add further initialization code.

Note: When execution reaches the user defined initialization code, the function block is already fully initialized via the implicit initialization code.

The init method is called after download all for each declared instance. Attention: In online change the previous values will overwrite the initialization values!

On the call sequence in case of inheritance please see: FB_exit, Execution order

Regard also the possibility of defining a function block instance method to be called automatically after initialization via FB_init: attribute call_after_init

Interface of the init method:

```
METHOD fb_init : BOOL
VAR_INPUT
  bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
  bInCopyCode : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online change)
END_VAR
```

The return value is not interpreted.

Note: Notice also the possible use of an "exit" method and the resulting execution order: See FB_exit Method.

User defined input:

In an init method additional inputs can be declared. Those must be assigned at the declaration of a function block instance.

Example for a init method for a function block "serialdevice"

```
METHOD fb_init : BOOL
VAR_INPUT
  bInitRetains : BOOL; (* Initialization of retains *)
  bInCopyCode : BOOL;  (* Instance moved into copy code *)
  nCOMnum : INT;      (* additional input: number of the COM interface to listen at *)
END_VAR
```

Declaration of function block "serialdevice":

```
COM1 : serialdevice(nCOMnum:=1);
COM0 : serialdevice(nCOMnum:=0);
```

FB_reinit

If a method named "FB_reinit" is declared for a function block instance, this method will be called when the **instance gets copied** (which for example is the case at an online change after a modification of the function block declaration). The method will cause a reinitialization of the new instance module that has been created by the copy code. An reinitialization might be desired because the data of the original instance will be written to the new instance after the copying, but you might want to get the original init values. The FB_reinit method has no inputs. Regard that in contrast to FB_init the method **must be declared explicitly**. If a reinitialization of the basic function block implementation is desired, FB_reinit must be explicitly called for that POU.

The FB_reinit method has no inputs.

On the call sequence in case of inheritance please see: FB_exit, Execution order

16.1.25 FB_Exit Method

A special method for a function block is a method named "FB_Exit". It must be declared explicitly, there is no implicit declaration. The exit method - if present - is called for all declared instances of the function block before a new **download**, at a **reset** or during **online change** for all moved or deleted instances.

Interface of the exit method: There is only one mandatory parameter:

```
METHOD fb_exit : BOOL
VAR_INPUT
 bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied afterwards (online change).
END_VAR
```

Regard also the "FB_init" method and the following execution order:

1. exit method: exit old instance: old_inst.fb_exit(bInCopyCode := TRUE);
2. init method: init new instance: new_inst.fb_init(bInitRetains := FALSE, bInCopyCode := TRUE);
3. copy function block values (copy code): copy_fub(&old_inst, &new_inst);

Besides this, in case of inheritance the following call sequence is true (whereby for the POUs used for example in this listing the following is assumed: "SubFB EXTENDS MainFB" and "SubSubFB EXTENDS SubFB"):

```
fbSubSubFb.FB_Exit(...);
fbSubFb.FB_Exit(...);
fbMainFb.FB_Exit(...);
fbMainFb.FB_Init(...);
fbSubFb.FB_Init(...);
fbSubSubFb.FB_Init(...);

for FB_reinit:
fbMainFb.FB_reinit(...);
fbSubFb.FB_reinit(...);
fbSubSubFb.FB_Init(...);
```

16.1.26 Pragma Instructions

A pragma instruction is used to affect the properties of one or several variables concerning the compilation resp. precompilation process. This means that a pragma influences the code generation. For example it might determine whether a variable will be initialized, monitored, added to a parameter list, made invisible in the library manager or should be added to the symbol configuration. Message outputs during the build process can be forced and also conditional pragmas can be used, which define how the variable should be treated depending on certain conditions. Those pragmas also can be entered as "defines" in the compile properties of a particular object.

A pragma can be used in a separate line, or in with supplementary text in an implementation or declaration editor line. Within the FBD/LD/IL editor use the command 'Insert label' and replace the default text "Label:" in the arising text field by the pragma. In case you want to set a label as well as a pragma, insert the pragma first and the label afterwards.

The pragma instruction is enclosed in curly brackets, upper- and lower-case are ignored:

{ <instruction text> }

The opening bracket can immediately come after a variable name. Opening and closing bracket must be in the same line.

Depending on the type and contents of a pragma the pragma either operates on the line in which it is located or on all subsequent lines until it is ended by an appropriate pragma, or until the same pragma is executed with different parameters, or until the end of the file is reached. A "file" in this context is a declaration part, implementation part, global variable list or type declaration.

If the compiler cannot meaningfully interpret the instruction text, the entire pragma is handled as a comment and read over. A warning will be issued in this case.

See the following pragma types:

- Message Pragmas
- Attribute Pragmas
- Conditional Pragmas

not yet implemented: Pragmas as used in 2.3

Message Pragmas

Message pragmas can be used to force the output of messages in the Messages window during the compilation (build) of the project.

The pragma instruction can be inserted in an existing line or in a separate line in the text editor of a POU. Message pragmas positioned within currently not "defined" sections of the implementation code will not be regarded when the project is compiled. On this see the example provided with the description of the "defined (*identifier*)" Conditional Pragma.

There are four types of message pragmas:

Pragma	Message type
{text *"textstring"*}	**Text**: The specified textstring will be displayed.
{info '*textstring*'}	**Information**: The specified textstring will be displayed.
{warning digit '*textstring*'}	**Warning**: The specified textstring will be displayed. Not yet implemented: The digit indicates the desired warning level between 1 and 4. Unlike a "data type-global" Obsolete Pragma this warning is explicitly defined for the local position.
{error '*textstring*'}	**Error**: The specified textstring will be displayed.

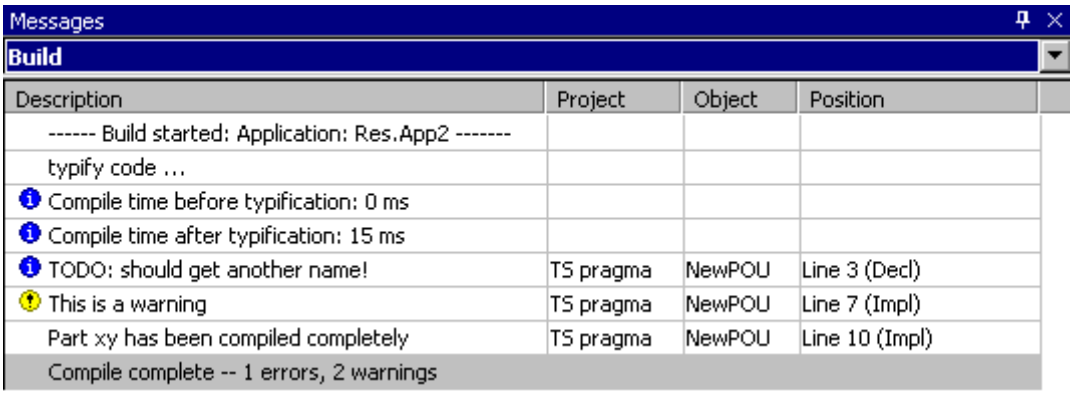
Note: In case of messages of types Information, Warning and Error the source position of the message - that is where the pragma is placed in a POU - can be reached via the 'Next Message' command. This is not possible for the Text type.

Example

Declaration and Implementation in ST-Editor:

```
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 Messages window:



Attribute Pragmas

Attribute pragmas might be assigned to a signature in order to influence the compilation resp. pre-compilation, i.e. the code generation.

There are userdefined pragmas which might be used in combination with conditional pragmas and there are also the following predefined standard attribute pragmas:

- Attribute 'displaymode'
- Attribute 'ExpandFully'
- Attribute 'global_init_slot'
- Attribute 'hide'
- Attribute 'hide_all_locals'
- Attribute 'initialize_on_call'
- Attribute 'Init_Namespace'
- Attribute 'init_on_onlchange'
- Attribute 'instance-path'
- Attribute 'linkalways'
- Attribute 'Monitoring'
- Attribute 'no_check'
- Attribute 'no_copy'
- Attribute 'No-Exit'
- Attribute 'noinit'
- Attribute 'no_virtual_actions'
- Attribute 'obsolete'
- Attribute 'pack_mode'
- Attribute 'parameterstringof'
- Attribute 'qualified_only'
- Attribute 'reflection'
- Attribute 'symbol'

Userdefined Attributes

It is possible to assign arbitrary user- or application-defined attribute pragmas to POUs, type declarations or variables. This attribute can be queried before compilation by means of conditional pragmas.

Syntax:

{attribute 'attribute'}

This pragma instruction is valid for the current or - if placed in a separate line - for the subsequent line.

An userdefined attribute can be assigned to the following objects:

- POUs, Actions Example

Attribute "vision" for function fun1:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

- Variables Example

Attribute "DoCount" for variable ivar is added :

```
PROGRAM PLC_PRG
VAR
ivar:INT; {attribute 'DoCount'};
bvar:BOOL;
END_VAR
```

- Types Example

Attribute "aType" for data type DUT_1 is added :

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
a:INT;
b:BOOL;
END_STRUCT
END_TYPE
```

Attribute call_after_init

With the help of the pragma {attribute call_after_init} a method can be defined that should be called implicitly after the initialization of a function block instance. For this purpose the attribute must be attached both to the function block itself and to the instance method to be called (for performance reasons). The method must be called after FB_Init and after having applied the variable values of an initialization expression in the instance declaration.
This functionality is supported as from compiler version >= 3.4.1.0.

Syntax:

{attribute 'attribute call_after_init'}

Example:

With the following definition:

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <functionblock definition>
{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

... a declaration like this:

```
inst : FB := (in1 := 99);
```

... will result in the following order of code processing:

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

So in FB_Afterinit you can react on the user defined initialization.

Attribute displaymode

With the help of the pragma {attribute 'displaymode'} the display mode of a single variable can be defined. This setting will overwrite the global setting for the display mode of all monitoring variables, done via the commands of the submenu Display Mode (per default available in the Online menu).

The pragma must be positioned in the line above the line containing the variables declaration.

Syntax:

{attribute 'displaymode’:=<displaymode>}

The following definitions are possible:

- for display in binary format:

```
{attribute 'displaymode':='bin'}
{attribute 'displaymode':='binary'}
```

- for display in decimal format:

```
{attribute 'displaymode':='dec'}
{attribute 'displaymode':='decimal'}
```

- for display in heximal format:

```
{attribute 'displaymode':='hex'}
{attribute 'displaymode':='hexadecimal'}
```

Example:

```
VAR
  {attribute 'displaymode':='hex'}
  dwVar1: DWORD;
END_VAR
```

Attribute ExpandFully

With the help of the pragma {attribute 'ExpandFully'} all members of arrays used as input variables for referenced visualizations can be made accessible within the Visualization Properties dialog.

Syntax:

{attribute 'ExpandFully'}

Example:

Visualization *visu* is intended to be inserted in a frame within visualization *visu_main*.

As input variable *arr* is defined in the Interface Editor of *visu* and so will later be available for assignments in the Properties dialog of the frame in *visu_main*.

In order to get the particular components of the array displayed in this Properties dialog also, the attribute 'ExpandFully' must be added directly before *arr* in the Interface editor of *visu*.

Declaration within the Interface editor of *visu*:

```
VAR_INPUT
  {attribute 'ExpandFully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```

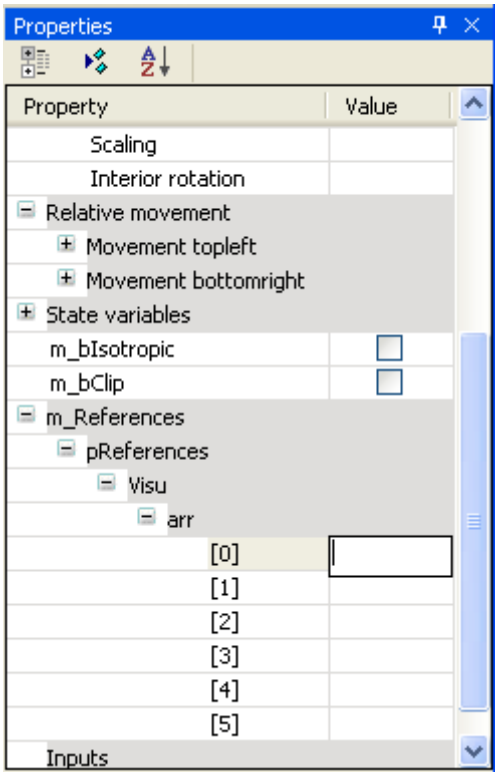


Abb. 537 Resulting properties dialog of frame in visu_main

Attribute Global_init_slot

The pragma **{attribute 'global_init_slot'}** can only be applied for signatures. Per default the order of initializing the variables of an global variable list is arbitrary. However, in some cases prescribing an order is necessary, e.g. if variables of one global list are referring to variables of another list. In this case you may make use of the pragma to arrange the order for global initialisation.

Syntax:

{attribute 'global_init_slot' := '<Value>'}

The template <Value> has to be replaced by an integer value describing the initialisation order of the signature. The default value is 50000. A lower value provokes an earlier initialisation. In case of signatures carrying the same value for the attribute 'global_init_slot' the sequence of their initialization rests undefined.

Example:

Assume the project including two global variable lists GVL_1 and GVL_2:

The global variable "A" is member of the exemplary global variable list GVL_1:

```
{attribute 'global_init_slot' := '300'}
VAR_GLOBAL
A : INT:=1000;
END_VAR
```

The initial values of the variables "B" and "C" of GVL_2 depend on variable "A".

```
{attribute 'global_init_slot' := '350'}
VAR_GLOBAL
B : INT:=A+1;
C : INT:=A-1;
END_VAR
```

Setting the attribute 'global_init_slot' of GVL_1 to 300, i.e. to the lowest value in the exemplary initialisation order, makes sure that the expression "A+1" is well defined when initialising "B".

Attribute Hide

With the help of the pragma {attribute 'hide'} you may prevent variables or even whole signatures from being displayed within the "List Components" functionality, the input assistant or the declaration part in online-modus. Only the variable subsequent to the pragma will be hidden.

Syntax:

{attribute 'hide’}

To hide all local variables of a signature use THIS attribute.

Example:

The function block myPOU is implemented making use of the attribute:

```
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
{attribute 'hide'}
a_invisible: BOOL;
a_visible: BOOL;
END_VAR
VAR_OUTPUT
b:INT;
END_VAR
VAR
END_VAR
```

In the main program two instances of function block myPOU are defined:

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

When assigning e.g. an input value to POU1, the "List Components" function working on typing "POU1." in the implementation part of PLC_PRG will display the variables "a", "a_visible" and "b", but not the hidden variable "a_invisible".

Attribute Hide_all_locals

With the help of the pragma {attribute 'hide_all_locals'} you may prevent all local variables of a signature from being displayed within the "List Components" functionality or input assistant. This attribute is identical to assigning the attribute 'hide' to each of the local variables.

Syntax:

{attribute 'hide_all_locals’}

Example:

The function block myPOU is implemented making use of 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
```

In the main program two instances of function block myPOU are defined:

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

When assigning e.g. an input value to POU1, the "List Components" function working on typing "POU1." in the implementation part of PLC_PRG will display the variables "a" and "b", but none of the hidden local variables "c" or "d".

Attribute Initialize_on_call

The pragma **{attribute 'initialize_on_call'}** may be added to input variables. An input of a function block with this attribute will be initialized at any call of the function block. If an input expects a pointer and if this pointer has been removed due to an online change, then the input will be set to NULL.

Syntax:

{attribute 'initialize_on_call’}

Attribute Init_Namespace

A variable of type STRING or WSTRING, which is provided with the pragma **{attribute 'init_namespace'}**, will be initialized with the current namespace, this is the path of the instance of the function block this variable is contained. Applying this pragma presumes the use of the additional attribute 'noinit' for the string variable and the attribute 'reflection' for the corresponding function block.

Syntax:

{attribute 'init_namespace’}

Example:

The function block POU is provided with all necessary attributes:

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR_OUTPUT
{attribute 'no_init'}
{attribute 'instance-path'}
myStr: STRING;
END_VAR
```

Within the main program PLC_PRG an instance "fb" of the function block POU is defined:

```
PROGRAM PLC_PRG
VAR
fb:POU;
newString: STRING;
END_VAR
newString:=fb.myStr;
```

The variable myStr will be initialized with the current namespace, e.g. "PLCWinNT.Application.PLC_PRG.fb". This value will be assigned to newString within the main program.

Attribute Init_On_Onlchange

The pragma **{attribute 'init_on_onlchange'}** attached to a variable will cause this variable to get initialized with each online change.

Syntax:

{attribute 'init_on_onlchange' }

Attribute Instance-path

The pragma **{attribute 'instance-path'}** may be added to a local string variable that, in consequence, will be initialized with the device tree path of the POU this string variable belongs to. This may be a useful feature for error messages. Applying this pragma presumes the use of the attribute 'reflection' for the corresponding POU and the additional attribute 'noinit' for the string variable.

Syntax:

{attribute 'instance-path'}

Example:

Assume the following function block being equipped with the attribute 'reflection':

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
{attribute 'instance-path'}
{attribute 'noinit'}
str: STRING;
END_VAR
```

An instance of the function block is called in the program PLC_PRG:

```
PROGRAM PLC_PRG
VAR
myPOU:POU;
myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

The instance myPOU having been initialized the string variable "str" will be assigned the path of the instance myPOU of POU, e.g. "PLCWinNT.Application.PLC_PRG.myPOU". This path will be assigned to variable "myString" within the main program.

Hint: The length of a string variable may be arbitrarily defined (even >255), however be aware that the string will be cut (from its back end) if it gets assigned to a variable of a too short data type.

Attribute linkalways

The pragma {attribute 'linkalways'} allows to mark POUs for the compiler in a way so that they are always included into the compile information. As a result, objects with this option will be always compiled and downloaded to the PLC. This option only affects POUs and GVLs that are located below an application or in libraries which are inserted below an application. The compiler option **Link always** does the same.

Syntax:

{attribute 'linkalways'}

When using the symbol configuration editor the marked POUs are used as basis for the selectable variables for the symbol configuration.

Example:

The global variable list GVLMoreSymbols is implemented making use of the attribute:

```
{attribute 'linkalways'}
VAR_GLOBAS
g_iVar1: INT;
g_iVar2: INT;
END_VAR
```

With this code the symbols of GVLMoreSymbols get selectable in the symbol configuration.

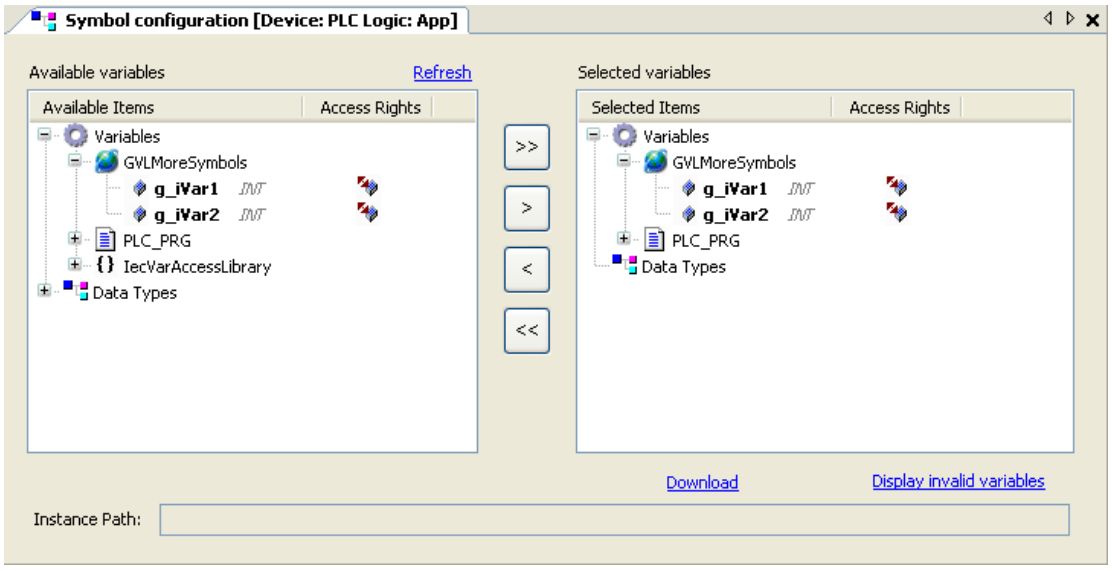


Abb. 538 Symbol configuration editor

Attribute Monitoring

A property may be monitored in online mode either with help of Inline Monitoring (not yet implemented) or with help of a Watch List.

The monitoring can be activated by adding the "monitoring" attribute pragma in the line above the property definition. Then in online view of the POU using the property resp. in a watch list the name, type and value of the variables of the property will be displayed. Therein you may also enter prepared values to force variables belonging to the property.

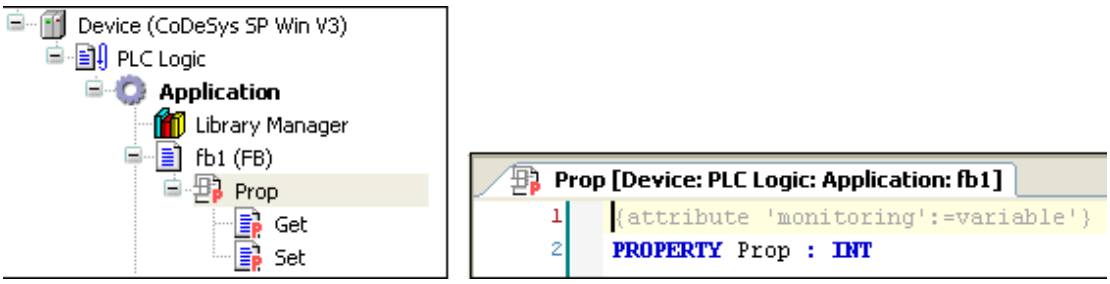


Abb. 539 Example of property prepared for variable monitoring

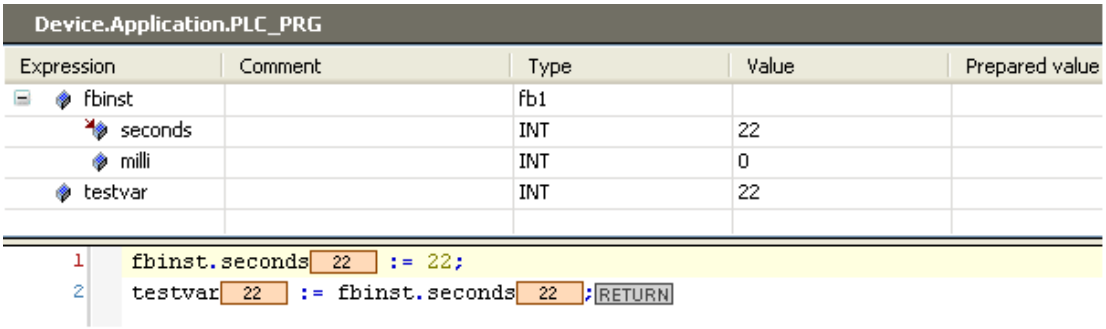


Abb. 540 Example of monitoring view

There are two different ways to get monitored the current value of the property variables. For the particular use case consider carefully which attribute is suitable to actually get the desired value. This will depend on whether operations on the variables are implemented within the property:

1. Pragma **'{attribute 'monitoring':='variable'}'**

An implicit variable is created for the property, which will get the current property value always when the application calls the Set or Get method. The value of this implicit variable will be monitored.

Syntax:

{attribute 'monitoring':='variable'}

2. Pragma **'{attribute 'monitoring':='call'}'**

This attribute can only be used for properties returning simple data types or pointers, not for structured types:

The value to be monitored is retrieved directly by calling the property, i.e. the monitoring service of the runtime system calls the Get method. Regard that if any operations on the variables are implemented within the property, the value might still change!

Syntax:

{attribute 'monitoring':='call'}

Attribute No_check

The pragma **{attribute 'no_check'}** is added to a POU in purpose to suppress the call of any check functions within this POU. As check functions may influence the performance, it is reasonable to apply this attribute to POUs that are frequently called or already approved.

Syntax:

{attribute 'no_check'}

Attribute No_Copy

Generally an online change will require a reallocation of instances e.g. of POUs. Thereby the value of the variables within this instance will get copied.

If the pragma **{attribute 'no_copy'}** is added to a variable, an online change copy of this variable will not be performed; instead this variable will be initialised. This might be reasonable in case of local pointer variables being assigned to the address of a variable actually shifted by the online change.

Syntax:

{attribute 'no_copy'}

Attribute No-exit

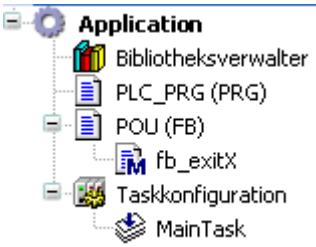
If a function block provides an exit method, you can suppress its call for a special instance with help of assigning the pragma **{attribute 'no-exit'}** to the function block instance.

Syntax:

{attribute 'symbol' := 'no-exit'}

Example:

Assume the exit method FB_Exit being added to a function block named "POU".



In program PLC_PRG two variables of type POU are instantiated:

```
PROGRAM PLC_PRG
VAR
POU1 : POU;
{attribute 'symbol' := 'no-exit'}
POU2 : POU;
END_VAR
```

When variable bInCopyCode becomes TRUE within POU1, the exit method FB_Exit is called exiting an instance that will get copied afterwards (online change). Though the value of variable bInCopyCode will have no influence on POU2.

Attribute Noinit

Variables provided with the pragma **{attribute 'no_init'}** won't be initialised implicitly. The pragma belongs to the variable declared subsequently.

Syntax:

{attribute 'no_init’}

also possible:

{attribute 'no-init’}

{attribute 'noinit’}

Example:

```
PROGRAM PLC_PRG
VAR
A : INT;
{attribute 'no_init'}
B : INT;
END_VAR
```

If a reset is performed on the associated application the integer variable "A" will be again initialised implicitly with 0, whereas variable "B" maintains the value it is currently assigned to.

Attribute No_virtual_actions

Assume function blocks, which are derived from a base function block implemented in SFC and which are using the main SFC workflow of the base class. The actions called therein show the same virtual behaviour as methods, that is, the base class actions may be overloaded by specific implementations related to the derived classes.

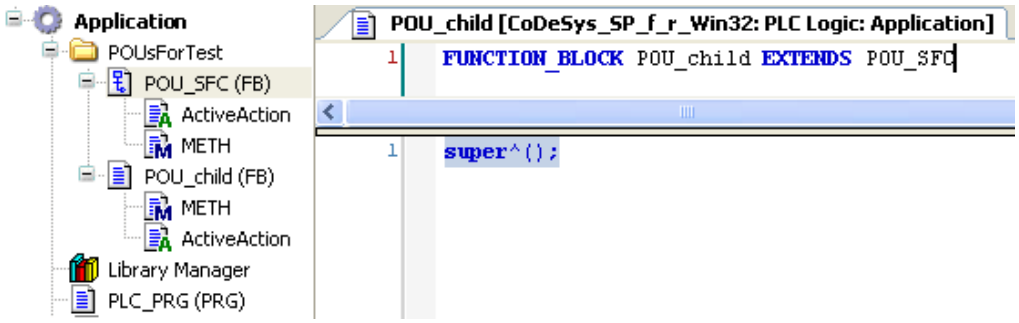
If the action of the base class shall be prevented from being overridden, the base class can be given the pragma {attribute 'no_virtual_actions'}.

Syntax:

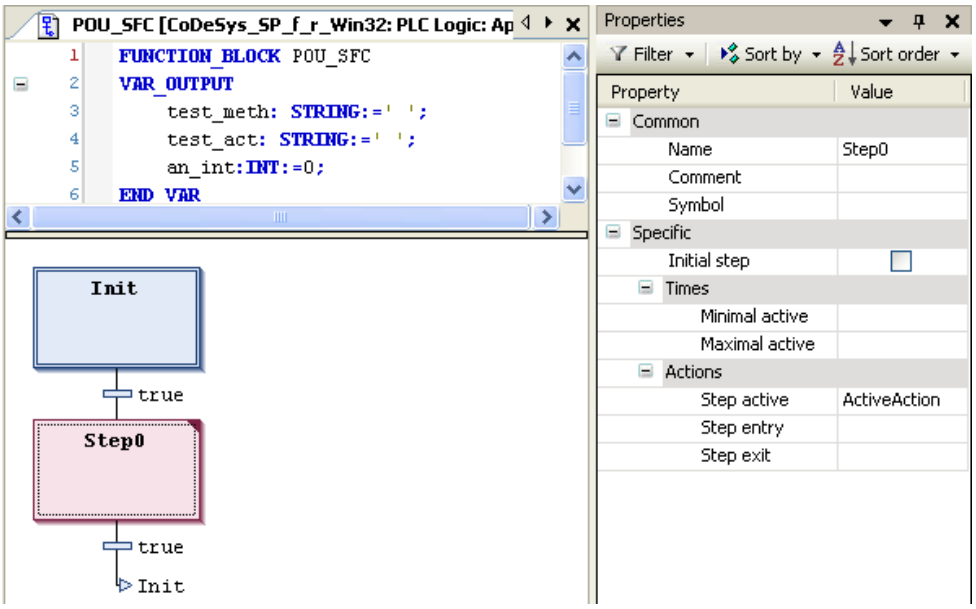
{attribute 'no_virtual_actions'}

Example:

In the following example the function block POU_SFC provides the base class to be extended by function block POU_child.



By use of the keyword SUPER the derived class POU_child calls the workflow of the base class that is implemented in SFC.



The exemplary implementation of this workflow is restricted to the initial step followed by one single step with associated step action ActiveAction concerned with the assignment of the output variables:

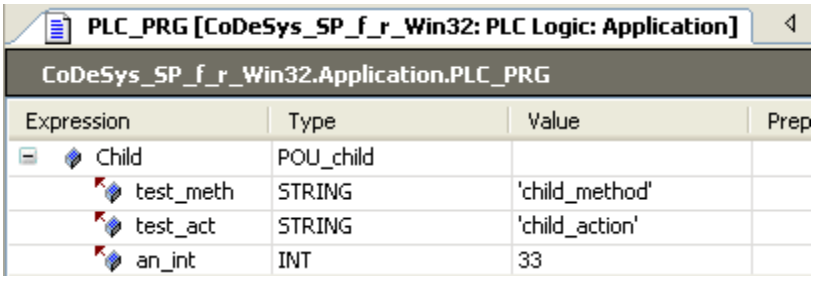
```
an_int:=an_int+1;              // integer counting the calls to the action
test_act:='father_action';     // assignment of the string variable test_act
METH();                        // call to method assigning the string variable test_meth
```

In case of the derived class POU_child the step action will be overwritten by a specific implementation of ActiveAction differing from the original one by assigning the string 'child_action' instead of 'father_action' to variable test_act.

Likewise the method METH, assigning the string "father_method" to variable test_meth within the base class, will be overwritten such, that test_meth will be assigned to "child_method" instead.

The main program PLC_PRG will execute repeated calls to "Child" (an instance of POU_child).

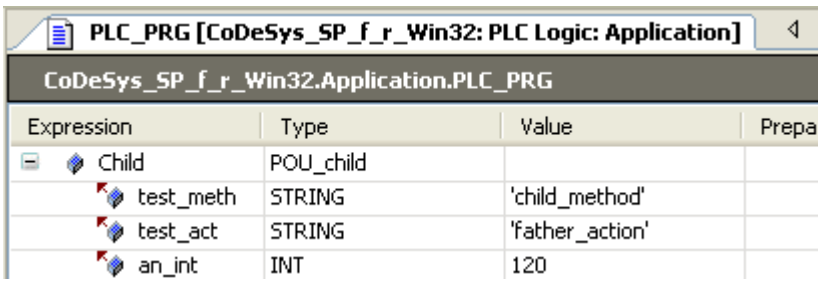
As expected, the actual value of the output strings report the call to action and method of the derived class:



However, if the base class is preceded by the attribute 'no_virtual_actions'

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK POU_SFC...
```

a different behaviour can be observed: whereas method METH will still be overwritten by its implementation within the derived class, a call of the step action will now result in a call of action ActiveAction of the base class. Therefore, test_act will be assigned to string 'father_action':



Attribute Obsolete

An "obsolete" pragma can be added to a data type definition in order to cause a user-defined warning during a build run, if the respective data type (structure, function block etc.) is used within the project. Thus for example you might announce that the data type should not be used any longer.

Unlike a locally used Message Pragma this warning is defined within the definition and thus "globally" for all instances of the data type.

This pragma instruction always is valid for the current line or - if placed in a separate line - for the subsequent line.

Syntax:

{attribute "obsolete' := *'user defined text'*}

Example:

The obsolete pragma is inserted in the definition of function block fb1:

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
i:INT;
END_VAR
...
```

If fb1 is used as a data type in a declaration, e.g. "fbinst: fb1;" the following warning will be dumped when the project is built: "datatype fb1 not valid"'

Attribute Pack_mode

The pragma **{attribute 'pack_mode'}** defines the mode a data structure is packed while being allocated. The attribute has to be set on top of a data structure and will influence the packing of the whole structure.

Syntax:

{attribute 'pack_mode' := '<Value>'}

The template <Value> included in single quotes has to be replaced by one of the following values available:

Value	Assigned pack mode
0	aligned, i.e. there will be no memory gaps
1	1-byte-aligned (identical to aligned)
2	2-byte-aligned, i.e. the maximum size of a memory gap is 1 byte
4	4-byte-aligned, i.e. the maximum size of a memory gap is 3 bytes
8	8-byte-aligned, i.e. the maximum size of a memory gap is 7 bytes

Example:

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
STRUCT
 Enable: BOOL;
 Counter: INT;
 MaxSize: BOOL;
 MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

A variable of dataytype myStruct will be instantiated aligned: If the address of its component Enable is 0x0100 for example, then the component Counter will follow on address 0x0101, MaxSize on 0x0103 and MaxSizeReached on 0x0104. With 'pack_mode'=2 Counter would be found on 0x0102, MaxSize on 0x0104 and MaxSizeReached on 0x0105.

Note: The attribute may also be applied to POUs. Though, you have to be careful with its application due to eventually existing internal pointers of the POU.

Attribute Parameterstringof

With the help of the pragma {attribute 'parameterstringof'} the instance name of a variable is made accessible by the visualization.

Syntax:

{attribute 'parameterstringof' := ' <variable>'}

Example:

In the main program e. g. the instance myDUT of the user defined data structure DUT is defined:

```
PROGRAM PLC_PRG
VAR
myDUT: DUT;
END_VAR
```

This instance shall be passed to visualization function block "Vis" (via parameter "instance") that is referenced by a frame of another visualization "MainVisu":

Elementname	GenElemInst_2
Clipping	☐
Show frame	☐
Scaletype	ANISOTROPIC
References	_3S.CoDeSys.VisualElem.StructuredTypeNode
References	
PLCWinNT.Application.Vis	
instance	PLC_PRG.myDut
Position	
Y	104

Abb. 541 Detail of properties of visualization element contained in MainVisu

In the associated interface editor of "Vis" we'll find, of course, the input/output variable "instance", but also another input variable named "instanceStr":

Vis [PLCWinNT: SPS-Logik: Application]

```
1  VAR_INPUT
2      {attribute 'parameterstringof' := 'instance'}
3      instanceStr : STRING;
4  END_VAR
5  VAR_IN_OUT
6      instance : DUT;
7  END_VAR
```

Although being an input variable "instanceStr" is not listed within the inputs of the reference (see above picture). This is, because the variable "instanceStr" carries the attribute 'parameterstringof' and will so be assigned automatically to the name of the variable associated to the attribute. In our example, the associated variable is instance; therefor the string variable instanceStr will get assigned to "PLC_PRG.myDUT" and can now be used by the visualization "Vis", e.g. as text variable defining a placeholder %s.

Attribute Qualified_Only

After assigning the pragma **{attribute 'qualified_only'}** on top of a global variable list, its variables can only be accessed by using the global variable name, e.g. gvl.g_var. This works even for variables of enumeration type and may be useful to avoid name mismatch with local variables.

Syntax:

{attribute 'qualified_only’}

Example:

Assume the following global variable list GVL with attribute 'qualified_only':

```
{attribute 'qualified_only'}
VAR_GLOBAL
iVar:INT;
END_VAR
```

Within a POU, e.g. program PLC_PRG, this global variable has to be called with the prefix GVL, e.g.

```
GVL.iVar:=5;
```

The following incomplete call of the variable will cause instead an error:

```
iVar:=5;
```

Attribute Reflection

The pragma **{attribute 'reflection'}** is attached to signatures. Due to performance reasons it is an obligatory attribute for POUs carrying the instance-path attribute.

Syntax:

{attribute 'reflection’}

Example:

see the example of the instance-path attribute.

Attribute Symbol

The pragma **{attribute 'symbol'}** defines which variables are to be handled in the Symbolconfiguration, which means that they will be exported as "symbols" into a symbol list, exported to a XML-file in the project directory as well as to a file not visible for the user and available on the target system for external access, e.g. by an OPC-Server. Variables provided with that attribute will be downloaded to the PLC even if they have not been configured or are not visible within the symbol configuration editor.

Regard that the symbol configuration must be available as an object below the respective application in the devices tree.

Syntax:

{attribute 'symbol' := 'none' | 'read' | 'write' | 'readwrite'}

Regard that access is only allowed on symbols coming from **programs** or **global variable lists**. For accessing a symbol the symbol name must be specified completely.

The pragma definition can be assigned to particular variables or collectively to all variables declared in a program:

- To be valid for a single variable the pragma has to be placed in the line before the variables declaration.
- To be valid for all variables contained in the declaration part of a program the pragma has to be placed in the first line of the declaration editor. Anyway also in this case the settings for particular variables might be modified by an explicit adding of a pragma.

The possible **access** on a symbol is defined by the pragma parameter 'none', 'read', 'write' or 'readwrite'. If no parameter is defined, the default 'readwrite' will be valid.

Example:

With the following configuration the variables A and B will be exported with read and write access. Variable D will be exported with read access.

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
A : INT;
B : INT;
{attribute 'symbol' := 'none'}
C : INT;
{attribute 'symbol' := 'read'}
D : INT;
END_VAR
```

Conditional Pragmas

The ExST (Extended ST) language supports several conditional pragma instructions which affect the code generation in the pre-compile or compile process.

The implementation code which will be regarded for compilation for example might depend on whether a certain data type or variable is declared, whether a type or variable has got a certain attribute, whether a variable has a certain data type, whether a certain POU or task is available resp. is part of the call tree, etc.

Note: It is not possible for a POU or GVL declared in the POUs tree to use a "{define...}" declared in an application. Defines in applications will only have effect on interfaces inserted below the respective application.

{define *identifier string*}	During preprocessing all subsequent instances of the identifier will be replaced with the given sequence of tokens, if the token string is not empty (which is allowed and well-defined). The identifier remains defined and in scope until the end of the object or until it is undefined in an {undefine} directive. Used e.g. for "conditional compilation", see below.
{undefine *identifier*} .	The identifier's preprocessor definition (by {define}, see above) will be removed, the identifier hence is "undefined". If the specified identifier is not currently defined, this pragma will be ignored.
{IF *expr*} ... {ELSIF *expr*} ... {ELSE} ... {END_IF}	These are pragmas for **conditional** compilation. The specified expressions *exprs* are required to be constant at compile time; they are evaluated in the order in which they appear until one of the expressions evaluates to a nonzero value. The text associated with the successful directive is preprocessed and compiled normally; the others are ignored. The order of the sections is determinate; however, the *elsif* and *else* sections are optional, and *elsif* sections may appear arbitrarily often. Within the constant *expr* several "conditional compilation operators" can be used, which are described below.

Conditional compilation operators

Within the constant expression *expr* of a conditional compilation pragma ({if} or {elsif}), see above, several operators can be used. These operators itself may not be undefined or redefined via {undefine} or {define}, respectively. Regard that these expressions as well as the definition done by {define} can also be used in the 'Compiler defines' in the Properties dialog of an object (category 'Build').

defined (*identifier*)	When applied to an identifier, its value is TRUE if that identifier has been defined with a {define} instruction and not later undefined using {undefine}; otherwise its value is FALSE.
	Example:
	Precondition: There are two applications App1 and App2. Variable "pdef1" is defined by a {define} instruction in App1, but not in App2.
	{IF defined (pdef1)}
	(*this code is processed in application "App1"*)
	{info 'pdef1 defined'}
	ivar := ivar + SINT#1;
	{ELSE}
	(*this code is processed in application "App2"*)
	{info 'pdef1 not defined'}
	ivar := ivar - SINT#1;
	Here additionally an example of a message pragma is included: Only information string "pdef1 defined" will be displayed in the message window when the application is built, because pdef1 actually is defined. Info message "pdef1 not defined" will be displayed in case pdef1 is not defined.

defined (variable:*variable*)	When applied to a variable, its value is TRUE if this particular variable is declared within the current scope; otherwise FALSE. Example: Precondition: There are two applications App1 and App2. Variable "hallo" is declared in App2, but not in App1. {IF defined (variable:g_bTest)} (*the following code is only processed in application "App2"*) g_bTest := x > 300; {END_IF}

defined (type:*identifier*)	When applied to a type identifier, its value is TRUE if a type with that particular name is declared; otherwise FALSE. Example: Precondition: There are two applications App1 and App2. Data type "DUT" is defined in App1, but not in App2. {IF defined (type:DUT)} (*the following code line only will be processed in App1*) bDutDefined := TRUE; {END_IF}

defined (pou:*pou-name*)	When applied to a pou-name, its value is TRUE if a POU or an action with that particular name is defined; otherwise FALSE.
	Example:
	Precondition: There are two applications App1 and App2. POU "CheckBounds" is available in App1, but not in App2.
	
	{IF defined (pou:CheckBounds)}
	(*the following code line is only processed in application "App1"*)
	arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
	{ELSE}
	(*the following code line is only processed in application "App2"*)
	arrTest[i] := arrTest[i]+1;
	{END_IF}

not yet implemented: defined (task:identifier)	When applied to a task identifier, its value is TRUE if a task with that particular name is defined; otherwise FALSE.
	Example:
	Precondition: There are two applications App1 and App2. Task "PLC_PRG_Task" is available in App1, but not in App2.
	
	{IF defined (task:PLC_PRG_Task)}
	(*the following code line is only processed in application "App1"*)
	erg := plc_prg.x;
	{ELSE}
	(*the following code line is only processed in application "App2"*)
	erg := prog.x;
	{END_IF}

not yet implemented: defined (resource:identifier)	When applied to a resource identifier, its value is TRUE if a resource with that particular name is defined; otherwise FALSE.
	Example:
	Precondition: There are two applications App1 and App2. A resource object "glob_var1" (global variable list) is available in App1, but not in App2.
	
	{IF defined (resource:glob_var1)}
	(*the following code line only will be processed in application "App1"*)
	gvar_x := gvar_x + ivar;
	{ELSE}
	(*the following code line is only processed in application "App2"*)
	x := x + ivar;
	{END_IF}

hasattribute (pou: pou-name, 'attribute')	When applied to a POU, its value is TRUE if this particular attribute is specified in the first line of the POUs declaration part. Example: Precondition: There are two applications App1 and App2. Function "fun1" is defined in App1 and App2, but in App1 additionally has got an attribute 'vision': Definition of fun1 in App1: {attribute 'vision'} FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR VAR END_VAR Definition of fun1 in App2: FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR VAR END_VAR {IF hasattribute (pou: fun1, 'vision')} (*the following code line will only be processed in App1*) ergvar := fun1(ivar); {END_IF}

hasattribute (variable: *variable*, 'attribute')	When applied to a variable, its value is TRUE if this particular attribute is specified via the {attribute} instruction in a line before the variable's declaration. Example Precondition: There are two applications App1 and App2. Variable g_globalInt is used in App1 and App2, but in App1 additionally has got an attribute 'DoCount' : 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 {IF hasattribute (variable: g_globalInt, 'DoCount')} (*the following code line will only be processed in App1, because there variable g_globalInt has got the attribute 'DoCount'*) g_globalInt := g_globalInt + 1; {END_IF}

hastype (variable:*variable*, *type-spec*)	When applied to a variable, its value is TRUE if this particular variable has the specified *type-spec*; otherwise FALSE. ANY ANY_DERIVED ANY_ELEMENTARY ANY_MAGNITUDE ANY_BIT ANY_STRING ANY_DATE ANY_NUM ANY_REAL ANY_INT 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: Precondition: There are two applications App1 and App2. Variable "g_multitype" is declared in App1 with type LREAL and in application App2 with type STRING: {IF (hastype (variable: g_multitype, LREAL))} (*the following code line will be processed only in App1*) g_multitype := (0.9 + g_multitype) * 1.1; {ELSIF (hastype (variable: g_multitype, STRING))} (*the following code line will be processed only in App2*) g_multitype := 'dies ist ein multitalent'; {END_IF}

hasvalue (*define-ident*, *char-string*)	If the define (*define-ident*) is defined and it has the specified value (*char-string*), then its value is TRUE, otherwise FALSE
	Example:
	Precondition: Variable "test" is used in applications App1 and App2, it gets value "1" in App1 and value "2" in App2:
	{IF hasvalue(test,'1')}
	(*the following code line will be processed in App1, because there variable test has value 1*)
	x := x + 1;
	{ELSIF hasvalue(test,'2')}
	(*the following code line will be processed in App2, because there variable test has value 2*)
	x := x + 2;
	{END_IF}

NOT *operator*	Inverts the value of the operator. Example: Precondition: There are two applications App1 and App2. POU "PLC_PRG1" is used in App1 and App2, POU CheckBounds is only available in App1: {IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))} (*the following code line is only executed in App2*) bANDNotTest := TRUE; {END_IF}
operator AND *operator*	Is TRUE if both operators are TRUE Example: Precondition: POU "PLC_PRG1" is used in applications App1 and App2, POU CheckBounds is only available in App1: {IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))} (*the following code line will be processed only in applications App1, because only there "PLC_PRG1" and "CheckBounds" are defined*) bORTest := TRUE; {END_IF}

operator OR *operator*	Is TRUE if one of the operators are TRUE Example: Precondition: POU "PLC_PRG1" is used in applications App1 and App2, POU CheckBounds is only available in App1: {IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))} (*the following code line will be processed in applications App1 and App2, because both contain at least one of the POUs "PLC_PRG1" and "CheckBounds"*) bORTest := TRUE; {END_IF}
(operator)	braces the operator

16.1.27 List Components Functionality

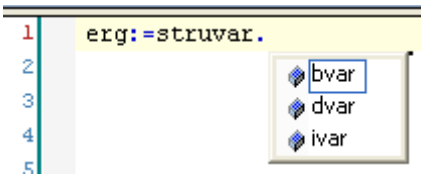
The text input is supported noticing standard IEC 61131-3. The 'IEC 61131-3 Text Editor' plug-in provides a kind of "List Components" functionality, which can be activated or deactivated in a project in the Text Editor options.

"List Components" helps to insert a correct identifier:

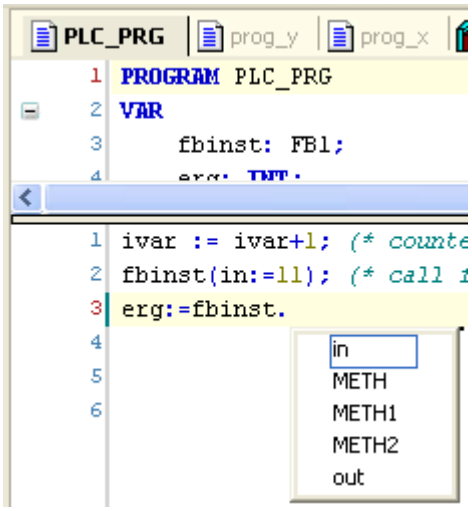
- If you - at any place, where a global identifier can be inserted - insert a dot "." instead of the identifier, a selection box will appear, listing all currently available global variables. You can choose one of these elements and press <Return> to insert it behind the dot. You can also insert the element by a doubleclick on the list entry.
- If you enter a function block instance or a structure variable followed by a dot, then a selection box listing all input and output variables of the corresponding function block resp. listing the structure components will appear, where you can choose the desired element and enter it by pressing <Return>or by a doubleclick.

Examples

"List Components" offers components of structure:



"List Components" offers components of a function block:



- If you enter any string and press **<Ctrl> + <Space Bar>**, a selection box will appear listing all POUs and global variables available in the project. The first list entry, which is starting with the given string, will be selected and can be entered to the program by pressing the <Return> key.

16.2 Data Types

You can use standard data types, user-defined data types or instances of function blocks when programming. Each identifier is assigned to a data type which dictates how much memory space will be reserved and what type of values it stores.

16.2.1 Standard Data Types

All data types described by standard IEC61131-3 are supported by »PLC Designer«. See here the following

- BOOL
- Integer Data Types
- REAL / LREAL
- STRING
- WSTRING
- Time Data Types

Notice that there are also some norm-extending data types and that you can also define types on your own (user defined data types).

bool

BOOL type variables may be given the values TRUE (1) and FALSE (0). 8 bits of memory space will be reserved.

See also: BOOL constants (Operands)

Integer Data Types

See below a list of all available integer data types. Each of the different number types covers a different range of values. The following range limitations apply to the integer data types:

Type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
LWORD	0	2^{64-1}	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit
LINT	-2^{63}	2^{63-1}	64 Bit
ULINT	0	2^{64-1}	64 Bit

As a result when larger types are converted to smaller types, information may be lost.

See also: Number constants (Operands)

REAL / LREAL

REAL and LREAL are so-called floating-point types. They are required to represent rational numbers. 32 bits of memory space is reserved for REAL and 64 bits for LREAL.

Value range for REAL: 1.175494351e-38 to 3.402823466e+38

Value range for LREAL: 2.2250738585072014e-308 to 1.7976931348623158e+308

Note: The support of data type LREAL depends on the target device. Please see in the corresponding documentation whether the 64 bit type LREAL gets converted to REAL during compilation (possibly with a loss of information) or persists.

See also: REAL-/LREAL constants (Operands)

STRING

A STRING type variable can contain any string of characters. The size entry in the declaration determines how much memory space should be reserved for the variable. It refers to the number of characters in the string and can be placed in parentheses or square brackets. If no size specification is given, the default size of 80 characters will be used.

The string length basically is not limited in »PLC Designer«, but string functions only can process strings of 1 - 255 characters! If a variable is initialized with a string too long for the variables datatype, the string will be correspondingly cut from right to left.

Example of a String Declaration with 35 characters:

```
str:STRING(35):='This is a String';
```

See also:

- WSTRING
- STRING constants (Operands)

Time Data Types

The data types **TIME**, **TIME_OF_DAY** (abb. **TOD**), **DATE** and **DATE_AND_TIME** (abb. **DT**) are handled internally like DWORD.

Time is given in milliseconds in TIME and TOD, time in TOD begins at 12:00 A.M.

Time is given in seconds in DATE and DT beginning with January 1, 1970 at 12:00 A.M.

Notice in this context also:

- LTIME, which in extension to the IEC 61131-3 standard is available as a 32-Bit time data type.
- TIME constants
- DATE constants
- DATE_AND_TIME constants
- TIME_OF_DAY constants

16.2.2 Extensions to Standard IEC1131-3

Norm-extending Data Types

In addition to the data types according to IEC1131-3 there are also some norm-extending data types available implicitly in »PLC Designer«:

- UNION
- ANY_TYPE
- LTIME
- WSTRING
- References
- Pointers

union

As an **extension to the IEC 61131-3 standard** it is possible to declare unions in user-defined types.

In a union all components have the same offset, that is they all occupy the same storage location. Thus, assuming a union definition as shown in the following example, an assignment to name.a also would manipulate name.b.

Example:

```
TYPE name: UNION
a : LREAL;
b : LINT;
END_UNION
END_TYPE
```

ANY_TYPE

not yet implemented: As an extension to the IEC 61131-3 standard it is possible to declare inputs of functions or methods as to be of type ANY_TYPE.

LTIME

As **extension to the IEC 61131-3 LTIME** is supported as time base for high resolution timers. LTIME is of size 64 Bit and resolution nanoseconds.

Syntax:

LTIME#<time declaration>

The time declaration can include the time units as used with the TIME constant and additionally:

"us" : microseconds

"ns" : nanoseconds

Example:

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns

Compare to TIME: size 32 Bit and resolution milliseconds.

wstring

This string data type is an **extension to the IEC 61131-3 standard**.

It differs from the standard STRING type (ASCII) by getting interpreted in Unicode format.

Example:

```
wstr:WSTRING:="This is a WString";
```

See also:

- STRING
- STRING constants (Operands)

References

This data type is an **extension to the IEC 61131-3 standard**.

A REFERENCE is an alias for an object. The alias can be written or read via identifiers. The difference to a pointer is that the value pointed to is directly affected and that the assignment of reference and value is fix. The address of the reference must be set via a separate assignement operation. Whether a reference points to a valid value (that is unequal 0) can be checked with the help of a special operator, see below.

A reference is declared according to the following syntax:

Syntax

<identifier> : REFERENCE TO <data type>

Example declaration:

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

ref_int is now available for being used as an alias for variables of type INT.

Example of use:

```
ref_int REF= a;   (* ref_int now points to a *)
ref_int := 12;    (* a does now have the value 12 *)
b := ref_int * 2;  (* b does now have the value 24 *)
ref_int REF= b;   (* ref_int does now point to b *)
ref_int := a / 2;  (* b does now have the value 6 *)
ref_int REF= 0;   (* explicit initialization of the reference *)
```

Note: It is not possible to declare references like REFERENCE TO REFERENCE or ARRAY OF REFERENCE or POINTER TO REFERENCE.

Note: With compiler version >= V3.3.0.0 references will be initialized (with 0).

Check for valid references

Operator "__ISVALIDREF" can be used to check whether a reference points to a valid value, that is a value unequal 0.

Syntax:

<boolean variable> := __ISVALIDREF(<identifier declared with type REFERENCE TO <datatype>);

<boolean variable> will be TRUE, if the reference points to a valid value, FALSE if not.

Example:

Declaration:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
```

Implementation:

```
ivar := ivar +1;
ref_int REF= hugo;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);  (* will be TRUE, because ref_int points to ivar, which is unequal 0 *)
testref0 := __ISVALIDREF(ref_int0);   (* will be FALSE, because ref_int is set to 0 *)
```

Pointers

As an **extension to the IEC 61131-3 standard** it is possible to use pointers.

Pointers save the addresses of variables, programs, function blocks, methods and functions while an application program is running. A pointer can point to any of those objects and to any data type, even to user-defined data types. Notice the possibility of using an implicit pointer check function.

Syntax of a pointer declaration:

<identifier>: POINTER TO <data type | function block | program | method | function>;

Dereferencing a pointer means to obtain the value currently stored at the address to which it is pointing A pointer can be dereferenced by adding the content operator "^" after the pointer identifier; see for example "pt^" in the example below.

The Address Operator ADR can be used to 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); (* address of varint1 is assigned to pointer pt *)
var_int2:= pt^;     (* value 5 of var_int1 gets assigned to var_int2 via dereferencing
of pointer pt; *)
```

Function pointers

In contrast to »PLC Designer« V2.3 also **function pointers** are supported, replacing the INDEXOF operator. This pointers can be passed to external libraries, but there is **no possibility to call a function pointer within** an application in the programming system! The runtime function for registration of callback functions (system library function) expects the function pointer, and, depending on the callback for which the registration was requested, then the respective function will be called implicitly by the runtime system (for example at STOP). In order **to enable such a system call** (runtime system) the respective property (category "Build") must be set for the function object.

The ADR operator can be used on function names, program names, function block names and method names. Since functions can move after online change, the result is not 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.

Index access to pointers

As **extension to the IEC 61131-3 standard**, index access "[]" to variables of type POINTER, STRING and WSTRING is allowed.

- pint[i] will return the base data type
- Index access on pointers is arithmetic: If the index access is used on a variable of type pointer, the offset will be calculated by: pint[i] = (pint + i * SIZEOF(base type))^. The index access also performs an implicit dereferencation on the pointer. The result type is the base type of the pointer. Note that pint[7] != (pint + 7)^!
- If the index access is used on a variable of type STRING, the result is the character at offset index-expr. The result is of type BYTE. str[i] will return the i-th character of the string as a SINT (ASCII).
- If the index access is used on a variable of type WSTRING the result is the character at offset index-expr. The result is of type WORD. wstr[i] will return the i-th character of the string as INT (Unicode).

Note: There is also the possibility of using References, which in contrast to a pointer directly affect a value.

CheckPointer function

For checking pointer access during runtime you might use the implicitly available check function "CheckPointer" being called before each access on the address of a pointer. Therefore add the object "POUs for implicit checks" to the application using the Add Object dialog. Mark the checkbox related to the type CheckPointer, choose an implementation language and confirm your settings with **Open,** whereon the check function will be opened in the editor corresponding to the implementation language selected. Independently of that choice the declaration part is preset and may not be modified except for adding further local variables! However, in contrast to other check functions, there is no default implementation of CheckPointer available, the implementation is left to the user!

Function CheckPointer should check whether the address the pointer refers to is within the valid memory range. In addition it should be taken care of that the alignment of the referenced memory area fits to the data type of the variable the pointer points to. If both conditions are fulfilled, CheckPointer should return the unchanged input pointer. A proper handling of detected error cases is left to the user.

Template :

Declaration part:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

Implementation part:

```
// No standard way of implementation. Fill your own code here
CheckPointer := ptToTest;
```

When called the function gets the following input parameters:

- ptToTest: Target address of the pointer
- iSize: Size of referenced variable; the data type of iSize must be integer-compatible and must cover the maximum potential data size stored at the pointer address
- iGran: Granularity of the access, that is the largest not-structured datatype used in the referenced variable; the data type of iGran must be integer-compatible
- bWrite: type of access (TRUE=write access, FALSE= read access); the data type of bWrite must be BOOL

Return value: Address which is used for dereferencing the pointer, thus at best the one that has been passed on as the first input parameter (ptToTest).

16.2.3 User Defined Data Types

Defined Data Types

Additionally to the standard data types the user can define special data types within a project.

These definitions are possible via creating DUT (Data Unit Type) objects in the POUs window resp. within the declaration part of a POU.

Please notice the given recommendations on the naming of objects in order to make it as unique as possible.

See the following user defined data types:

- ARRAY
- Structure
- Enumeration
- Subrange type
- Reference
- Pointer

Arrays

One-, two-, and three-dimensional fields (arrays) are supported as elementary data types. Arrays can be defined both in the declaration part of a POU and in the global variable lists. Notice the possibility of using implicit boundary checks.

Syntax:

<Field_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>

ll1, ll2, ll3 identify the lower limit of the field range; ul1, ul2 and ul3 identify the upper limit. The range values must be integers.

Example:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Initializing Arrays:

Attention: In contrast to »PLC Designer« V2.3 now squared brackets must be put around the initialization part !

Example for complete initialization of an array:

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];
arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)]; (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];
    (* short for 0,0,4,4,4,4,2,3 *)
```

Example of the initialization of an array of a structure:

Structure definition:

```
TYPE STRUCT1
STRUCT
 p1:int;
 p2:int;
 p3:dword;
END_STRUCT
END_TYPE
```

Array initialization:

```
ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299),
(p1:=14,p2:=5,p3:=112)];
```

Example of the partial initialization of an Array:

```
arr1 : ARRAY [1..10] OF INT := [1,2];
```

Elements to which no value is pre-assigned, are initialized with the default initial value of the basic type. In the example above, the elements anarray[6] to anarray[10] are therefore initialized with 0.

Accessing array components:

Array components are accessed in a two-dimensional array using the following syntax:

<Field_Name>[Index1,Index2]

Example:

```
Card_game [9,2]
```

Check Functions

In order to a proper access to array elements during runtime the function CheckBounds must be available to the application. Therefore add the object "POUs for implicit checks" to the application using the Add Object dialog. Mark the checkbox related to the type CheckBounds, choose an implementation language and confirm your settings with **Open,** whereon the function CheckBound will be opened in the editor corresponding to the implementation language selected. Independently of that choice the declaration part is preset and may not be modified except for adding further local variables! A proposal default implementation of the function that might be modified is given in the ST Editor.

This check function has to treat boundary violations by an appropriate method (for example by setting a detected error flag or changing the index). The function will be called implicitly as soon as a variable of type array is assigned.

Attention: In order to maintain the check functionality do not change the declaration part of an implicit check function.

Example for the use of function CheckBounds:

The default implementation of the check function is the following

Declaration part:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
index, lower, upper: DINT;
END_VAR
```

Implementation part:

```
// Implicitly generated code : Only an Implementation suggestion
IF  index < lower THEN
CheckBounds := lower;
ELSIF  index > upper THEN
CheckBounds := upper;
ELSE
CheckBounds := index;
END_IF
```

When called the function gets the following input parameters:

- index: field element index
- lower: the lower limit of the field range
- upper: the upper limit of the field range

As long as the index is within the range, the return value is the index itself. Otherwise - in correspondence to the range violation- either the upper or the lower limit of the filed range will be returned.

In the programm beyond the upper limit of the array "a" is exceeded:

```
PROGRAM PLC_PRG
VAR
 a: ARRAY[0..7] OF BOOL;
 b: INT:=10;
END_VAR
a[b]:=TRUE;
```

In this case the implicit call to the CheckBounds function preceding the assignment effects that the value of the index is changed from "10" into the upper limit "7". Therefore the value TRUE will be assigned to the element a[7] of the array. This is how attempted access outside the field range can be corrected via the function CheckBounds.

Structures

Structures are created as "DUT" (Data Type Unit) objects via the Add Object dialog.

They begin with the keywords TYPE and STRUCT and end with END_STRUCT and END_TYPE.

Attention: In contrast to »PLC Designer« V2.3 TYPE in structure declarations must be followed by a ":" !

The syntax for structure declarations is as follows:

```
TYPE <structurename>:
STRUCT
 <declaration of variables 1>
 .
 .
 <declaration of variables n>
END_STRUCT
END_TYPE
```

<structurename> is a type that is recognized throughout the project and can be used like a standard data type.

Interlocking structures are allowed. The only restriction is that variables may not be assigned to addresses (the AT declaration is not allowed!).

Example for a structure definition named Polygonline:

```
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
```

Initialization of structures:

Example for the initialization of a structure:

```
Poly_1:polygonline := ( Start:=[3,3], Point1 =[5,2], Point2:=[7,3], Point3:=[8,5],
Point4:=[5,7], End := [3,5]);
```

Initializations with variables are not possible. See an example of the initialization of an array of a structure under 'Arrays'.

Access on structure components:

You can gain access to structure components using the following syntax:

<Structure_Name>.<Componentname>

So for the above mentioned example of the structure 'polygonline' you can access the component 'start' by `Poly_1.Start`.

Enumerations

An enumeration is an user-defined data type that is made up of a number of string constants. These constants are referred to as enumeration values.

Enumeration values are recognized globally in all areas of the project even if they were declared within a POU.

An enumeration is created as a "DUT" object via the Add Object dialog.

Attention: In contrast to »PLC Designer« V2.3 a local enumeration declaration is no longer possible except within TYPE.

Syntax:

TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>) |<base data type>;
END_TYPE

A variable of type <Identifier> can take on one of the enumeration values <Enum_..> and will be initialized with the first one. These values are compatible with whole numbers which means that you can perform operations with them just as you would do with integer variables. You can assign a number x to the variable. If the enumeration values are not initialized with specific values within the declaration, counting will begin with 0. When initializing, make sure that the initial values are increasing within the row of components. The validity of the number will be checked at the time it is run.

Example:

```
TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10);  (* The initial value for each of the colors is red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* The value of the traffic signal is red*)
FOR i:= red TO green DO
 i := i + 1;
END_FOR;
```

Extensions to the IEC 61131-3 standard:

1. The type name of enumerations can be used (as a scope operator) to disambiguate the access to an enumeration constant.

   So it is possible to use the **same constant in different enumerations**.

Example:

Definition of two enumerations:

```
TYPE COLORS_1: (red, blue);
END_TYPE
TYPE COLORS_2: (green, blue, yellow);
END_TYPE
```

Use of enumeration value blue in a POU:

Declaration:

```
colorvar1 : COLORS_1;
colorvar2 : COLORS_2;
```

Implementation:

```
(* possible: *)
colorvar1 := colors_1.blue;
colorvar2 := colors_2.blue;
(* not possible: *)
colorvar1 := blue;
colorvar2 := blue;
```

2. The base data type of the enumeration - which by default is INT - can be explicitly specified.

Example:

The base data type for enumeration BigEnum should be DINT:

```
TYPE BigEnum : (yellow, blue, green:=16#8000) DINT;
END_TYPE
```

Subrange types

A subrange type is a user defined type whose range of values is only a subset of that of the basic data type. Notice the possibility of using implicit range boundary checks.

The declaration can be done in a DUT object but also a variable can be directly declared with a subrange type:

Syntax for the declaration as a DUT object:

TYPE <Name> : <Inttype> (<ug>..<og>) END_TYPE;

<Name>	must be a valid IEC identifier,
<Inttype>	is one of the data types SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD).
<ug>	Is a constant which must be compatible with the basic type and which sets the lower boundary of the range types. The lower boundary itself is included in this range.
<og>	Is a constant that must be compatible with the basic type, and sets the upper boundary of the range types. The upper boundary itself is included in this basic type.

Examples:

```
TYPE
 SubInt : INT (-4095..4095);
END_TYPE
```

Direct declaration of a variable with a subrange type:

```
VAR
 i : INT (-4095..4095);
 ui : UINT (0..10000);
END_VAR
```

If a value is assigned to a subrange type (in the declaration or in the implementation) but does not match this range (for example i:=5000 in the upper shown declaration example), an error message will be issued.

Check Functions

In order to check the range limits during runtime the functions **CheckRangeSigned** or **CheckRangeUnsigned** must be available to the application. Therefore add the object "POUs for implicit checks" to the application using the Add Object dialog. Mark the checkbox related to the type CheckRangeSigned or CheckRangeUnsigned, choose an implementation language and confirm your settings with **Open,** whereon the selected function will be opened in the editor corresponding to the implementation language selected. Independently of that choice the declaration part is preset and may not be modified except for adding further local variables! A proposal default implementation of the function that might be modified is given in the ST Editor.

The purpose of this check function is the proper treatment of violations of the subrange (for example by setting a detected error flag or changing the value). The function will be called implicitly as soon as a variable of subrange type is assigned.

Attention: In order to maintain the check functionality do not change the declaration part of an implicit check function!

Example:

The assignment of a variable belonging to a signed subrange type entail an implicit call to CheckRangeSigned. The default implementation of that function trimming a value to the permissible range is provided as follows:

Declaration part:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
value, lower, upper: DINT;
END_VAR
```

Implementation part:

```
// Implicitly generated code : Only an Implementation suggestion
IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
END_IF
```

When called the function gets the following input parameters:

- value: the value to be assigned to the range type
- lower: the lower boundary of the range
- upper: the upper boundary of the range

As long as the assigned value is within the range, the output of the function is the value itself. Otherwise -in correspondence to the range violation- either the upper or the lower boundary of the range will be returned.

The assignment i:=10*y will now be replaced implicitly by

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

If y has the value "1000" for example, the variable i will not be assigned to "10*1000=10000" as provided by the original implementation, but to the upper boundary of the range, that is "4095".

The same applies to function CheckRangeUnsigned.

Note: If neither of the functions CheckRangeSigned or CheckRangeUnsigned is present, no type checking of subrange types occurs during runtime! In this case variable i could get any value between –32768 and 32767 at any time!

Attention: The use of the functions CheckRangeSigned and CheckRangeUnsigned may result in an endless loop, for example if a subrange type is used as increment of a FOR loop that does not match the subrange.

Example of an endless loop:

```
VAR
 ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
...
END_FOR
```

The FOR loop will never be left as the check function prevents the variable ui to be assigned to values greater than 10000.

References

This data type is an **extension to the IEC 61131-3 standard**.

A REFERENCE is an alias for an object. The alias can be written or read via identifiers. The difference to a pointer is that the value pointed to is directly affected and that the assignment of reference and value is fix. The address of the reference must be set via a separate assignement operation. Whether a reference points to a valid value (that is unequal 0) can be checked with the help of a special operator, see below.

A reference is declared according to the following syntax:

Syntax

<identifier> : REFERENCE TO <data type>

Example declaration:

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

ref_int is now available for being used as an alias for variables of type INT.

Example of use:

```
ref_int REF= a;   (* ref_int now points to a *)
ref_int := 12;    (* a does now have the value 12 *)
b := ref_int * 2;  (* b does now have the value 24 *)
ref_int REF= b;   (* ref_int does now point to b *)
ref_int := a / 2;  (* b does now have the value 6 *)
ref_int REF= 0;   (* explicit initialization of the reference *)
```

Note: It is not possible to declare references like REFERENCE TO REFERENCE or ARRAY OF REFERENCE or POINTER TO REFERENCE.

Note: With compiler version >= V3.3.0.0 references will be initialized (with 0).

Check for valid references

Operator "__ISVALIDREF" can be used to check whether a reference points to a valid value, that is a value unequal 0.

Syntax:

<boolean variable> := __ISVALIDREF(<identifier declared with type REFERENCE TO <datatype>);

<boolean variable> will be TRUE, if the reference points to a valid value, FALSE if not.

Example:

Declaration:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
```

Implementation:

```
ivar := ivar +1;
ref_int REF= hugo;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);  (* will be TRUE, because ref_int points to ivar, which is unequal 0 *)
testref0 := __ISVALIDREF(ref_int0);   (* will be FALSE, because ref_int is set to 0 *)
```

Pointers

As an **extension to the IEC 61131-3 standard** it is possible to use pointers.

Pointers save the addresses of variables, programs, function blocks, methods and functions while an application program is running. A pointer can point to any of those objects and to any data type, even to user-defined data types. Notice the possibility of using an implicit pointer check function.

Syntax of a pointer declaration:

<identifier>: POINTER TO <data type | function block | program | method | function>;

Dereferencing a pointer means to obtain the value currently stored at the address to which it is pointing A pointer can be dereferenced by adding the content operator "^" after the pointer identifier; see for example "pt^" in the example below.

The Address Operator ADR can be used to 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); (* address of varint1 is assigned to pointer pt *)
var_int2:= pt^;     (* value 5 of var_int1 gets assigned to var_int2 via dereferencing
of pointer pt; *)
```

Function pointers

In contrast to »PLC Designer« V2.3 also **function pointers** are supported, replacing the INDEXOF operator. This pointers can be passed to external libraries, but there is **no possibility to call a function pointer within** an application in the programming system! The runtime function for registration of callback functions (system library function) expects the function pointer, and, depending on the callback for which the registration was requested, then the respective function will be called implicitly by the runtime system (for example at STOP). In order **to enable such a system call** (runtime system) the respective property (category "Build") must be set for the function object.

The ADR operator can be used on function names, program names, function block names and method names. Since functions can move after online change, the result is not 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.

Index access to pointers

As **extension to the IEC 61131-3 standard**, index access "[]" to variables of type POINTER, STRING and WSTRING is allowed.

- pint[i] will return the base data type
- Index access on pointers is arithmetic: If the index access is used on a variable of type pointer, the offset will be calculated by: pint[i] = (pint + i * SIZEOF(base type))^. The index access also performs an implicit dereferencation on the pointer. The result type is the base type of the pointer. Note that pint[7] != (pint + 7)^!
- If the index access is used on a variable of type STRING, the result is the character at offset index-expr. The result is of type BYTE. str[i] will return the i-th character of the string as a SINT (ASCII).
- If the index access is used on a variable of type WSTRING the result is the character at offset index-expr. The result is of type WORD. wstr[i] will return the i-th character of the string as INT (Unicode).

Note: There is also the possibility of using References, which in contrast to a pointer directly affect a value.

CheckPointer function

For checking pointer access during runtime you might use the implicitly available check function "CheckPointer" being called before each access on the address of a pointer. Therefore add the object "POUs for implicit checks" to the application using the Add Object dialog. Mark the checkbox related to the type CheckPointer, choose an implementation language and confirm your settings with **Open,** whereon the check function will be opened in the editor corresponding to the implementation language selected. Independently of that choice the declaration part is preset and may not be modified except for adding further local variables! However, in contrast to other check functions, there is no default implementation of CheckPointer available, the implementation is left to the user!

Function CheckPointer should check whether the address the pointer refers to is within the valid memory range. In addition it should be taken care of that the alignment of the referenced memory area fits to the data type of the variable the pointer points to. If both conditions are fulfilled, CheckPointer should return the unchanged input pointer. A proper handling of detected error cases is left to the user.

Template :

Declaration part:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

Implementation part:

```
// No standard way of implementation. Fill your own code here
CheckPointer := ptToTest;
```

When called the function gets the following input parameters:

- ptToTest: Target address of the pointer
- iSize: Size of referenced variable; the data type of iSize must be integer-compatible and must cover the maximum potential data size stored at the pointer address
- iGran: Granularity of the access, that is the largest not-structured datatype used in the referenced variable; the data type of iGran must be integer-compatible
- bWrite: type of access (TRUE=write access, FALSE= read access); the data type of bWrite must be BOOL

Return value: Address which is used for dereferencing the pointer, thus at best the one that has been passed on as the first input parameter (ptToTest).

16.3 Operators

16.3.1 IEC Operators and norm-extending Functions

»PLC Designer« V3.x supports all IEC operators. In contrast to the standard functions these operators are recognized implicitly throughout the project.

Besides the IEC operators **also the following operators are supported which are not prescribed by the standard**: ANDN, ORN, XORN, INDEXOF and SIZEOF (see Arithmetic Operators), ADR, BITADR and content operator (see Address Operators), some "scope operators".

Operators are used like functions in a POU.

Note: At operations with floating point variables the result depends on the currently used target system !

See the following categories of operators:

- Assignment Operators: :=, MOVE
- Arithmetic Operators
- Bitstring Operators
- Bit-Shift Operators
- Selection Operators
- Comparison Operators
- Address Operators
- Calling Operator
- Type Conversion Functions
- Numeric Functions
- IEC extending Operators
- IEC extending "Scope Operators"

16.3.2 Arithmetic Operators

The following operators, prescribed by the IEC1131-3 standard, are available:

- ADD, MUL, SUB, DIV, MOD, MOVE

There are also two norm-extending operators:

- INDEXOF, SIZEOF

ADD

IEC Operator: Addition of variables.

Allowed types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

Two TIME variables can also be added together resulting in another time (for example, t#45s + t#50s = t#1m35s)

Example in IL:

```
LD    7
ADD   2
ADD   4
ADD   7
ST    iVar
```

Example in ST:

```
var1 := 7+2+4+7;
```

Example in FBD:

MUL

IEC Operator: Multiplication of variables .

Allowed types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

Example in IL:

```
LD    7
MUL   2    ,
      4    ,
      7
ST    Var1
```

Example in ST:

```
var1 := 7*2*4*7;
```

Example in FBD:

SUB

IEC Operator: Subtraction of one variable from another one.

Allowed types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

A TIME variable may also be subtracted from another TIME variable resulting in third TIME type variable. Note that negative TIME values are undefined.

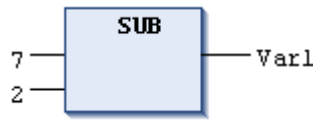
Example in IL:

```
LD    7
SUB   2
ST    Var1
```

Example in ST:

```
var1 := 7-2;
```

Example in FBD:



DIV

IEC Operator: Division of one variable by another one.

Allowed types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

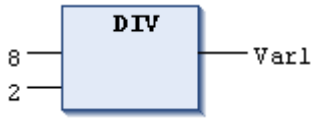
Example in IL: (Result in Var1 is 4)

```
LD      8
DIV     2
ST      Var1
```

Example in ST:

```
var1 := 8/2;
```

Example in FBD:



Note: Please notice, that different target systems may behave differently concerning a division by zero.

Check functions

In order to check the value of the divisor, for example in order to avoid a division by 0 you may make use of the provided check functions **CheckDivInt**, **CheckDivLint**, **CheckDivReal** and **CheckDivLReal.** After they have been included in the application each division occurring in the related code will provoke a preceding call to these functions. To include them in the application use the Add Object dialog. Therein choose the object "POUs for implicit checks", mark the checkbox of a corresponding check function, select an implementation language and confirm your choice with **Open**. The selected function will be opened in the editor corresponding to the choice of the implementation function. Independently of this choice the declaration part of the functions is preset and must not be changed except for adding local variables. A default implementation of the functions that might be modified is available in ST.

See the following example for an implementation of the function CheckDivReal:

Default implementation of the function CheckDivReal:

Declaration part:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
 divisor:REAL;
END_VAR
```

Implementation part:

```
// Implicitly generated code : only an suggestion for implementation
IF divisor = 0 THEN
 CheckDivReal:=1;
ELSE
 CheckDivReal:=divisor;
END_IF;
```

The operator DIV uses the output of function CheckDivReal as divisor. In the following example a division by 0 is prohibited as the implicitly with 0 initialized value of the divisor "d" is changed to "1" by CheckDivReal prior to the execution of the division. Therefore the result of the division is 799.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL;
END_VAR
erg:= v1 / d;
```

MOD

IEC Operator: Modulo Division of one variable by another one.

Allowed types: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT. The result of this function will be the remainder of the division. This result will be a whole number.

Example in IL: (Result in Var1 is 1)

```
LD    9
MOD   2
ST    Var1
```

Example in ST:

```
var1 := 9 MOD 2;
```

Example in FBD:

MOVE

IEC Operator: Assignment of a variable to another variable of an appropriate type.

As MOVE is available as a box in the graphic editors FBD, LD, CFC, there the (unlocking) EN/ENO functionality can also be applied on a variable assignment.

Example in CFC in conjunction with the EN/ENO function:

Only if en_i is TRUE, var1 will be assigned to var2.

Beispiel in AWL: (Result: var2 gets value of var1)

```
LD      var1
MOVE
ST      var2
```

(! you get the same result with:

```
LD      var1
ST      var2
```

)

Beispiel in ST:

```
ivar2 := MOVE(ivar1);
```

(! you get the same result with: ivar2 := ivar1;)

SIZEOF

This arithmetic operator is not prescribed by the standard IEC 61131-3.

It can be used to determine the number of bytes required by the given variable x.

The SIZEOF operator returns an unsigned value. The type of the return value will be adapted to the found size of variable x.

return value of SIZEOF(x)	data type of the constant implicitly used for the found size
0 <= size of x < 256	USINT
256 <= size of x < 65536	UINT
65536 <= size of x < 4294967296	UDINT
4294967296 <= size of x	ULINT

Example in ST:

```
var1 := SIZEOF(arr1);  (* d.h.:  var1:=USINT#10; *)
```

Example in IL: (Result is 10)

```
arr1:ARRAY[0..4] OF INT;
Var1:INT;
LD      arr1
SIZEOF
ST      Var1
```

INDEXOF

This arithmetic operator is not prescribed by the standard IEC 61131-3.

Perform this function to find the internal index for a POU.

Example in ST:

```
var1 := INDEXOF(POU2);
```

16.3.3 Bitstring Operators

The following bitstring operators are available, matching the IEC1131-3 standard:
AND, OR, XOR, NOT

not yet implemented: The following operators are not prescribed by the standard:
ANDN, ORN, XORN

Bitstring operators compare the corresponding bits of two or several operands.

AND

IEC Bitstring Operator: Bitwise AND of bit operands. If the input bits each are 1, then the resulting bit will be 1, otherwise 0.

Allowed types: BOOL, BYTE, WORD, DWORD, LWORD.

Example in IL: (Result in Var1 is 2#1000_0010)

```
Var1:BYTE;
LD      2#1001_0011
AND     2#1000_1010
ST      var1
```

Example in ST:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

Example in FBD:

ANDN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

OR

IEC Bitstring Operator: Bitwise OR of bit operands. If at least one of the input bits is 1, the resulting bit will be 1, otherwise 0.

Allowed types: BOOL, BYTE, WORD or DWORD, LWORD.

Example in IL: (Result in var1 is 2#1001_1011)

```
var1 :BYTE;
LD      2#1001_0011
OR      2#1000_1010
ST      Varl
```

Example in ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

Example in FBD:

ORN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

XOR

IEC Bitstring Operator: Bitwise XOR operation of bit operands. If only one of the input bits is 1, then the resulting bit will be 1; if both or none are 1, the resulting bit will be 0.

Allowed types: BOOL, BYTE, WORD, DWORD, LWORD.

Note: Notice the behaviour of the XOR function in extended form, that means if there are more than 2 inputs. The inputs will be checked in pairs and the particular results will then be compared again in pairs (this complies with the standard, but may not be expected by the user).

Example in IL: (Result is 2#0001_1001)

```
Var1 :BYTE;
LD      2#1001_0011
XOR     2#1000_1010
ST      var1
```

Example in ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

Example in FBD:

XORN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

NOT

IEC Bitstring Operator: Bitwise NOT operation of a bit operand. The resulting bit will be 1, if the corresponding input bit is 0 and vice versa.

Allowed types: BOOL, BYTE, WORD, DWORD, LWORD.

Example in IL: (Result in Var1 is 2#0110_1100)

```
Var1 :BYTE;
LD      2#1001_0011
NOT
ST      var1
```

Example in ST:

```
Var1 := NOT 2#1001_0011
```

Example in FBD:

16.3.4 Bit-Shift Operators

The following bit-shift operators, matching the IEC1131-e standard, are available:
SHL, SHR, ROL, ROR

SHL

IEC Operator: Bitwise left-shift of an operand.

```
erg:= SHL (in, n)
```

in: Operand to be shifted to the left.

n: Number of bits, by which *in* gets shifted to the left.

If *n* exceeds the data type width, BYTE, WORD, DWORD and LWORD operands will be filled with zeros, operands of signed data types, like for example INT, will get an arithmetic shift, that means they will be filled with the value of the topmost bit.

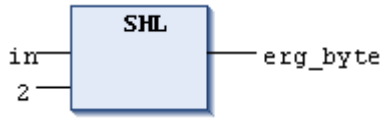
Note: Please note, that the amount of bits, which is noticed for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is noticed. The data type of the output variable has no effect at all on the arithmetic operation.

Note: See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:

```
PROGRAM shl_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:=SHL(in_byte,n); (* Result is 16#14 *)
erg_word:=SHL(in_word,n); (* Result is 16#0114 *)
```

Example in FBD:



Example in IL:

```
LD    in_byte
SHL   2
ST    erg_byte
```

SHR

IEC Operator: Bitwise right-shift of an operand.

```
erg:= SHR (in, n)
```

in: Operand to be shifted to the right.

n: Number of bits, by which *in* gets shifted to the right.

If *n* exceeds the data type width, operands of type BYTE, WORD, DWORD and LWORD will be filled with zeros, operands of signed data types, like for example INT, will get an arithmetic shift, that means they will be filled with the value of the topmost bit.

Note: Please note, that the amount of bits, which is noticed for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is noticed. The data type of the output variable has no effect at all on the arithmetic operation.

See the following example in hexadecimal notation to notice the results of the arithmetic operation depending on the type of the input variable (BYTE or WORD).

Example in ST:

```
PROGRAM shr_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:=SHR(in_byte,n); (* Result is 11 *)
erg_word:=SHR(in_word,n); (* Result is 0011 *)
```

Example in FBD:

Example in IL:

```
LD    in_byte
SHR   2
ST    erg_byte
```

ROL

IEC Operator: Bitwise rotation of an operand to the left.

```
erg:= ROL (in, n)
```

Allowed data types: BYTE, WORD, DWORD, LWORD.

in will be shifted one bit position to the left *n* times while the bit that is furthest to the left will be reinserted from the right.

Note: Please note, that the amount of bits, which is noticed for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is noticed. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in 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 is 16#15 *)
erg_word:=ROL(in_word,n); (* Result is 16#0114 *)
```

Example in FBD:

Example in IL:

```
LD      in_byte
ROL     n
ST      erg_byte
```

ROR

IEC Operator: Bitwise rotation of an operand to the right.

```
erg = ROR (in, n)
```

Allowed data types: BYTE, WORD, DWORD, LWORD.

in will be shifted one bit position to the right *n* times while the bit that is furthest to the left will be reinserted from the left.

Note: Please note, that the amount of bits, which is noticed for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is noticed. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in 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 is 16#51 *)
erg_word:=ROR(in_word,n); (* Result is16#4011 *)
```

Example in FBD:

Example in IL:

```
LD      in_byte
ROR     n
ST      erg_byte
```

16.3.5 Selection Operators

All selection operations can also be performed with variables. For purposes of clarity we will limit our examples to the following which use constants as operators: SEL, MAX, MIN, LIMIT, MUX

SEL

IEC Selection Operator: Binary Selection. *G* determines whether *IN0* or *IN1* is assigned to OUT.

OUT := SEL(G, IN0, IN1) means:

OUT := IN0; if G=FALSE

OUT := IN1; if G=TRUE.

Allowed data types:

IN0, IN1, OUT: any type

G: BOOL.

Example in IL:

```
LD  TRUE
SEL 3,4   (* IN0 = 3, IN1 =4 *)
ST  Var1  (* Result is 4 *)
LD  FALSE
SEL 3,4
ST  Var1  (* Result is 3 *)
```

Example in ST:

```
Var1:=SEL(TRUE,3,4); (* Result is 4 *)
```

Example in FBD:

Note: Note that an expression occurring ahead of IN1 or IN2 will not be processed if IN0 is TRUE.

MAX

IEC Selection Operator: Maximum function. Returns the greater of the two values.

```
OUT := MAX(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

Example in IL: (Result is 90)

```
LD   90
MAX  30
MAX  40
MAX  77
ST   Var1
```

Example in ST:

```
Var1:=MAX(30,40); (* Result is 40 *)
Var1:=MAX(40,MAX(90,30)); (* Result is 90 *)
```

Example in FBD:

MIN

IEC Selection Operator: Minimum function. Returns the lesser of the two values.

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

Example in IL: (Result is 30)

```
LD   90
MIN  30
MIN  40
MIN  77
ST   Var1
```

Example in ST:

```
Var1:=MIN(90,30); (* Result is 30 *);
Var1:=MIN(MIN(90,30),40); (* Result is 30 *);
```

Example in FBD:

LIMIT

IEC Selection Operator: Limiting

```
OUT := LIMIT(Min, IN, Max) means:
OUT := MIN (MAX (IN, Min), Max)
```

Max is the upper and Min the lower limit for the result. Should the value IN exceed the upper limit Max, LIMIT will return Max. Should IN fall below Min, the result will be Min.

IN and OUT can be any type of variable.

Example in IL: (Result is 80)

```
LD      90
LIMIT   30 ,
        80
ST      Var1
```

Example in ST:

```
Var1:=LIMIT(30,90,80); (* Result is 80 *);
```

MUX

IEC Selection Operator: Operator: Multiplexer

OUT := MUX(K, IN0,...,INn) means:
OUT := INK.

IN0, ...,INn and OUT can be any type of variable. K must be BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT or UDINT. MUX selects the Kth value from among a group of values.

Example in IL: (Result is 30)

```
LD      0
MUX     30 ,
        40 ,
        50 ,
        60 ,
        70 ,
        80
ST      Var1
```

Example in ST:

```
Var1:=MUX(0,30,40,50,60,70,80); (* Result is 30 *);
```

Note: Note that an expression occurring ahead of an input other than INK will not be processed to save run time ! Only in simulation mode all expressions will be executed.

16.3.6 Comparison Operators

The following operators, matching the IEC1131-3 standard are available:

GT, LT, LE, GE, EQ, NE

These are boolean operators, each comparing two inputs (first and second operand).

GT

EC Comparison Operator: Greater than

A Boolean operator which returns the value TRUE when the value of the first operand is greater than that of the second. The operands can be of any basic data type.

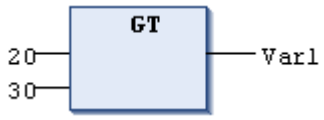
Example in AWL: (Result is FALSE)

```
LD    20
GT    30
ST    Var1
```

Example in ST:

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
```

Example in FBD:



LT

EC Comparison Operator: Less than

A Boolean operator that returns the value TRUE when the value of the first operand is less than that of the second. The operands can be of any basic data type.

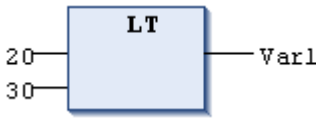
Example in IL: (Result is TRUE)

```
LD    20
LT    30
ST    Var1
```

Example in ST:

```
VAR1 := 20 < 30;
```

Example in FBD:



LE

EC Comparison Operator: Less than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is less than or equal to that of the second. The operands can be of any basic data type.

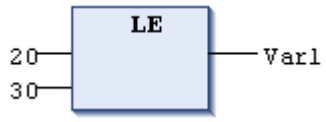
Example in ILL: (Result is TRUE)

```
LD    20
LE    30
ST    Var1
```

Example in ST:

```
VAR1 := 20 <= 30;
```

Example in FBD:



GE

EC Comparison Operator: Greater than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is greater than or equal to that of the second. The operands can be of any basic data type.

```
LD    60
GE    40
ST    Var1
```

Example in ST:

```
VAR1 := 60 >= 40;
```

Example in FBD:

EQ

EC Comparison Operator: Equal to

A Boolean operator that returns the value TRUE when the operands are equal. The operands can be of any basic data type.

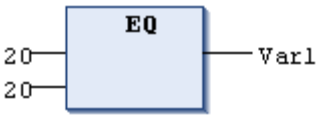
Example in IL: (Result is TRUE)

```
LD    40
EQ    40
ST    Var1
```

Example in ST:

```
VAR1 := 40 = 40;
```

Example in FBD:



NE

EC Comparison Operator: Not equal to

A Boolean operator that returns that value TRUE when the operands are not equal. The operands can be of any basic data type.

Example in IL: (Result is FALSE)

```
LD    40
NE    40
ST    Var1
```

Example in ST:

```
VAR1 := 40 <> 40;
```

Example in FBD:

16.3.7 Address Operators

ADR and BITADR and the content operator "^" are norm-extending address operators available in »PLC Designer«.

ADR

This address operator is not prescribed by the standard IEC 61131-3.

ADR returns the address of its argument in a DWORD. This address can be sent to manufacturing functions to be treated as a pointer or it can be assigned to a pointer within the project.

Note: In contrast to »PLC Designer« V2.3 now the ADR-Operator can be used with function names, program names, function block names and method names, thus replacing the INDEXOF operator. See in this context 'Function pointers'. Notice anyway that function pointers can be passed to external libraries, but there is no possibility to call a function pointer within »PLC Designer«! In order to enable a system call (runtime system) the respective property (category "Build") must be set for the function object.

Example in ST:

```
dwVar:=ADR(bVAR);
```

Example in IL:

```
LD      bVar
ADR
ST      dwVar
```

Note: After an Online Change there might be changes concerning the data on some addresses. Notice this in this case of using pointers on addresses.

BITADR

This address operator is not prescribed by the standard IEC 61131-3.

BITADR returns the bit offset within the segment in a DWORD. Notice that the offset value depends on whether the option byte addressing in the target settings is activated or not.

```
VAR
 var1 AT %IX2.3:BOOL;
 bitoffset: DWORD;
END_VAR
```

Example in ST:

```
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 19, if byte
addressing=FALSE: 35 *)
```

Example in IL:

```
LD          Var1
BITADR
ST          bitoffset
```

Note: After an Online Change there might be changes concerning the data on some addresses. Please notice this in this case of using pointers on addresses.

Content Operator

This address operator is not prescribed by the standard IEC 61131-3.IEC. A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example in ST:

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

Note: After an Online Change there might be changes concerning the data on some addresses. Please notice this in this case of using pointers on addresses.

16.3.8 Calling Operator

CAL

EC Operator: Calling a function block or a program

Use CAL **in IL** to call up a function block instance. The variables that will serve as the input variables are placed in parentheses right after the name of the function block instance.

Example:

Calling up the instance *Inst* from a function block where input variables *Par1* and *Par2* are 0 and TRUE respectively.

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

16.3.9 Type Conversion Functions

Its is forbidden to implicitly convert from a "larger" type to a "smaller" type (for example from INT to BYTE or from DINT to WORD). Special type conversions are required if one wants to do this. One can basically convert from any elementary type to any other elementary type.

Syntax:

<elem.Typ1>_TO_<elem.Typ2>

Please notice that at ...TO_STRING conversions the string is generated left-justified. If it is defined to short, it will be cut from the right side.

The following type conversions are supported:

- BOOL_TO Conversions
- TO_BOOL Conversions
- Conversion between Integral Number Types
- REAL_TO-/ LREAL_TO Conversions
- TIME_TO/TIME_OF_DAY Conversions
- DATE_TO/DT_TO Conversions
- STRING_TO Conversions
- new in comparison to V2.3:
- TRUNC (conversion to DINT)
- TRUNC_INT
- ANY_NUM_TO_<numeric datatype>
- ANY_TO_<any datatype>

BOOL_TO Conversions

IEC Operator: Converting from type BOOL to any other type.

Syntax for a BOOL_TO conversion operator:

BOOL_TO_<data type>

For number types the result is 1, when the operand is TRUE, and 0, when the operand is FALSE.

For the STRING type the result is ‚TRUE', when the operand is TRUE or ‚FALSE' when the operand is FALSE.

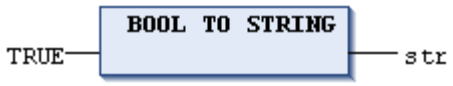
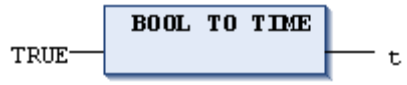
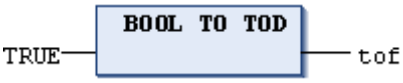
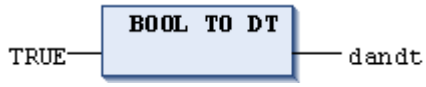
Examples in IL:

LD TRUE BOOL_TO_INT ST i	(* Result is 1 *)
LD TRUE BOOL_TO_STRI... ST str	(* Result is 'TRUE' *)
LD TRUE BOOL_TO_TIME ST t	(* Result is T#1ms *)
LD TRUE BOOL_TO_TOD ST tof	(* Result is TOD#00:00:00.001 *)
LD FALSE BOOL_TO_DATE ST dandt	(* Result is D#1970-01-01 *)
LD TRUE BOOL_TO_DT ST dandt	(* Result is DT#1970-01-01-00:00:01 *)

Examples in ST:

i:=BOOL_TO_INT(TRUE);	(* Result is 1 *)
str:=BOOL_TO_STRING(TRUE);	(* Result is "TRUE" *)
t:=BOOL_TO_TIME(TRUE);	(* Result is T#1ms *)
tof:=BOOL_TO_TOD(TRUE);	(* Result is TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);	(* Result is D#1970 *)
dandt:=BOOL_TO_DT(TRUE);	(* Result is DT#1970-01-01-00:00:01 *)

Examples in FBD:

	(* Result is 1 *)
	(* Result is 'TRUE' *)
	(* Result is T#1ms *)
	(* Result is TOD#00:00:00.001 *)
	(* Result is D#1970-01-01 *)
	(* Result is DT#1970-01-01-00:00:01 *)

TO_BOOL Conversions

IEC Operator: Converting from another variable type to BOOL

Syntax for a TO_BOOL conversion operator:

<data type>_TO_BOOL

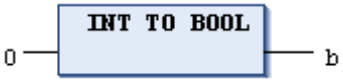
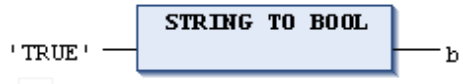
The result is TRUE when the operand is not equal to 0. The result is FALSE when the operand is equal to 0.

The result is TRUE for STRING type variables when the operand is "TRUE", otherwise the result is FALSE.

Examples in IL:

LD 213 BYTE_TO_BOOL ST b	(* Result is TRUE *)
LD 0 INT_TO_BOOL ST b	(* Result is FALSE *)
LD T#5ms TIME_TO_BOOL ST b	(* Result is TRUE *)
LD 'TRUE' STRING_TO_BOOL ST b	(* Result is TRUE *)

Examples in FBD:

	(*Result is TRUE *)
	(* Result is FALSE *)
	(* Result is TRUE *)
	(* Result is TRUE *)

Examples in St:

b := BYTE_TO_BOOL(2#11010101);	(* Result is TRUE *)
b := INT_TO_BOOL(0);	(* Result is FALSE *)
b := TIME_TO_BOOL(T#5ms);	(* Result is TRUE *)
b := STRING_TO_BOOL('TRUE');	(* Result is TRUE *)

Conversion between Integral Number Types

Conversion from an integral number type to another number type.

Syntax for the conversion operator:

<INT data type>_TO_<INT data type>

When you perform a type conversion from a larger to a smaller type, you risk losing some information. If the number you are converting exceeds the range limit, the first bytes for the number will be ignored.

Beispiel in ST:

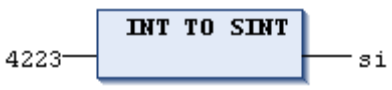
```
si := INT_TO_SINT(4223); (* Result is 127 *)
```

If you save the integer 4223 (16#107f represented hexadecimally) as a SINT variable, it will appear as 127 (16#7f represented hexadecimally).

Beispiel in AWL:

```
LD          4223
INT_TO_SINT
ST          si
```

Beispiel in FUP:



REAL_TO-/ LREAL_TO Conversions

IEC Operator: Converting from the variable type REAL or LREAL to a different type:

The value will be rounded up or down to the nearest whole number and converted into the new variable type. Exceptions to this are the variable types STRING, BOOL, REAL and LREAL.

Note: If a REAL or LREAL is converted to SINT, USINT, INT, UINT, DINT, UDINT, LINT or ULINT and the value of the real number is out of the value range of that integer, the result will be undefined and will depend on the target system. Even an exception is possible in this case! In order to get target-independant code, handle any range exceedance by the application. If the real/lreal number is within the integrer value range, the conversion will work on all systems in the same way.

Notice at a conversion to type STRING that the total number of digits is limited to 16. If the (L)REAL-number has more digits, then the sixteenth will be rounded. If the length of the STRING is defined to short, it will be cut beginning from the right end.

Note: When you perform a type conversion from a larger to a smaller type, you risk losing some information.

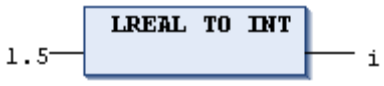
Example in ST:

```
i := REAL_TO_INT(1.5); (* Result is 2 *)
j := REAL_TO_INT(1.4); (* Result is 1 *)
i := REAL_TO_INT(-1.5); (* Result is -2 *)
j := REAL_TO_INT(-1.4); (* Result is -1 *)
```

Example in IL:

```
LD          2.7
REAL_TO_INT
ST          i
```

Example in FBD:



TIME_TO/TIME_OF_DAY Conversions

IEC Operator: Converting from the variable type TIME or TIME_OF_DAY to a different type.

Syntax for the conversion operator:

<time data type>_TO_<data type>

The time will be stored internally in a DWORD in milliseconds (beginning with 12:00 A.M. for the TIME_OF_DAY variable). This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For the STRING type variable, the result is a time constant.

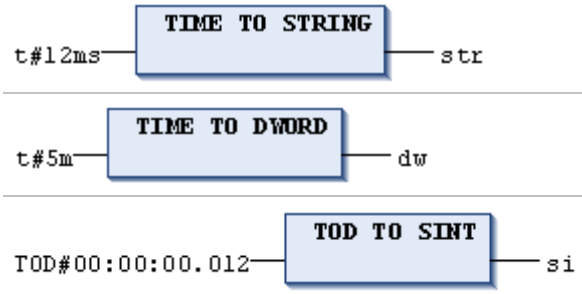
Examples in IL:

LD T#12ms TIME_TO_STTI... ST str	(*Result is 'T#12ms' *)
LD T#300000ms TIME_TO_DWORD ST dw	(*Result is 300000 *)
LD TOD#00:00:00.012 TIME TO SINT ST si	(*Result is 12 *)

Examples in ST:

str :=TIME_TO_STRING(T#12ms);	(* Result is T#12ms *)
dw:=TIME_TO_DWORD(T#5m);	(* Result is 300000 *)
si:=TOD_TO_SINT(TOD#00:00:00.012);	(* Result is 12 *)

Examples in FBD:



DATE_TO/DT_TO Conversions

IEC Operator: Converting from the variable type DATE or DATE_AND_TIME to a different type.

Syntax for the conversion operator:

<date data type>_TO_<data type>

The date will be stored internally in a DWORD in seconds since Jan. 1, 1970. This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For STRING type variables, the result is the date constant.

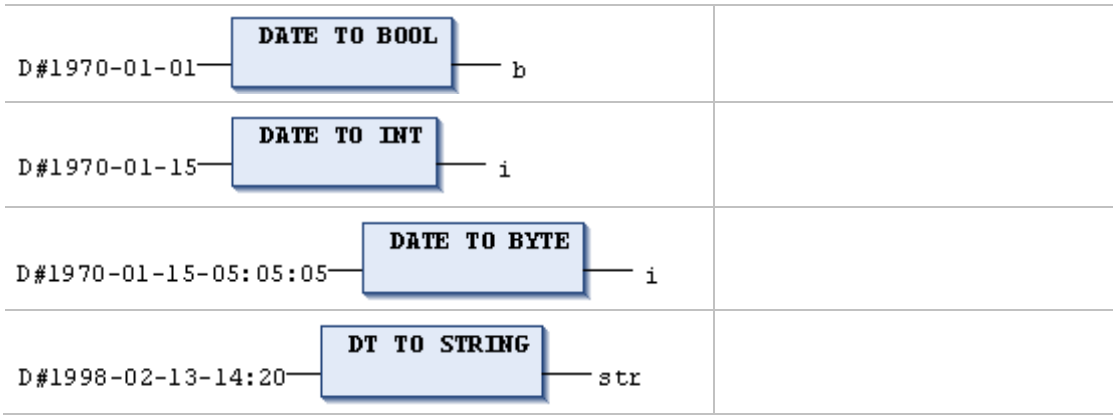
Examples in IL:

LD D#1970-01-01 DATE_TO_BOOL ST b	(* Result is FALSE *)
LD D#1970-01-01 DATE_TO_INT ST i	(* Result is 29952 *)
LD D#1970-01-15-05:05:. DATE_TO_BYTE ST byt	(* Result is 129 *)
LD D#1998-02-13-14:20 DATE_TO_STRI... ST str	(* Result is 'DT#1998-02-13-14:20' *)

Examples in ST:

b :=DATE_TO_BOOL(D#1970-01-01);	(* Result is FALSE *)
i :=DATE_TO_INT(D#1970-01-15);	(* Result is 29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05);	(* Result is 129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20);	(* Result is 'DT#1998-02-13-14:20' *)

Examples in FBD:



STRING_TO Conversions

IEC Operator: Converting from the variable type STRING to a different type.

The conversion works according to the standard C compilation mechanism: STRING to INT and then INT to BYTE. The higher byte will be cut, thus only values of 0-255 result.

This way it is possible on most processors to generate optimal code because the conversion can be performed by a single machine instruction.

Syntax for the conversion operator:

STRING_TO_<data type>

The operand from the STRING type variable must contain a value that is valid in the target variable type, otherwise the result will be 0.

Examples in IL:

`LD 'TRUE'`
`STRING_TO_BOOL`
`ST b`
	(* Result is TRUE *)

`LD 'abc34'`
`STRING_TO_WORD`
`ST w`
	(* Result is 0 *)

`LD 't#117ms'`
`STRING_TO_TIME`
`ST t`
	(* Result is T#117ms *)

`LD '500'`
`STRING_TO_BYTE`
`ST bv`
	(* Result is 244 *)

Examples in ST:

b :=STRING_TO_BOOL('TRUE');	(* Result is TRUE *)
w :=STRING_TO_WORD('abc34');	(* Result is 0 *)
t :=STRING_TO_TIME('T#127ms');	(* Result is T#127ms *)
bv :=STRING:TO_BYTE('500');	(* Result is 244 *)

Examples in FBD:

	(* Result is TRUE *)
	(* Result is 0 *)
	(* Result is T#127ms *)
	(* Result is 244 *)

TRUNC

IEC Operator: Converting from REAL to DINT. The whole number portion of the value will be used.

Note: In »PLC Designer« V2.3 the TRUNC operator converts from REAL to INT. For this reason when importing a V2.3 project TRUNC automatically will be replaced by TRUNC_INT.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Example in IL:

```
LD      1.9
TRUNC
ST      diVar
```

Examples in ST:

```
diVar:=TRUNC(1.9); (* Result is 1 *)
diVar:=TRUNC(-1.4); (* Result is -1 *).
```

TRUNC_INT

IEC Operator: Converting from REAL to INT. The whole number portion of the value will be used.

Note: TRUNC_INT corresponds to operator TRUNC in »PLC Designer« V2.3 and in case of a V2.3 import will automatically replace that one. Notice the modified conversion function of TRUNC in V3.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Example in IL:

```
LD      1.9
TRUNC_INT
ST      iVar
```

Examples in ST:

```
iVar:=TRUNC_INT(1.9); (* Result is 1 *)
iVar:=TRUNC_INT(-1.4); (* Result is -1 *).
```

ANY_..._TO Conversions

IEC Operator: Converting from any data type, resp. more specifically from any numeric data type to another data type. A reasonable choice of data types is assumed.

Syntax:

ANY_NUM_TO_<numeric data type>

ANY_TO_<any data type>

Example:

Conversion from a variable of data type REAL to INT:

```
re : REAL := 1.234;
i : INT := ANY_TO_INT(re)
```

16.3.10 Numeric Functions

The following numeric IEC operators are available:
ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN, EXPT

ABS

IEC Operator: Returns the absolute value of a number. ABS(-2) returns 2.
In- and output can be of any numeric basic data type.

Example in IL: (Result in i is 2)

```
LD    -2
ABS
ST    i
```

Example in ST:

```
i:=ABS(-2);
```

Example in FBD:

SQRT

IEC Operator: Returns the square root of a number.
The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 4)

```
LD    16
SQRT
ST    q
```

Example in ST:

```
q:=SQRT(16);
```

Example in FBD:

LN

IEC Operator: Returns the natural logarithm of a number.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 3.80666)

```
LD    45
LN
ST    q
```

Example in ST:

```
q:=LN(45);
```

Example in FBD:

LOG

IEC Operator: Returns the logarithm of a number in base 10.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 2.49762 *)

```
LD    314.5
LOG
ST    q
```

Example in ST:

```
q:=LOG(314.5);
```

Example in FBD:

EXP

IEC Operator: Returns the exponential function.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 7.389056099)

```
LD    2
EXP
ST    q
```

Example in ST:

```
q:=EXP(2);
```

Example in FBD:

SIN

IEC Operator: Returns the sine of a number.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

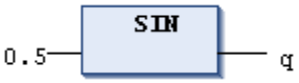
Example in IL: (Result in q is 0.479426 *)

```
LD    0.5
SIN
ST    q
```

Example in ST:

```
q:=SIN(0.5);
```

Example in FBD:



COS

IEC Operator: Returns the cosine of number. The value is calculated in arch minutes.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

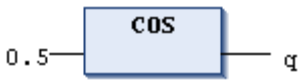
Example in IL: (* Result in q is 0.877583 *)

```
LD      0.5
COS
ST      q
```

Example in ST:

```
q:=COS(0.5);
```

Example in FBD:



TAN

IEC Operator: Returns the tangent of a number. The value is calculated in arch minutes.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 0.546302 *)

```
LD      0.5
TAN
ST      q
```

Example in ST:

```
q:=TAN(0.5);
```

Example in FBD:

ASIN

IEC Operator: Returns the arc sine (inverse function of sine) of a number. .

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 0.523599 *)

```
LD      0.5
ASIN
ST      q
```

Example in ST:

```
q:=ASIN(0.5);
```

Example in FBD:

ACOS

IEC Operator: Returns the arc cosine (inverse function of cosine) of a number. The value is calculated in arch minutes.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

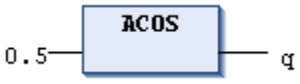
Example in IL: (Result in q is 1.0472)

```
LD      0.5
ACOS
ST      q
```

Example in ST:

```
q:=ACOS(0.5);
```

Example in FBD:



ATAN

IEC Operator: Returns the arc tangent (inverse function of tangent) of a number. The value is calculated in arch minutes.

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

Example in IL: (Result in q is 0.463648)

```
LD      0.5
ATAN
ST      q
```

Example in ST:

```
q:=ATAN(0.5);
```

Example in FBD:

EXPT

IEC Operator: Exponentiation of a variable with another variable:

```
OUT = IN1IN2.
```

The input variable can be of any numeric basic data type, the output variable must be type REAL or LREAL.

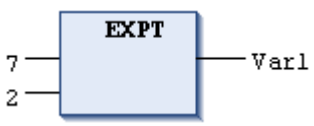
Example in IL: (Result is 49)

```
LD      7
EXPT    2
ST      Var1
```

Example in ST:

```
var1 := EXPT(7,2);
```

Example in FBD:



16.3.11 IEC extending Operators

Additionally to the IEC operators »PLC Designer« supports following operators:

- ADR
- BITADR
- INDEXOF
- SIZEOF
- ANDN
- ORN
- XORN
- __DELETE
- __ISVALIDREF
- __NEW
- __QUERYINTERFACE
- __QUERYPOINTER

__DELETE

This operator is not prescribed by the IEC 61131-3 standard.

Note: For compatibility reasons the compiler version must be >= 3.3.2.0.

The operator __DELETE deallocates the memory of objects who has allocated memory with the operator __NEW before. __DELETE has no return value and its operand will be set to 0 after the operation.

For allocating use __NEW.

Syntax:

__DELETE(<pointer>)

If pointer is a pointer to a function block the dedicated method FB_Exit will be called before the pointer is set to NULL.

Example with a function block:

```
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;

METHOD FB_Exit : BOOL
VAR_INPUT
bInCopyCode : BOOL;
END_VAR
__Delete(_dut);

METHOD FB_Init : BOOL
VAR_INPUT
bInitRetains : BOOL;
bInCopyCode : BOOL;
END_VAR
_dut := __NEW(DUT);

METHOD INC : INT
VAR_INPUT
END_VAR
_inc := _inc + 1;
INC := _inc;
```

```
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
```

__ISVALIDREF

This operator is not prescribed by the IEC 61131-3 standard. It can be used to check whether a reference points to a valid value. For how to use and an example see the description of the reference data type.

__NEW

This operator is not prescribed by the IEC 61131-3 standard.

Note: For compatibility reasons the compiler version must be >= 3.3.2.0.

The operator __NEW allocates memory for function block instances or arrays of standard data types. The operator returns a suitably typed, nonzero pointer to the object. If the operator is not used within an assignment an error message will occur.

If unsuccessful, by __NEW returns 0.

For deallocating use __DELETE.

Syntax:

__NEW (<type>, [<size>])

The operator will create a new object of the specified type <type> and returns a pointer to that <type>. The initialization of the object is called after creation. If 0 is returned the operation is failed.

If <type> is scalar the optional operand <length> has to be set additionally and the operator creates an array of scalar types with the size length.

Example:

```
pScalarType := __New(ScalarType, length);
```

Note: A copy code in online change of dynamically created objects is not possible!

Therefore only function blocks out of libraries (because they can't change) and function blocks with attribute 'enable_dynamic_creation' are allowed for the __New-Operator.
If a function block change with this flag so that copy code would be necessary, an error message is produced.

Note: The code for memory allocation needs to be non-reentrant.

A semaphor (SysSemEnter) is used to avoid two tasks try to allocate memory at the same time. Thus, extensive usage of __New may produce higher jitter.

Example with a scalar type:

```
TYPE DUT :
STRUCT
a,b,c,d,e,f : INT;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
pDut : POINTER TO DUT;
bInit: BOOL := TRUE;
bDelete: BOOL;
END_VAR
```

```
IF (bInit) THEN
pDut := __NEW(DUT);
bInit := FALSE;
END_IF
IF (bDelete) THEN
__DELETE(pDut);
END_IF
```

Example with a function block:

```
FBDynamic(FP)
{attribute 'enable_dynamic_creation'}
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;

PLC_PRG(PRG)
VAR
pFB : POINTER TO FBDynamic;
loc : INT;
bInit: BOOL := TRUE;
bDelete: BOOL;
END_VAR
IF (pFB <> 0) THEN
pFB^(in1 := 1, in2 := loc, out => loc);
pFB^.INC();
END_IF
IF (bDelete) THEN
__DELETE(pFB);
END_IF
```

Example with an array:

```
PLC_PRG(PRG)
VAR
bInit: BOOL := TRUE;
bDelete: BOOL;
pArrayBytes : POINTER TO BYTE;
pArrayDuts : POINTER TO BYTE;
test: INT;
parr : POINTER TO BYTE;
END_VAR
IF (bInit) THEN
pArrayBytes := __NEW(BYTE, 25);
bInit := FALSE;
END_IF
IF (pArrayBytes <> 0) THEN
pArrayBytes[24] := 125;
test := pArrayBytes[24];
END_IF
IF (bDelete) THEN
__DELETE(pArrayBytes);
END_IF
```

__QUERYINTERFACE

This operator is not prescribed by the IEC 61131-3 standard.

At runtime __QUERYINTERFACE is enabling a type conversion of an interface reference to another. The operator returns a result with type BOOL. TRUE implies, that the conversion is successfully executed.

Note: For compatibility reasons the definition of the reference to be converted must be an extension of the base interface __SYSTEM.IQueryInterface and the compiler version must be >= 3.3.0.20.

Syntax:

__QUERYINTERFACE(<ITF_Source>, < ITF_Dest>)

The operator needs as first operand an interface reference or a function block instance of the intended type and as second operand an interface reference. After execution of __QUERYINTERFACE the ITF_Dest holds a reference to the intended interface if the object referenced from ITF Source implements the interface. In this case the conversion is successfully and the result of the operator returns TRUE. In all other cases the operator returns FALSE.

Precondition for an explicit conversion is that not only the ITF_Source but also ITF Dest is an extension of the interface __System.IQueryInterface. This interface is provided implicitly and needs no library.

Example in ST:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived1 EXTENDS __System.IQueryInterface
METHOD mderived1 : BOOL
END_METHOD

INTERFACE ItfDerived2 EXTENDS __System.IQueryInterface
METHOD mderived2 : BOOL
END_METHOD

PROGRAMM POU
VAR
itfderived1 : ItfDerived1;
itfderived2 : ItfDerived2;
bTest1, bTest2, xResult1, xResult2: BOOL;
END_VAR

xResult1 := __QUERYINTERFACE(itfbase, itfderived1); // variables of type ItfBase
resp. ItfDerived1
xResult2 := __QUERYINTERFACE(itfbase, itfderived2); // variables of type ItfBase
resp. ItfDerived2
IF (xResult1 = TRUE) THEN
bTest1 := itfderived1.mderived1();
ELSIF xResult2 THEN
bTest2 := itfderived2.mderived2();
END_IF
```

__QUERYPOINTER

This operator is not prescribed by the IEC 61131-3 standard.

At runtime __QUERYPOINTER is assigning an interface reference to a untyped pointer. The operator returns a result with type BOOL. TRUE implies, that the conversion has been successfully executed.

Note: For compatibility reasons the definition of the intended interface reference must be an extension of the base interface __SYSTEM.IQueryInterface and the compiler version must be >= 3.3.0.20.

Syntax:

__QUERYPOINTER (<ITF_Source>, < Pointer_Dest>)

For the first operand the operator rquires a interface reference or a function block instance of the intended type and for the second operand a untyped pointer. After execution of __QUERYPOINTER the Pointer_Dest holds the address of the reference to the intended interface. In this case the conversion is successful and the result of the operator returns TRUE. In all other cases the operator returns FALSE. Pointer_Dest is untyped and can be cast to any type. The actual type has to be ensured by the programmer. For example the interface could provide a method returning a type code.

Precondition for an explicit conversion is that the ITF_Source is an extension of the interface __System.IQueryInterface. This interface is provided implicitly and needs no library.

Example:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD

FUNCTION_BLOCK FBVariante IMPLEMENTS ITFDerived

PROGRAMM POU
VAR
itfderived : ItfDerived;
insV : FBVariante;
xResult, xTest : BOOL;
pVar: POINTER TO DWORD;
END_VAR
itfderived := insV;
xResult := __QUERYPOINTER(itfderived, pVar);
IF xResult THEN
xTest := pVar.mderived();
END_IF
```

ADR

This address operator is not prescribed by the standard IEC 61131-3.

ADR returns the address of its argument in a DWORD. This address can be sent to manufacturing functions to be treated as a pointer or it can be assigned to a pointer within the project.

Note: In contrast to »PLC Designer« V2.3 now the ADR-Operator can be used with function names, program names, function block names and method names, thus replacing the INDEXOF operator. See in this context 'Function pointers'. Notice anyway that function pointers can be passed to external libraries, but there is no possibility to call a function pointer within »PLC Designer«! In order to enable a system call (runtime system) the respective property (category "Build") must be set for the function object.

Example in ST:

```
dwVar:=ADR(bVAR);
```

Example in IL:

```
LD      bVar
ADR
ST      dwVar
```

Note: After an Online Change there might be changes concerning the data on some addresses. Notice this in this case of using pointers on addresses.

BITADR

This address operator is not prescribed by the standard IEC 61131-3.

BITADR returns the bit offset within the segment in a DWORD. Notice that the offset value depends on whether the option byte addressing in the target settings is activated or not.

```
VAR
 var1 AT %IX2.3:BOOL;
 bitoffset: DWORD;
END_VAR
```

Example in ST:

```
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 19, if byte
addressing=FALSE: 35 *)
```

Example in IL:

```
LD      Var1
BITADR
ST      bitoffset
```

Note: After an Online Change there might be changes concerning the data on some addresses. Please notice this in this case of using pointers on addresses.

INDEXOF

This arithmetic operator is not prescribed by the standard IEC 61131-3.

Perform this function to find the internal index for a POU.

Example in ST:

```
var1 := INDEXOF(POU2);
```

SIZEOF

This arithmetic operator is not prescribed by the standard IEC 61131-3.

It can be used to determine the number of bytes required by the given variable x.

The SIZEOF operator returns an unsigned value. The type of the return value will be adapted to the found size of variable x.

return value of SIZEOF(x)	data type of the constant implicitly used for the found size
0 <= size of x < 256	USINT
256 <= size of x < 65536	UINT
65536 <= size of x < 4294967296	UDINT
4294967296 <= size of x	ULINT

Example in ST:

```
var1 := SIZEOF(arr1);  (* d.h.:  var1:=USINT#10; *)
```

Example in IL: (Result is 10)

```
arr1:ARRAY[0..4] OF INT;
Var1:INT;
LD      arr1
SIZEOF
ST      Var1
```

ANDN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

ORN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

XORN

This IEC Bitstring Operator is not prescribed by the standard IEC 61131-3.

not yet implemented

Scope Operators

In extension to the IEC operators there are several possibilities to disambiguate the access to variables or modules if the variables or module name is used multiple times within the scope of a project. To define the respective namespace the following scope operators can be used:

"." (global scope operator)

"<global variable list name>."

"<library name>."

"<enumeration name>."

Global scope Operator

Scope operator: Extension to the IEC 61131-3 standard.

An instance path starting with "**.**" opens a global scope (namespace). So, if there is a local variable with the same name "<varname>" as a global variable ".<varname>" will refer to the global variable.

Global variable list name

Scope operator: Extension to the IEC 61131-3 standard.

The name of a global variable list can be used as a namespace for the variables enclosed in this list. Thus it is possible to declare variables with identical names in different global variable lists and by preceding the variable name by "**<global variable list name>.**" it is possible to access the desired one.

Example:

The global variables lists "globlist1" and "globlist2" each contain a variable named "varx". In the following line" varx" out of "globlist2" is copied to "varx" in "globlist1":

```
globlist1.varx := globlist2.varx; (* in this codeline *)
```

If a variable name declared in more than one global variable lists is referenced without the global variable list name as a preceding operator, an error message will be output.

Library namespace

Scope operator: Extension to the IEC 61131-3 standard.

The library namespace can be used to explicitly access the library components. Example: If a library which is included in a project, contains a module "fun1" and there is also a POU "fun1" defined locally in the project, then add the "namespace" of the library can be added to the module-name in order to make the access unique. The syntax is: <namespace>.<module name>, for example "lib1.fun1".

By default the "namespace" of a library is identic with the library name, however you can define another one either in the Project Information when creating a library project, or later in the Properties dialog for a library.

Example:

There is a function "fun1" in library "lib". There is also a function "fun1" declared in the project. By default the namespace of library lib is named "lib":

```
res1 := fun(in := 12); // call of the project function fun
res2 := lib.fun(in := 12); // call of the library function fun
```

Enumeration name

Scope operator: Extension to the IEC 61131-3 standard.

The type name of enumerations can be used to disambiguate the access to an enumeration constant. In this case "**<enumeration name>.**" precedes the constant name. So it is possible to use the same constant in different enumerations.

Example:

The constant "Blue" is a component of enumeration "Colors" as well as of enumeration "Feelings".

```
color := Colors.Blue;  // Access to enum value Blue in type Colors
feeling := Feelings.Blue; // Access to enum value Blue in type Feelings
```

16.4 Operands

The following can be used as an operand:

- Constant (BOOL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME, number, REAL/LREAL, STRING, Typed Literals)
- Variable
- Address
- Function

16.4.1 Constants

BOOL Constants

BOOL constants are the logical values TRUE and FALSE.

See also: Data type BOOL

TIME Constants

TIME constants are generally used to operate the standard timer modules. Besides the time constant TIME, which is of size 32 Bit and matching the IEC 61131-3 standard, LTIME is supported as an extension to the standard as time base for high resolution timers. LTIME is of size 64 Bit and resolution nanoseconds.

Syntax for TIME constant:

t#<time declaration>

Instead of "t#" also the following can be used: "T#", "time", "TIME".

The time declaration can include the following time units. These must be used in the following sequence, but it is not required to use all of them.

"d": days

"h": hours

"m": minutes

"s": seconds

"ms": milliseconds

Examples of correct TIME constants in a ST assignment:

TIME1 := T#14ms;	
TIME1 := T#100S12ms;	(*The highest component may be allowed to exceed its limit*)
TIME1 := t#12h34m15s;	

the following would be incorrect:

TIME1 := t#5m68s;	(* limit exceeded in a lower component *)
TIME1 := 15ms;	(* T# is missing *)
TIME1 := t#4ms13d;	(* Incorrect order of entries *)

Syntax for LTIME constant:

LTIME#<time declaration>

The time declaration can include the time units as used with the TIME constant (see above) and additionally:

"us" : microseconds

"ns" : nanoseconds

Example:

Examples of correct LTIME constants in a ST assignment:

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns	
LTIME1 := LTIME#3445343m3424732874823ns	

see also: Time Data Types

DATE Constants

These constants can be used to enter dates.

Syntax:

dt#<date declaration>

Instead of "d#" also the following can be used: "D#", "date", "DATE".

The date declaration is to be entered in format "<year-month-day>".

DATE values are internally handled as DWORD values, containing the time span in seconds since 01.01.1970, 00:00 clock.

Examples:

```
DATE#1996-05-06
d#1972-03-29
```

see also: Time Data Types

TIME_OF_DAY Constants

Use this type of constant to store times of the day.

Syntax:

tod#<time declaration>

Instead of "tod#" also the following can be used: "TOD#", "time_of_day", "TIME_OF_DAY".

The time declaration is to be entered in format "<hour:minute:second>".

Seconds can be entered as real numbers, that is also fractions of a second can be specified.

TIME_OF_DAY values are internally handled as DWORD values, containing the time span in milliseconds since 00:00 clock.

Examples:

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

see also: Time Data Types

DATE_AND_TIME Constants

DATE constants and TIME_OF_DAY constants can also be combined to form so-called DATE_AND_TIME constants.

Syntax:

dt#<date and time declaration>

Instead of "dt#" also the following can be used: "DT#", "date_and_time", "DATE_AND_TIME".

The date and time declaration is to be entered in format "<year-month-day-hour:minute:second>".

Seconds can be entered as real numbers, that is also fractions of a second can be specified.

DATE_AND_TIME values are internally handled as DWORD values, containing the time span in seconds since 01.01.1970, 00:00 clock.

Examples:

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

see also: Time Data Types

Number Constants

Number values can appear as binary numbers, octal numbers, decimal numbers and hexadecimal numbers.

If an integer value is not a decimal number, the base must be followed by the number sign (#) in front of the integer constant.

The values for the numbers 10-15 in hexadecimal numbers will be represented by the letters A-F.

You may include the underscore character within the number.

Examples:

14	(decimal number)
2#1001_0011	(dual number)
8#67	(octal number)
16#A	(hexadecimal number)

These number values can be of type BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL or LREAL.

Implicit conversions from "larger" to "smaller" variable types are not permitted. This means that a DINT variable cannot simply be used as an INT variable. You must use the type conversion.

REAL/LREAL Constants

REAL and LREAL constants can be given as decimal fractions and represented exponentially. Use the standard American format with the decimal point to do this.

Example:

7.4	instead of 7,4
1.64e+009	instead of 1,64e+009

STRING Constants

A string is a sequence of characters.

STRING constants are preceded and followed by **single quotation marks**. You may also enter blank spaces and special characters (umlauts for instance). They will be treated just like all other characters.

Notice the following possibilities of using the dollar sign "**$**" in string constants:

entered combination	interpretation
$<two hex numbers>	hexadecimal representation of the eight bit character code
$$	Dollar sign
$'	Single quotation mark
$L or $l	Line feed
$N or $n	New line
$P or $p	Page feed
$R or $r	Line break
$T or $t	Tab

Examples:

'w1Wüß?'

' Abby and Craig '

':-)'

'costs ($$)'

Typed Literals

Basically in using IEC constants the smallest possible data type will be used. If another data type must be used, this can be achieved with the help of typed literals without the necessity of explicitly declaring the constants.

For this purpose the constant will be provided with a prefix which determines the type.

Syntax:

<Type>#<Literal>

<Type> specifies the desired data type; possible entries are: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. The type must be written in uppercase letters.

<Literal> specifies the constant. The data entered must fit within the data type specified under <Type>.

Example:

```
var1:=DINT#34;
```

If the constant can not be converted to the target type without data loss, an error message will be issued.

Typed literals can be used wherever normal constants can be used.

16.4.2 Variables

Variables can be declared either locally in the declaration part of a POU or in a Global Variable List.

See Variables Declaration for information on the declaration of a variable, including the rules concerning the variable identifier and multiple use.

Variables can be used anywhere the declared type allows for them.

You can access available variables through the Input Assistant.

Accessing variables for arrays, structures and POUs

Use the following syntax for accessing...

Two-dimensional array components:

<Fieldname>[Index1, Index2]

Structure variables:

<Structurename>.<Variablenname>

Function block and program variables:

<Functionblockname>.<Variablename>

Addressing bits in variables

In integer variables individual bits can be accessed. For this purpose the index of the bit to be addressed is appended to the variable, separated by a dot. The bit-index can be given by any constant. Indexing is 0-based.

Note: Bit accesses used in visualizations will only work over a Data Server connection if the bit index is defined literally (and not by a defined constant).

Syntax:

<variablename>.<bit index>

Example:

```
a : INT;
b : BOOL;
...
a.2 := b;
```

The third bit of the variable a will be set to the value of the variable b.

If the index is greater than the bit width of the variable, the following error message will be issued: "Index '<n>' outside the valid range for variable '<var>'!"

Bit addressing is possible with the following variable types: SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD.

If the variable type does not allow bit accessing, the following error message will be issued: "Invalid data type '<type>' for direct indexing".
A bit access must not be assigned to a VAR_IN_OUT variable!

Bitaccess via a global constant:
If you have declared a global constant defining the bit-index, you can use this constant for a bitaccess.

Examples for a bit access via a global constant:
1.on a variable and 2. on a structure variable:

Declaration in global variables list for both examples:
Variable *enable* defines which bit should be accessed:

```
VAR_GLOBAL CONSTANT
enable:int:=2;
END_VAR
```

Example 1, Bit access on an integer variable:
Declaration in POU:

```
VAR
xxx:int;
END_VAR
Bitaccess:
xxx.enable:=true;   (* -> the third bit in variable xxx will be set TRUE *)
```

Example 2, Bit access on an integer structure component:
Declaration of structure *stru1*:

```
TYPE stru1 :
STRUCT
bvar:BOOL;
rvar:REAL;
wvar:WORD;
{bitaccess enable 42 'Start drive'}
END_STRUCT
END_TYPE
```

Declaration in POU:

```
VAR
x:stru1;
END_VAR
```

Bitaccess:

```
x.wvar.enable:=true;
```

This will set TRUE the 42. bit in variable *x*. Since *bvar* has 8 bits and *rvar* has 32 bits, the bitaccess will be done on the second bit of variable *wvar*, which as a result will get value 4.

16.4.3 Addresses

Note: Online Change might change the contents on addresses. Please notice this when using pointers on addresses.

Memory location

You can use any supported size to access the memory location.

For example, the address %MD48 would address bytes numbers 192, 193, 194, and 195 in the memory location area (48 * 4 = 192). The number of the first byte is 0. The table below shows the corresponding memory location dependent on the size (X: bit, B: byte, W: word, D: dword) for IEC addressing.

Example:

%MX	96.0 - 96.7	96.8 - 192.15	97.0 - 97.7	97.8 - 97.15
%MB	192	193	194	195
%MW	96		97	
%MD	48			

You can access words, bytes and even bits in the same way: the address %MX96.0 allows you to access the first bit in the 96th word. (Bits are generally saved wordwise).

See Address for further information on addressing.

Note: Online Change might change the contents on addresses. Please notice this when using pointers on addresses.

Address

When specifying an address, the memory location and size are indicated by special character sequences.

Syntax:

%<memory area prefix><size prefix><number|.number|.number....>

The following memory area prefixes are supported:

I	Input (physical inputs via input driver, "sensors")
Q	Output (physical outputs via output driver, "actors")
M	Memory location

The following size prefixes are supported:

X	Single bit
None	Single bit
B	Byte (8 Bits)
W	Word (16 Bits)
D	Double word (32 Bits)

Examples:

%QX7.5 and %Q7.5	Output bit 7.5
%IW215	Input word 215
%QB7	Output byte 7
%MD48	Double word in memory position 48 in the memory location.
%IW2.5.7.1	Interpretation depends on the current PLC Configuration (see below)
ivar AT %IW0 : WORD;	Example of a variable declaration including an address assignment

Make sure to use a valid address:

For assigning a valid address within an application, first of all you must know the appropriate position within the process image, that is the **memory area** to be used: Input (I), Output (Q) or Memory (M) area, see above. Further on specify the desired **size**: bit, byte, word, dword (see above: X, B, W, D)

A decisive role plays the **current device configuration** and settings (hardware structure, device description, I/O settings). Especially consider the differences in address interpretation between devices using "byte addressing mode" or those using word oriented "IEC addressing mode".

So depending on the size and addressing mode different memory cells might be addressed by the same address definition.

See the table below for a comparison of **byte addressing** and **word oriented IEC addressing** for bits, bytes, words and dwords. After all it visualizes the **overlapping memory areas** in case of byte addressing mode (see an example following to the table.

Concerning the notation regard that for bit addresses the IEC addressing mode is always word oriented, that means that the place before the dot corresponds to the number of the word, the place behind names the number of the bit.

DWords / **W**ords				**B**ytes	**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

Abb. 542 Comparison of byte and word oriented addressing for the address sizes D,W, B and X:

n = number of the byte

Overlap of memory ranges in case of byte addressing mode, example:

D0 contains B0 - B3, W0 contains B0 and B1, W1 contains B1 and B2, W2 contains B2 and B3 -> In order to get around the overlap W1 resp. D1, D2, D3 must not be used for addressing!

Note: Boolean values will be allocated bytewise, if no explicit single-bit address is specified. Example: A change in the value of varbool1 AT %QW0 affects the range from QX0.0 to QX0.7.

Note: Online Change might change the contents on addresses. Please notice this when using pointers on addresses.

16.4.4 Functions

In ST a function call can also appear as an operand.

Example:

```
Result := Fct(7) + 3;
```

TIME()-Function

This function returns the time (based on milliseconds) which has been passed since the system was started.

The data type is TIME.

Example in IL:

```
TIME
ST systime  (* Result for example: T#35m11s342ms *)
Example in ST:
systime:=TIME();
```

17 Devices

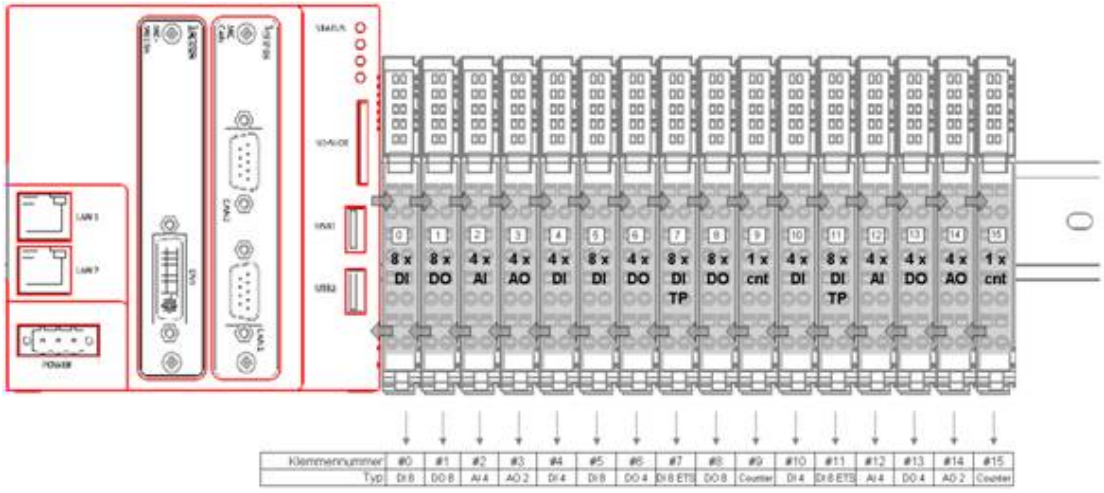
Note on use of a Hilscher cifX card

For the use of a cifX card from Hilscher GmbH (with appropriate IEC library as driver) the »PLC Designer« configuration file (.cfg) has to be enlarged by the following entry:

Component.<next free number>=CmpHilscherCIFX

17.1 Motion Controller

The following illustration is a schematic representation of the hardware structure of a Motion system.



The Motion system consists of the following components:

- L-force Controller 3221 C or 3231 C with a programmable logic controller in the form of a soft PLC
- I/O compound modules which exchange process data, parameter data and diagnostic information with the L-force controller via an internal backplane bus

In order to gain access to the I/O compound modules in a PLC program, i.e. read input signals and write outputs, a terminal configuration has to be created in PLC Designer.

To do this, the arrangement of the I/O compound modules must be simulated in the control configuration of the PLC.

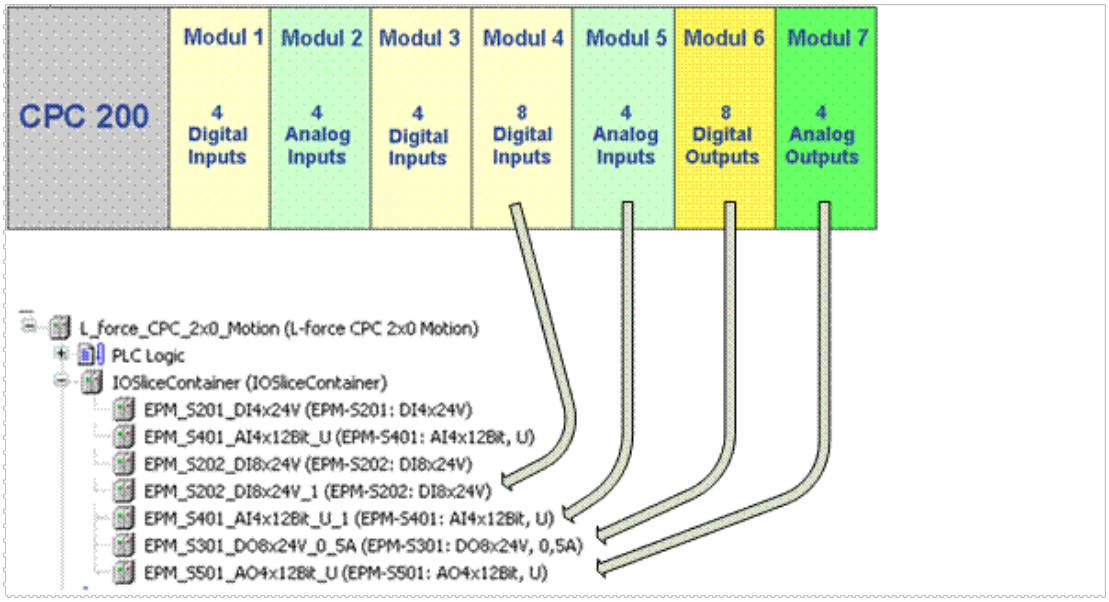
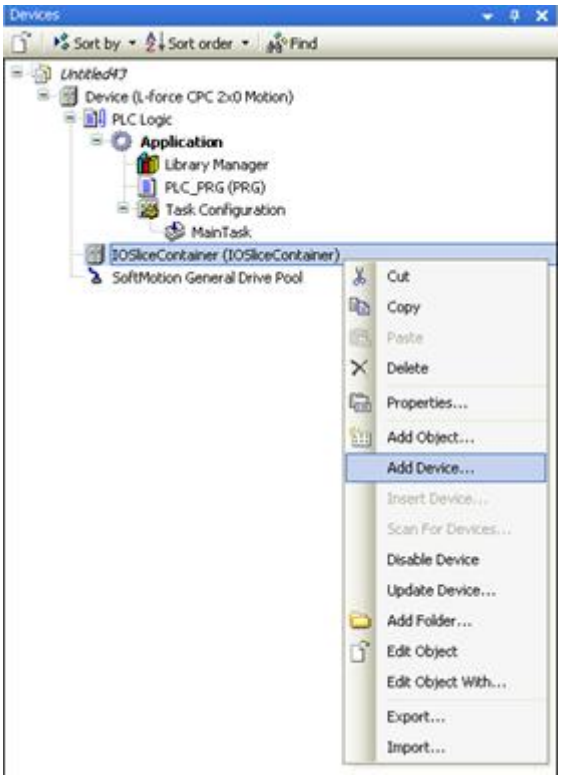


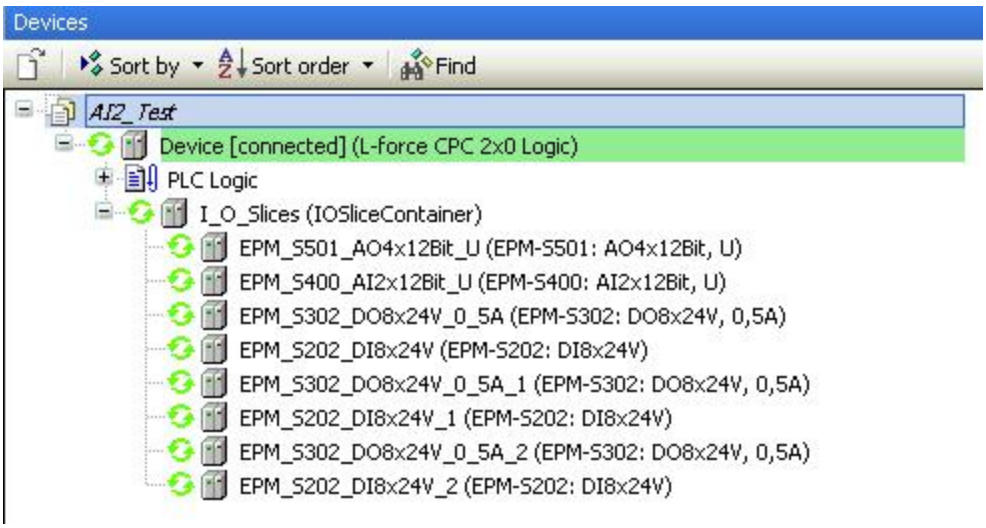
Abb. 543 Example: Mapping of the physical topology in the control configuration

Configuration takes place in that the I/O compound modules are inserted as corresponding devices under the L-force controller in *IOSliceContainer*:



It is important that the sequence of I/O compound modules in the device tree matches the physical arrangement.

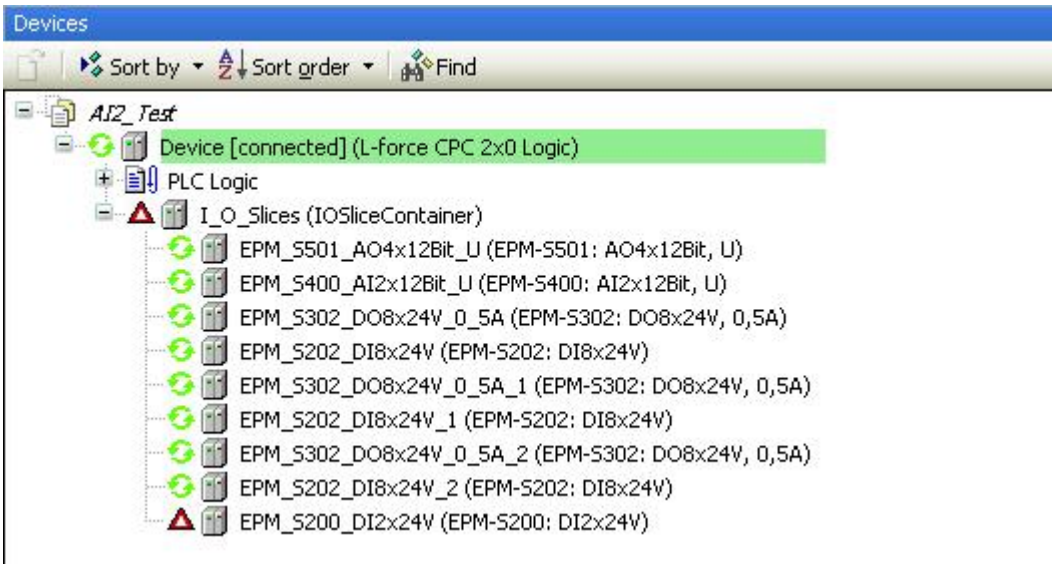
If everything has been configured correctly, all I/O compound modules are marked with a green circle in the device tree after the user goes online and starts the PLC.

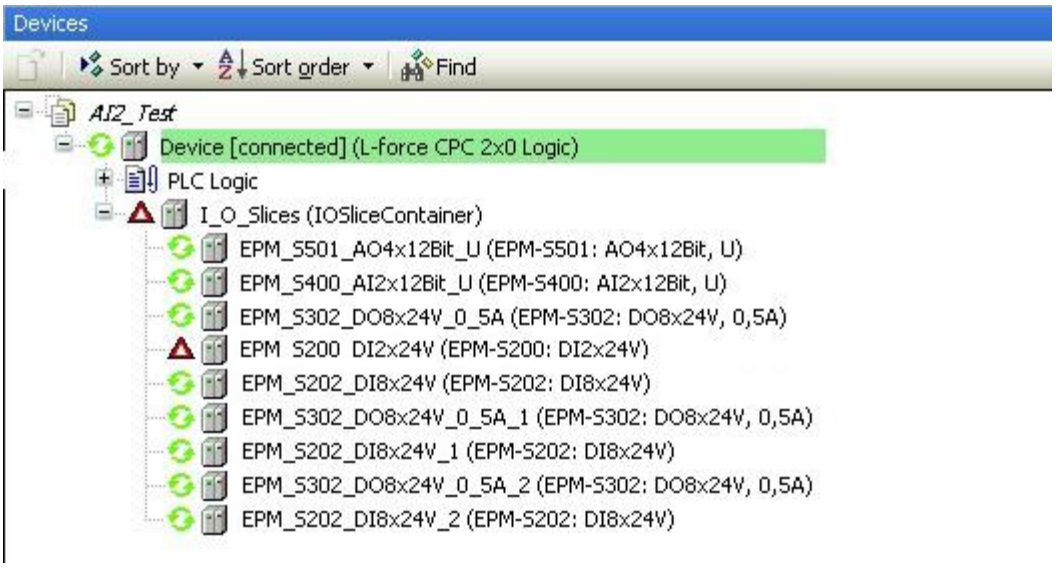


In the event of an error, i.e. the configuration does not match the physical arrangement, the I/O compound modules are marked with a red circle.

Examples of incorrect configurations:

EPM_S200 does not exist in the physical configuration

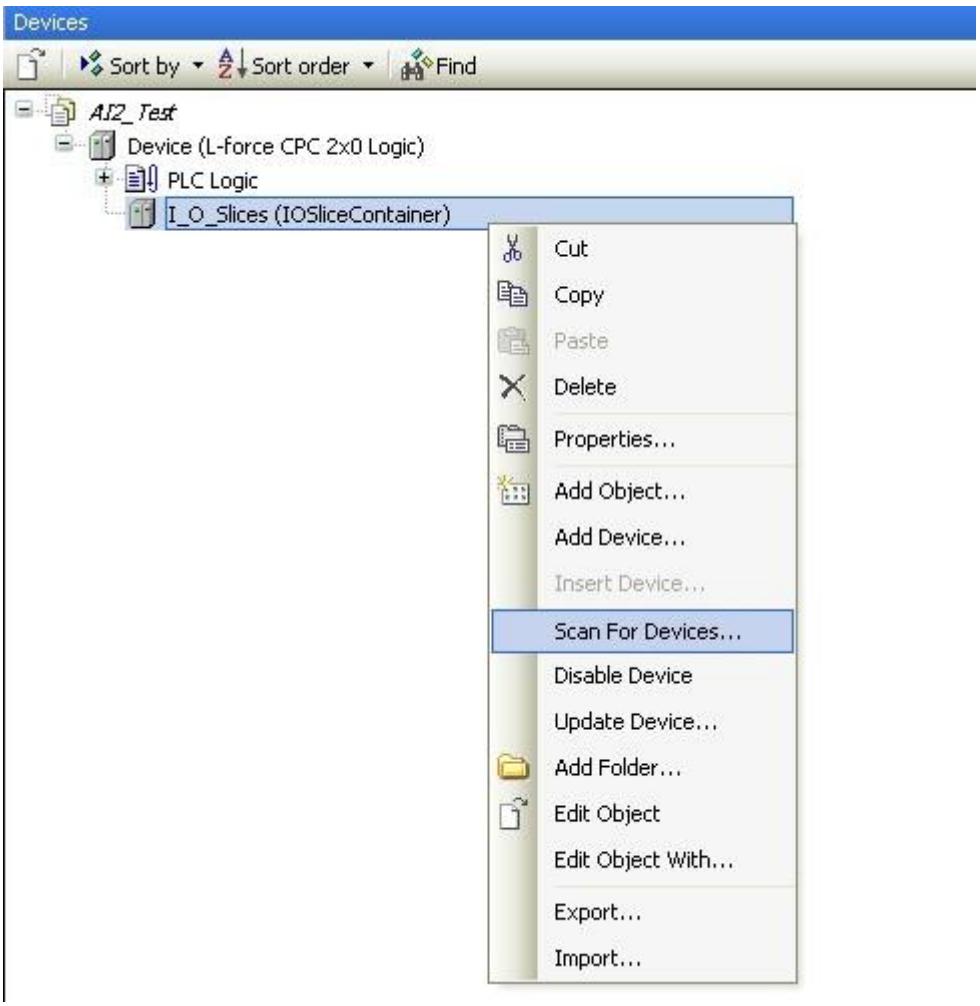




17.1.1 Automatic detection of the I/O compound module topology

In addition to manual configuration, automatic detection of the I/O compound module in the PLC Designer projecqt tree is possible.

The function is selected by means of the menu item "Scan For Devices" in PLC Designer:



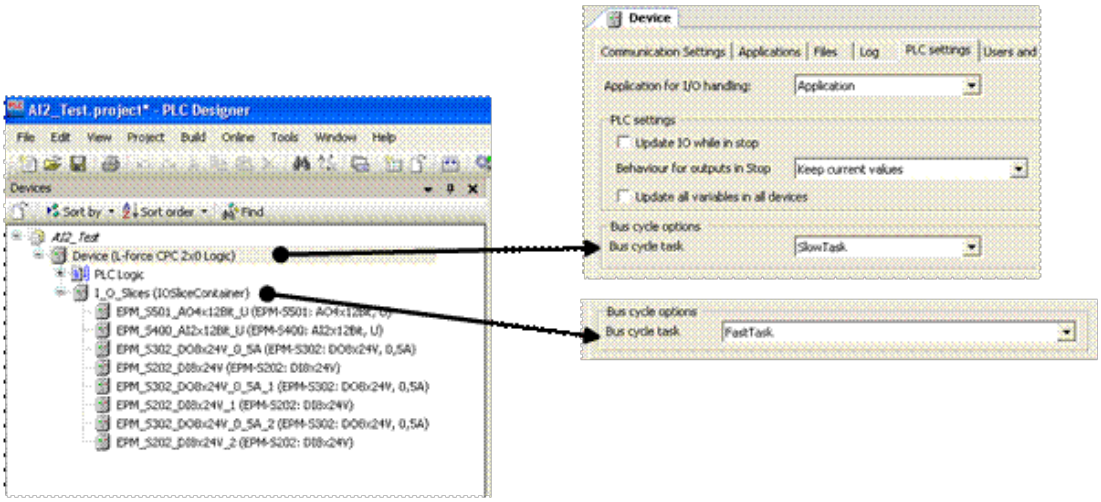
17.1.2 Adjusting the cycle time of access to the I/O compound module

In the "L-force Controller" device, a bus cycle task for processing all connected field buses and for the I/O compound module connected via the backplane bus is selected centrally.

This selection which is made is a pre-assignment and can be overwritten in the bus master nodes (EtherCAT Master, CANopen_Manager, IOSliceContainer, ...).

If the backplane bus is to be operated with a shorter cycle time than selected in the "L-force Controller" device, a faster task can be selected in 'IOSliceContainer'.

Example: Different cycle times



In PLC Designer, a processing task for formation of the I/O image is first created for the PLC. A different possibility, however, is to specify a so-called bus cycle time for each individual I/O compound module, whereby the process image for these I/O compound modules is created in this time.

All in all, three different tasks can be defined for I/O compound module processing. The restriction to three tasks is due to the nature of the system and is prescribed by the backplane bus specification. The backplane bus controller uses two internal bus cycle groups for each task. One for input signals and one for the outputs. There is a total of 6 groups available.

18 PLC Designer Libraries

It depends on the target system, which of the following libraries can be used in the control program:

- **Library Standard.library**
- **Library Util.library**
- **Bus specific Libraries:**
  - library IIoDrvBusControl.Library
  - library IIoDrvDPV1C1.library
  - library IIoDrvDPV1C2.library
  - library IIoDrvProfibus.Library
  - library IIoDrvAcyclicServices.library
  - library CAA_CiA405.lib
- **Library SM3_Basic.Library**
- **Library SM3_CNC.library**
- **System Libraries:**
  - Runtime Error Codes
  - SysCom.library
  - SysComAsync.library
  - SysCPUHandling
  - SysDir.library
  - SysDirAsync.library
  - SysEvent.library
  - SysFile.library
  - SysFileAsync.library
  - SysInt.library
  - CoDeSys V3.x System Libraries
  - SysMem.library
  - SysPci.library
  - SysPort.library
  - SysPortAsync.library
  - SysProcess.library
  - SysSem.library

  - SysSemProcess.library
  - SysShm.library
  - SysSocket.library
  - SysSocketAsync.library
  - SysTask.library
  - SysTime.library
  - SysTimer.library
  - SysTypes.library
- **CAA Libraries:**
  - CAA_AsyncManager.library
  - CAA_CallBack.Library
  - CAA_CanL2.library
  - CAA_CanOpenMan.library
  - CAA_CanOpenStack.library
  - CAA_CiA405.library
  - CAA_DTUtil.library
  - CAA_FILE.library
  - CAA_Mathematics.library
  - CAA_MemBlockMan.library
  - CAA_Memory.library
  - CAA_NetBaseServices.library
  - CAA_RealTimeClock.library
  - CAA_ResMan.library
  - CAA_SegBufferMan.library
  - CAA_SerialCom.library
  - CAA_Settings.library
  - CAA_Tick.library
  - CAA_Timer.library
  - CAA_Types.library

Note: For more Information about the libraries see the PLC Designer Online Help.

19 Index

__DELETE 1198, 1200
__ISVALIDREF 1141, 1155, 1200
__NEW 1198, 1200
__QUERYINTERFACE 1203
__QUERYPOINTER 1205
1006h object 977
3D mode 700
abort if error 988, 1006
About 407
ABS 1192
absolute 642
absolute address 126
absolute movement of visualization elements 891
absolute value 1192
access 1005
access control 152, 605
access control properties 605
access on variables 1127
access rights 186, 583, 586
access rights on device 966
accessrights
  data source 742
account properties 401
ACOS 1196
action
  add 96
  call action 96
action 96
action association 543
action call 91
action call in IL 766
action in SFC 814, 816
action name 1073
action qualifier 543, 823
action status 824
action width 501
actions
  rights 586
actions on device 966
activate keyboard usage 595
activate watchdog 858
active application 437, 445
Active editor 348
acyclic transmission 984
adapt the dimensions automatically 268
ADD 1161
add a new recipe 532, 797
add action 96
add all forces to watchlist 464
add data server 740
add data source 740
add data type unit 85
add device 115, 296, 312, 944
add event 863
add external file 102
add folder 438
add function 73
add gateway 313, 944
add group 401, 963
add interface 93
add language 566
add library 421, 791
add library repository 409
add mapping 984
add method 88
add object
  method 88
add object 73, 85
add object 93
add object 96
add object 102
Add Object 431
Add Objects From Source Control 553
add PDO 984

add POU to task 858
add repository 415
add user 401, 963
add variable to recipe 797
Add Visualelement 589
addition 1161
address
  absolute 126
  broadcast 126
  change manually 956
  determination 124
  FBD LD and IL editor 494
  fix value 956
  gateway channels 944
  relative 126
  structure 126
address 124
address 1082
address 1219
address 1220
address in IL 766
address of device 301
address of device I/Os 956
address of I/O-channel 956
address operator 1181, 1207
Address Operator ADR 1143, 1157
address prefix 1220
Addressing 124, 126
Addressing and Routing 126
addressing bits 1216
ADR 1143, 1157, 1181, 1207
Advanced Settings for data source control 549
alarm color 891
ALIAS 1141, 1155
Align bottom 595
Align horizontal center 595
Align left 594
Align right 595
Align top 594
Align vertical center 595
alignment 194, 889
All open editors 348
allow editing of read-only projects 473
Alt-Click 476
alternative 541
alternative branch 541, 816
always update variables 956
analyze dynamics 288
AND 1167
AND operator in IL 759
ANDN 1168, 1209
ANY_TYPE 1139, 1140
appearance of trace 880
append device 296
app-file 448
application
  active 437
  arrangement 115
  boot application 448
  build 260, 261
  clean 264
  details 139
  dynamic memory settings 605
  father and child 115
  for symbol information 241
  generate code 261
  name 115
  object properties 605
  online change 139
  options 605
  rebuild 261
  remove from device 470
  remove from PLC 949
application 119
application download 450
application for I/O handling 954

application forcelist 464
application information dialog 139
application online change 451
application online status 174
application start 453
application stop 453
applications dialog 949
applications hierarchiy 119
applications on PLC 949
applicaton
  hierarchy 115
arc 688
arc cosine 1196
arc sine 10-26 1196
arc tangent 1197
archive
  extract archive 369
  project archive 369, 372
  save archive 372
  send archive 372
archive 369
archive 372
archive.prj 452
array
  expandfully 1108
array 1216
ARRAY 1146
array as input in visualization 1103
array of structure 1146
array visualization 223
arrays
  initialization 1146
arrays 1146
arrays for visualization 1108
ASCII 266
ASIN 1196
Assign other names 1141, 1155
assignment 1165
Assignment 842
Assignment as expression 841
assignment in FBD and LD 326
assignment in LD 782
assignment operator 844
association for SFC action 543
aste contacts 339
asynchronous transmission 984
AT 1071, 1082
AT declaration 1082
ATAN 1197
attribute
  call_after_init 1106
attribute displaymode 1107
attribute ExpandFully 1108
attribute Global_init_slot 1109
attribute Hide 1110
attribute Hide_all_locals 1111
attribute init_namespace 1112
attribute Init_On_Onlchange 1113
attribute Initialize_on_call 1112
attribute Instance-path 1113
attribute linkalways 1114
attribute No_check 1117
attribute No_Copy 1117
attribute No_virtual_actions 1120
attribute No-exit 1118
attribute noinit 1119
attribute pack_mode 1123
Attribute Parameterstringof 1124
attribute Qualified_Only 1125
attribute Reflection 1125
attribute symbol 1127
attribute width 501
attributes
  monitoring 1115
attributes for variables types 1090
attributes of a SFC step 814

author name 388
auto clear mode 1028
auto declare 535, 1080
auto hide a view 628
auto hide a view window 628
auto restart slaves 997
auto save backup 358
auto save backup dialog 473
autoconfig master/slaves 997
autoconfig PDO mapping 979, 980, 984
autoconnect in CFC 492
autodeclaration dialog 1080
autodeclare 485
autodeclare option 476
autoformat 485
autoinc address 1001
autoincremental address 1001
automatic mapping 979
automatic startup 1028
automatically generate POUs for property access 388
automatically list selection in cross reference view 485
automatically log out after time of inactivity 394
automatically save 473
autosave file 473
available devices 301
available elements 415
average cycle time 864
axis control
  example 1058
Axis Error Messages SoftMotion 57
axis motion control
  example 1053, 1055
AXIS_REF_LOGICAL_SM3 1047
B 1220
background 593
background color for text items 486
background image 112
background image of visualization 605
backup file 366, 473
bar display 214
bar display configuration 214
batch mode
  project 355, 358, 365, 367, 375
baud rate 1028
Baudrate for CANbus 972
behaviour for outputs in stop 954
best fit in online mode 936
binary 469
bind new objects to source control 508
binding
  all objects 549
binding 549
binding 605
binding of an object 605
bit access 748, 1216
bit address operator 1181, 1207
bit operation 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1209
BITADR 1181, 1207
bitmap
  associate to object 605
  transparent 605
bitmap 891
bitmap for box in FBD/LD/IL 494
bitmap for visualization 593
bitmap for visualization background 593
bitmap id variable 891
bitmap ID variable 112
bitmap ID variable for visualization elements 891
bitmap ID variables for visualization elements 891
bitmap info of visualization elements 891
bitmap pool 408
bitmap properties 605
bitmapinfo 891
bitmaps in visualization 408
bits in variables 1216

bookmark 249, 250
bookmarks commands 249
BOOL 1137
BOOL constants 1211
BOOL_TO Conversions 1184
BOOL_TO_DATE 1184
BOOL_TO_DT 1184
BOOL_TO_INT 1184
BOOL_TO_STRING 1184
BOOL_TO_TIME 1184
boot application
  remove from device 470
boot application 139, 448
boot application settings 605
boot project
  remove 470
boot project 139, 448
border color 476
box 328, 331
box icon for FBD/LD/IL 494
box in CFC 659
box in FBD/LD/IL 331, 778
box parameters update 345
bracket 565
bracket scope 476
branch
  alternative 816
  parallel 816
branch 816
branch in FBD/LD 344, 345
branch in FBD/LD/IL 779
branch in SFC 541, 816
branch label 541, 816
branch/alternative 541
branch/parallel 541
breakpoint
  CFC Editor 668
  clear 599
  clear all 599
  condition 251
  current hit count 599
  disable all 599
  display 254
  enable all 599
  FBD/LD/IL Editor 784
  go to source position 599
  location 251
  new 251, 599
  properties 599
  ST-Editor 836
  toggle 251
breakpoint 144
breakpoint 251
breakpoint 599
breakpoint 784
breakpoint in function blocks 251
breakpoint position 254
breakpoint symbols 254
breakpoints commands 250
breakpoints dialog 599
breakpoints in ST-Editor 836
Bring one to front 594
Bring to front 594
broadcast address 126, 605
broadcasting 131
browse cross references 258, 596
browse location 605
browse project commands 257, 258
buffer 1005
buffersize
  trace 871
build
  enable system call 605
  exclude from build 605
  external implementation 605
  link always 605

pool objects 264
syntax check 260
warning via pragma 1122
build 261
build application 260, 261
build commands 260
build menu 179
build messages 260
build options for SFC 504
build process 139
build properties 605
bus cycle options 954
bus cycle task 954
Bus specific Libraries 1230
button 208, 891
button height 891
buttonheight 891
buttonstate variable 891
buttonstate variables for visualization elements 891
BYTE 1137
byte adressing 1220
BYTE_TO_BOOL 1185
C in CoDeSys 60, 262
C instruction 759
C modifier 759
C routines 60, 262
C stub file for external library 262
CAA Libraries 1232
CAA_AsyncManager.library 1232
CAA_CallBack.Library 1232
CAA_CanL2.library 1232
CAA_CanOpenMan.library 1232
CAA_CanOpenStack.library 1232
CAA_CiA405.lib 1230
CAA_CiA405.library 1232
CAA_DTUtil.library 1232
CAA_FILE.library 1232
CAA_Mathematics.library 1232
CAA_MemBlockMan.library 1232
CAA_Memory.library 1232
CAA_NetBaseServices.library 1233
CAA_RealTimeClock.library 1233
CAA_ResMan.library 1233
CAA_SegBufferMan.library 1233
CAA_SerialCom.library 1233
CAA_Settings.library 1233
CAA_Tick.library 1233
CAA_Timer.library 1233
CAA_Types.library 1233
CAL 328, 1183
CAL operator in IL 759
calculate active transitions only 504
call of a program 71
call parameters 78
call stack 601
call_after_init 1106
calling a method 82
calling operator 1183
CAM
application example 1058, 1060
basic settings 605
compile format 605
continuity 605
dimensions 605
Element properties 654
master and slave 642
offset 642
periodic 605, 642
properties 605
read CAM data from ASCII table 266
read CAM online file 266
scaling 642
start mode 642
switching 642
write CAM data into ASCII table 266
write CAM online file 266

CAM 264
CAM 654
CAM Commands
read CAM data from ASCII table 266
read CAM online file 266
write CAM data into ASCII table 266
write CAM online file 266
CAM Commands 264
CAM editor 1039
CAM Editor 264, 605, 642, 650, 651, 652, 653, 654
CAM properties 605
CAN communication 55
CAN over EtherCAT 1018
CANbus
Baudrate 972
CANopen I/O Mapping 992
CANopen Manager 977
CANopen module 993
CANopen-Module Configuration 992
I/O Mapping 993
Net 972
network number 972
parameters 974
Receive PDO Mapping 984
CANbus Configuration
CANopen device 980
modular devices 979
non-modular devices 979
CANbus Configuration 974
CANbus node
CANbus 972
CANbus Configuration 974
Information 974, 979, 992, 993
CANopen Configuration 990
CANopen device
CANopen Configuration 990
CANopen Remote Device 980
parameters 990
Receive PDO Mapping 984
Send PDO Mapping 988
Service Data Object 988
CANopen device 979
CANopen I/O Mapping 992
CANopen Manager
Common.CANbus Configuration 977
CANopen Manager 977
CANopen module
I/O Mapping 993
CANopen module 992
CANopen Remote Device 980
CANopen-Module Configuration 992
caret color 476
CASE instruction 842, 847
category description file 146
category for library 146
category of a library 409, 421
CDS_MemMan 1198
center 891
center of visualization elements 891
CFC
arrange elements 665
box 659
comment 659
composer 659
copy 665
cursor positions 658
delete 665
dimensions 268
edit 665
edit working sheet 268
execution order 665
grid unit 268
input 659
insert 665
jump 659
Label 659

monitoring 668
move 665
offset 268
options 492
output 659
return 659
select elements 665
selector 659
set/reset 272
size 268
VAR_IN_OUT 278
VAR_INPUT CONSTANT 277
working sheet size 665
CFC 658
CFC Editor
breakpoint positions 668
monitoring 668
online 668
CFC editor commands 267
CFC editor options 492
CFC elements 659
CFC execution order 267
CFC toolbox 659
cfg-file 115
ch 174
change binding 605
Change database connection 553
change device name 315
change language 891
change namespace 426
change properties heartbeat consumer 980
change shown visualization 891
change the camera's pitch 722
change the camera's yaw 722
change the language 891
change variable values 704
change visualization 891
channel
address 956
channel 956
channel of device I/Os 956
channel type 956
character set 891
charset 891
check all pool objects 264
check for updates when loading this project 386
check function 1143, 1146, 1152, 1157, 1163
check functions
suppress 1117
check functions 114
check in 556, 624
check out 555
Check Visualization Text Ids 573
CheckBounds 73, 1146
CheckDivByte 73
CheckDivDWord 73
CheckDivInt 1163
CheckDivLInt 1163
CheckDivLReal 1163
CheckDivReal 73, 1163
CheckDivWord 73
checkin 556, 624
checkin after successful build 508
checkin before project save 508
checkin to database 556
Checkout 555, 557
checkout before object edit 508
checkout from database 555
CheckPointer 1143, 1157
CheckRangeSigned 73, 1152
CheckRangeUnsigned 73, 1152
child application 115, 119
circle 688
clean 264
clean all 264
clear all breakpoints 599

clear bookmarks 250
clear breakpoint 599
clear log list 951
clear permission 966
click 476
client height 936
client width 936
cliententries for visualization 605
clipboard commands 292
clipping 891
clipping of visualization elements 891
close all editors 629
close editor 629
Close Project 365
Close view 169
cmp name 951
CNC
  control panel 722
  DIN66025 679
  online 735
  Parameter editor 738
  properties of CNC object 730
  properties of path element 724
  table editor 719
  table view 719
  text editor 677
  toolbox 725
CNC 730
CNC 730
CNC commands
  analyze dynamics 280
  import from DXF file 280
  load program from ASCII file 280
  move program 280
  renumber CNC program 280
  reverse program 280
  rotate program 280
  scale program 280
  scroll path view 280
  set variables 280
  show end points 280
  show grid 280
  show interpolation points 280
  show preprocessed path 280
  step suppression 280
  write program to ASCII file 280
CNC commands 280
CNC decoding online with use of variables 1064
CNC editor 1039
CNC example 1 1060
CNC example 2 1064
CNC example 3 1067
CNC graphical editor 730
CNC language 679
CNC new object 730
CNC online 735
CNC program 730
CNC settings 730
CNC table editor 730
CNC text editor 730
CNC toolbox 725
CNC_File.project 1069
CNC_table.project 1069
COB-ID 980, 984
code generation 139, 450, 1128
code generation for SFC 504, 605
code generation offline 261
CoDeSys 2.3 converter 358
CoDeSys concepts 60
CoDeSys HMI 188, 206
CoDeSys SP service 153
CoDeSys V2.3 converter 490
CoDeSys V2.3 projects 61, 66
CoDeSys V3.x System Libraries 1231
coil 326, 335, 782
coil/set and reset 783

col 174
cold start 457
color definition 891
color for visualization background 593
color of text items 486
color variables for visualization elements 891
colors 891
colors for text 476
colors of visualization elements 891
command
  execute by visualization input 891
command category
  bookmarks 249
  breakpoints 250
  browse project 257, 258
  build 260
  clipboard 292
  Declaration 294
  device communication 311
  devices 295, 301
  Devices 308
  FBD/LD/IL 324
  Find/Replace 348
  Help 406
  image pool 407
  Installation 408, 414
  message view 428
  objects 438, 439
  Online 443
  preferences 470
  print 519
  recipe commands 531
  smart coding 533, 535
  text editor 564
  text list 407, 566
  undo/redo 581
  User Management 582
  view 602, 622, 623, 625
  View 596, 598, 603, 624, 626
  Visual Commands 588
  window 627
command category 514
command category 518
command line option
  culture 1070
commands 179
comment
  FBD LD and IL editor 494
comment 939
comment in FBD/LD/IL 776
comment in Instruction List 757
comment state in FBD/LD 325
comments in ST 852
common properties 605
Common.CANbus Configuration 977
communication
  configure list of devices 944
  gateway 944
Communication
  Addressing and Routing 124, 126
  Network topology 123
  network variables 100, 131
communication cycle in CANbus 980
communication cycle period 980
communication parameters 944
communication parameters for data server 742
communication path to device 324
communication settings dialog 944
company 421
company name 388
compare 379
compare projects 379
compatibility profile 200
compilation 260, 1128
compile errors 261
compile format for CAM 605

compile information 139, 264, 448, 450, 1114
compile options 394
compile process 139
compiled library 146
compiled library files 375
compileinfo-file 139, 264, 448, 450
compiler defines 605
compiler version 386, 394
complete access on object dictionary 1006
complex control 211
complex controls 214, 217
composer in CFC 659
compress 577
compression of the project 473
concepts of CoDeSys 60
condition
  breakpoint 251
conditional call 759
conditional compilation 605
conditional jump 759
conditional pragmas 1128
conditional return 759
conditonal pragmas 1103
config 1035
configread 1013
configuration
  scan 301
  trace 871
configuration dialog 500
configuration dialog for task 858
configuration file 115
configuration of a device 948, 1043
configuration of DP modules 1036
configuration of Profibus DP 1036
configuration of visualization elements 891
Configuration within the Device Softmotion Editor 1043
connect boxes with straight lines 494
connect in FBD/LD/IL 779
connect selected pins 278
connect to device 470
connect to local gateway 315
connected drive 1042
connection in FBD/LD 343
constant
  compile options 394
constant 1211
constant 1211
constant 1212
constant 1213
constant 1214
constant 1214
constant 1215
CONSTANT
  online 1093
CONSTANT 1090
CONSTANT 1093
constant inputs in CFC 277
constant variable
  online mode 753, 1093
  replace constants 394
constant variable 1093
contact 335, 338, 781
contact parallel 338, 339
container libraries 146
container library 146
content of project archive 394
content operator 1182
Content Operator 1143, 1157
contents of online help 406, 407
contents... 406
context sensitive help 153
CONTINUE 848, 849, 851
continuity requirements of a CAM 605
Continuous Function Chart 658
control panel 722

controller
reset 470
conversion
library 487
conversion of a project 487
conversion of device entries 358
conversion of devices 490
conversion of FBD/LD/IL views 346, 347
conversion of integral number types 1186
conversion of projects 358
conversion of referenced libraries 358
conversion of V2.3 projects 66
convert and install the library as well 358
convert keywords to uppercase automatically (autoformat) 485
convert project 358
convert project file 375
converter 61
converter application 375
converter for 2.3 projects 358
converter options 490
coordinate shift 709
coordinates in visualization editor 891
copy 293
copy element in CFC 665
copy element in FBD/LD editor 762
copy element in SFC 812
copy in IL 766
copy objects 375
copy project 375
COS 1195
cosine 1195
create all SDO's 980
create backup files 366, 473
create boot application 448
create global text list 567
create GlobalImagePool 408
create recipe 797, 891
CreateRecipe 801
CreateRecipeNoSave 801
cross reference list 258, 596
cross reference view 258, 485, 596
cross references in message window 596
C-routine 146
Ctrl-Click 476
culture 484, 1070
current hit count 599
current password 394
current state of EtherCAT slave 1001
current user 174, 585
current version 358
cursor
trace 575
cursor 575
cursor position 174
cursor positions in CFC 658
cursor positions in FBD 772
cursor positions in LD 772
cursor positions in SFC 811
curve 208, 891
customize
keyboard 518
menu 514
toolbars 513
customize user interface 169
Cut 292
cut element in SFC 812
cut in IL 766
cycle
process single cycle 453
cycle count 864
cycle length 864
cycle time in CANbus 980
Cycle time violation 56
cyclic task 858
cyclic transmission 984

cyclic transmittsion interval 605
D 1138, 1220
d# 1212
dar-file 966
data array in visualization 223
data base 243
data control time 1028
data exchange 130, 131
data flow in CFC 274
data server
  add 740
  communication 742
data server 130
data server 742
data server for visualization 199
data source
  add 740
  configure 742
  login configuration 742
  visualization 748
data source 130
data source for visualization 199
data source items 742
data structures 650
data type
  obsolete pragma 1122
  user defined 1096
  warning 1122
Data type 85
data type name 1073
data type unit
  extending 85
Data Type Unit (DUT) 85
data types
  enumeration 1150
  norm-extending 1139
  reference 1141, 1155
data types 1139
database 508
Database connection 549
database for devices 308
database for libraries 409
database location 549
database object history 558
database project history 560
DATE 1138
date and time in visualization 194
DATE Constants 1138, 1212
DATE_AND_TIME 1138
DATE_AND_TIME Constants 1138, 1213
DATE_TO conversions 1188
dcf-file 308
DCF-file 980
debug functionality 144
debug menu 179, 443
debug mode
  visualization 510
debugging
  CFC Editor 668
  FBD/LD Editor 784
  ST-Editor 836
debugging 144, 250, 254
debugging in ST-Editor 836
decimal 469
Declaration
  functions 1097
  methods 1097
Declaration 1071
declaration editor 174, 1080, 1081
declaration editor commands 294
declaration editor online view 753
declaration editor options 493
declare unknown variables automatically 485
default appearance 577
default binding 605
Default binding 549

default library 421, 426
default navigators 602
default right 186
default settings for SFC 605
default slave ratio for heartbeat time in percent 977
default value of device parameter 948, 1043
define
  pragma 1128
define 1128
defined 1128
Defined Data Types 1146
delay time 691
delete 294
delete element in CFC 665
delete element in SFC 812
delete group 963
delete IL line 340
delete recipe 797
delete selected device 322
delete user 963
DeleteRecipe 801
DeleteRecipeFile 801
denied rights 586
deny permission 966
dependencies of a library 409
dereferencing 1182
Dereferencing 1143, 1157
destination address mac 997
destination device for sourcedownload 394
details
  application 139
Details
  Object history 558
details of database object 558
details of database project version 560
details on library 409
detect external changes 473
devdesc.xml file 115, 308
device
  access rights 966
  add 296
  add to list 312
  CANopen Remote Device 980
  connect to 470
  delete selected from list 322
  description 308
  description file devdesc.xml 115
  disable and enable 300
  disconnect from 470
  Generate EtherCAT XML 299
  information 969
  inserting 296, 298, 299
  installation 115, 308
  location 308
  name 296
  plug 296
  repository 308
  simulation 464
  slot 299
  update 299
  user management 963
  vendor 296
  version 296
device 115
device 295
device 301
device 308
device access rights 966
device baseaddress 956
device commands
  Generate EtherCAT XML 299
  Generate SercosIII SCI XML 299
device communication
  add device 312
  add gateway 313
  change device name 315

connect to local gateway 315
delete selected device 322
edit gateway 323
set active path 324
device communication 311
device communication 944
device communication commands 311
device configuration 948, 1043
device conversion 358, 490
device conversion mapping 358
device database 308
device description 115
device description file 115, 308
device dialog
access rights 966
applications 949
ccommunication settings 944
configuration 948, 1043
I/O mapping 956
users and groups 963
device dialog 115
device editor options 500
device editor softmotion 1039
device for CANopen
modular 979
non-modular 979
device for standard project 355
device identification 301
device in device tree 115
device location 308
device management 497
device name
change 315
device name 301
device name 969
device order number 969
device parameters 948, 1043
device repository
availability 497
device repository 115, 308
device repository plug-in 308
device reset 470
device scanning 301
device settings 954
device softmotion editor
configuartion of devices 1043
device softmotion editor 1043
device status 1038
device tree
how to arrange 115
device tree 60
device tree 115
device tree 301
device type 301, 969
device user management 963
device version 969
Devices
Generate SercosIII SCI XML 299
devices commands 295, 301
Devices commands 308
devices tree 295
devices view 602
Devices window 174
device-specific library 421
diagnosis visualization 748
Diagnosis Visualization
restrictions 203
Diagnosis Visualization 188
Diagnosis Visualization 203
diagnostics 1001
diagnostics message 997, 1001
dialog in visualization 188, 236, 891
dialogmanager 236
differences of database objects 557, 558
differences toCoDeSys V2.3 61

dimensions of a CAM 605
dimensions of CFC 268
DIN66025
  sentence 679
  sentence number 679
DIN66025 679
DINT 1137
Direct Creation of the OutQueue 1060
disable all breakpoints 599
disable breakpoint 599
disable device 300
disable drive 1050
disconnect from device 470
display all versions 421
display mode 469
display object histories 560
display target-visualization 936
displaymode 1107
distributed clocks 1001
DIV 1163
division 1163, 1164
Do not get the latest version 555
dock 627
docking a view 627
Docking views 169
document project 520
document settings 521
documentation
  data source 742
documentation of a library module 791
documentation of library modules 146
documentation of the project 248
download
  information at login 445
  multiple 454
  remanent variables 1090
download 394
download 452
download 1099
download application 1090
Download Application 450
download of project 452
download of source code 452
download to device 963, 966
download trace 574
DP libraries 1038
DP master
  group properties 1028
  groups 1028
DP master 1028
DP parameters
  DP master 1028
  DP slave 1032
  DP-Module 1035
  freeze mode 1028
  groups 1028, 1032
  sync mode 1028
DP slave 1032
DP-Master 1036
DP-Module 1035
DP-Module configuration 1036
DP-Slave 1036
drive
  example application 1053
drive configuration 1039
drive control
  example 1058
drive interface 1039
drive logical 1047
driver type 944
DT 1138
dt# 1213
DT_TO conversions 1188
dum-file 963
DUT 85
DUT name 1073

DWORD 1137
DXF file 287
dynamic creation of objects 1198, 1200
dynamic image 112
dynamic memory settings 605
dynamic text 105, 194
dynamic texts 891
dynamic texts for visualization elements 891
dynamic variables 891
dynamic variables for visualization elements 891
edge detection in fbd/ld 342
edit commands 292, 581
edit control closing 229
edit declaration header 294
EDIT FMMU 1005
edit gateway 323
edit gateway settings 944
edit group 401, 963
edit library repository 409
edit locations
  visual element repository 415
edit locations 415
edit menu 179
Edit Object 433
Edit Object (Offline) 433
Edit Object with 433
edit parameters 277
edit repository 415
edit title page 521
edit user 401, 963
edit working sheet 268, 665
editing a visualization 192
editing mode 174, 564
editing options 476
editor options 476
editor view 174
eds-file 115
EDS-file 980
Eingabekonfiguration
  visualization element 891
Eingabekonfiguration 891
element
  insert in visualization 891
  visualization 885, 887
element 891
element in FBD/LD editor 762
Element in Visualisierung 891
element list 593
element number in CFC 274, 275, 277
element numbers in CFC 665
element order in CFC 274, 275, 277
element properties
  CAM 654
element properties 623, 625
Element Properties 814
element repository 415
Elementeigenschaft in Visualisierung 891
elementlook 891
elementlook of visualization elements 891
Elementname (Visualisierung) 891
elements arranging in CFC 665
ellipse 208, 698, 891
EN/ENO 271, 328
enabeltaskoutputmessage 1013
enable all breakpoints 599
enable autoconnect 492
enable breakpoint 599
enable cycle 980
enable drive 1050
enable expert pdo settings 980
enable expert settings 1001
enable heartbeate generation 980
enable messages per task 997
enable nodeguarding 980
enable project file encryption 394
enable sync 0 1001

enable system call 605
encoding library 375
encoding of libraries 146
encryption password 394
end of line markers 476
END_STRUCT 1149
END_UNION 1139
Engineering Interface 243
ENI 243, 548
ENI client 243
ENI Server 243, 508
ENI Server login 563
enter password 963
Entire project 348
entry action 814, 816
Enumeration 1150
enumeration name 1210
EQ 1180
EQ operator in IL 759
equal to 1180
error 1101
error analyzation in SFC 824
error in SFC 824
error in task configuration 864
error message 622
esd-file 308
ETC_CO_SdoRead 1018
ETC_CO_SdoRead_Access 1022
ETC_CO_SdoRead4 1021
ETC_CO_SdoReadChannel 1022
ETC_CO_SdoWrite 1023
ETC_CO_SdoWrite4 1024
ETC_CO_SdoWriteAccess 1025
ETC_CO_SdoWriteDWord 1025
ETCSlave 1016
EtherCAT
  CoE mailbox 1025
  IODrvEtherCAT.library 1018
  master instance 1013
  read and write parameters 1018
  read parameter 1018, 1021
  restart master 1013
  slave instance 1016
  stop bus 1013
  write parameter 1023, 1024
EtherCAT Address 1001
EtherCAT Configuration
  Autoconfiguration and expert mode 997
EtherCAT Configuration 1000
EtherCAT Configuration 1011
EtherCAT configuration status 1013
EtherCAT I/O Mapping 1000, 1011
EtherCAT Master
  auto restart slaves 997
  enable messages per task 997
  properties 1013
  restart 1013
EtherCAT Master 997
EtherCAT Master configuration 997
EtherCAT Master parameters 1000
EtherCAT network name 997
EtherCAT Slave
  autoconfiguration 1001
  complete access 1006
  expert mode 1001
  expert settings 1005, 1006
  SDOs for startup 1006
  startup checking 1001
  startup parameters 1006
  status 1016
  timeouts 1001
EtherCAT Slave 1001
EtherCAT Slave 1005
EtherCAT Slave 1005
EtherCAT Slave 1006
EtherCAT Slave parameters 1011

event 858
event for triggered tasks 858
event logging 951
event task 858
event-time 984
everyone 186
Everyone 401
execute command 891
execute program 954
execute ST-code 891
execution order 665
execution order in CFC 273, 274, 275, 277
EXIT 379, 851
exit action 814, 816
EXIT instruction 842, 851
exit method 88
Exit method 1099, 1118
exlclude from build 605
EXP 1194
ExpandFully 1108
expandfully attribute 1103
expert mode 997, 1001
expert process data 1005
expert settings 1005, 1006
expert settings for adding a library 421
explicit folding 476
exponential function 1194
exponentiation 1197
export 435
export before compile 605
export file for GVL 100
export log entries 951
export object 435
export of loggings 951
export only text differences 567
export PLCOpenXML 439
export text list 105
export textlists 567
export-file 435, 437
expression 753, 939
Expression
  Valuation 841
Expression 841
EXPT 1197
EXTENDS 80, 85
extensible functions 73, 1097
extension of function blocks 80, 81, 82
external data base 243
external data source for visualization 199
external event 858
external file
  add 102
  properties 102
external file 102
external file properties 605
external function 98
external function block 98
external implementation 605
external library 146, 262
external method 98
external variables 1089
extract archive 369
Extras
  Show grid 283
F1 153
factory settings 980
FALSE 1211
father application 115, 119
fb_exit 1118
FB_exit method 88
FB_Exit Method 1099
FB_init 1106
FB_init method 88, 1097
FB_reinit method 88, 1097
FBD 755
FBD commands 774

FBD editor
  assignment 326
  cursor positions 772
FBD editor 754
FBD editor 762
FBD editor commands 324
FBD editor options 494
FBD view 346
FBD/LD
  branch 344, 345
FBD/LD editor
  copy element 762
  insert element 762
  insert positions 762
FBD/LD/IL
  box 328, 331
FBD/LD/IL editor
  insert negated contact parallel below 338
  insert reset coil 335
  insert set coil 335
  negated contact 338
FBD/LD/IL Editor
  breakpoint positions 784
  monitoring 784
  online 784
  tooltip 762
FBD/LD/IL Editor 754
FBD/LD/IL editor commands 324
FBD/LD/IL menu 774
FBD/LD/IL/Editor commands
  insert coil 335
feature set 60
feature sets 497
features 497
file as text 564
file close 365
file menu 179
file merge 375
file new 355
file open 358
file properties 102, 388
file save 366
file saveas 367
file system objects 966
file transfer between host and controller 950
file transfer to/from controller 950
file type 358
files 950
fill color 891
fillattributes 891
Filter by user 558, 560
Find 348
Find & Replace 348
find definition 257
Find Next 352
Find Next (Selected) 352
find object 602
Find Previous 352
Find Previous (Selected) 353
find sourcecode position 257
fix address 956
fix target device address 742
fix version 394
fixed size for operands 494
flag options for sfc 504, 605
flags 504, 605
flags in SFC 824
float 628
floating view 628
floating window 628
FMMU 1005
folded line background 476
folded line foreground 476
folder 438
folding 476
font 194, 476, 486

font flags 891
font height 891
font in visualization elements 891
font name 891
font settings for printout 521
font variables for visualization elements 891
footer 521
FOR 848
FOR instruction 842
FOR loop 848
force
  watch all forces 626, 941
force 941
force I/O values 954
force values 459
force values all applications 459
force variables for IO mapping 954, 956
forcelist 464
forcing 463, 464
foreground color for text items 486
forename 301
forename device 301
format
  conversion 487
format 487
formatting of text 105
formatting strings 194
frame 193, 208, 891, 926
frame color 891
frame commands 470
Frame commands 603
frame configuration 590, 891
frame properties of visualization elements 891
frame reference 891
frame selection 590, 891
frameattributes 891
framevisualization 590
free drive 1042
freewheeling task 858
freeze 1028, 1032
full screen 621
full text search in online help 407
function
  add 73
  additional outuputs 1087
  PARAMS OF <data type> 73
function 73
Function
  Address Operator ADR 1143, 1157
function block
  breakpoints 251
  call 78
  extension 80
  implementing interfaces 81
  initialization 1112
  method invocation 82
  methods 88
  no virtual actions 1120
  properties 88
  property 91
  tooltip on parameters in FBD/LD 762
function block 1089
function block call
  ST 845
function block call in ST 845
function block diagram 346
Function Block Diagram 755
function block download 1099
function block in FBD/LD/IL 331, 778
function block initialization 1097, 1106
function block instance 78
function block name 1073
function call 73
function call in IL 766
function in FBD/LD/IL 331, 778
function name 1073

function pointer 605, 1143, 1157, 1181, 1207
FUNCTION_BLOCK 80
functionality 60
g_CAMManager 650
G1 686
G2 688
G20 702
G3 688
G36 704
G37 704
G4 691
G40 705
G41 705
G42 705
G50 705
G51 705
G52 705
G53 709
G54 709
G55 709
G56 709
G6 696
G60 705
G61 705
G75 711
G90 712
G91 712
G98 712
G99 712
gap 1028
gateway
  add 313
  connect to local gateway 315
  edit 323
  select 944
gateway 153
gateway 944
gateway address 944
gateway driver 944
gateway service 153
G-code
  3D mode 700
  change variable values 704
  coordinate shift 709
  delay time 691
  ellipse 698
  G1 686
  G10 693, 700
  G15 700
  G16 700
  G17 700
  G18 700
  G19 700
  G2 688
  G20 702
  G3 688
  G36 704
  G37 704
  G4 691
  G40 705
  G41 705
  G42 705
  G5 693
  G50 705
  G51 705
  G52 705
  G53 709
  G54 709
  G55 709
  G56 709
  G6 696
  G60 705
  G61 705
  G75 711
  G8 698
  G9 698

G90 712
G91 712
G98 712
G99 712
H 714
H-function 714
K 702
L 702
line 686
M 716
M-function 716
modes 712
parable 696
positioning 684
preprocessor 705
prevent processing 711
R 718
spline 693
switch point 714
usage of variables 718
G-code 679
GE 1179
GE operator in IL 759
generate code 261
Generate EtherCAT xml 299
generate force variables for IO mapping 954, 956
generate runtime system files 262
Generate SercosIII SCI xml 299
generic device configuration 500
generic device configuration view 500
get database version 558
get lates version after changed in the database 508
Get Latest Version (Entire Project) 554
Get Latest Version (Selected Objects) 555
get latest version after project open 508
get latest version before build 508
getbooleanproperty 388
getcompany 388
GetLiveList 1037
getnumberproperty 388
get-property 91
GetRecipeCount 801
GetRecipeNames 801
GetRecipeValues 801
gettextproperty 388
gettitle 388
getversion 388
getversionproperty 388
global group properties 1032
global network variable list 604
global network variables list 100, 131
global scope operator 1209
global text list 567
global variable list
qualified only 1125
global variable list 98
global variable list operator 1209
global variables 1088
global variables list 131
global_init_slot 1109
GlobalImagePool 112, 408
globaltextlist 567
GlobalTextList 105
GlobStartAddr 1005
GNVL 100, 131
go to definition 257, 258, 596
go to line... 564
go to matching bracket 565
go to source position 428, 599
grant permission 966
granted rights 586
greater than 1178
greater than or equal to 1179
grid
visualization 510
grid in CFC 268

group
  user management 401
group 401
group 589
group 889
group 963
group by category 409, 421
group Everyone 186, 401
group management 394
group Owner 186, 401
group properties 401
group properties DP slave 1032
group properties for DP slaves 1028
grouping elements 889
groups
  user management 394
groups 1028
groups 1032
groups dialog 401
gs?-file 1028, 1032, 1035
gsd-file 115, 308
GT 1178
GT operator in IL 759
guard time 980
GVL
  initialization 98
GVL 98
GVL 131
GVL 604
gvl file 100, 604
halted on bp 174
handling of mechanical brakes 1050
hanging coil in LD 779
hardware device 115
hardware scan 115, 301
hastype 1128
hasvalue 1128
header 521
heading level 521
hearbeat 980
hearbeat producer time 980
heartbeat consumer 980
height of CFC 268
height of SFC steps 501
heigth of operand comment field 494
heigth of operand field 494
heigth of symbol comment field 494
Help 153
Help commands 406
help menu 179
help text 533
hexadecimal 469
H-function 714
hidden libraries 146
hidden library 426
hide 146
Hide 1110
hide a view 628
hide a window 628
hide library objects 146
hide system libraries 497
Hide_all_locals 1111
hierarchy in device tree 115
hierarchy of applications 119
highest station address 1028
highlight current line 476
History 558
History of entire project 560
HMI 188, 206
homepage 473, 622
homing 1050
horizontal tab group 630
host 604
Host
  ENI Server 563
Host 549, 553

host files 950
hotkeys configuration 592
hotkeys for visualization 891
htm-file for Web-Visualization 202
hungarian style 1073
I 1220
I/O configuration
  local 1095
I/O configuration 1095
I/O handling 954
I/O mapping
  fix address 956
  generate force variables for IO mapping 956
  modify address 956
I/O mapping 956
I/O mapping 1037
I/O Mapping for EtherCAT 1000, 1011
I/O Mapping in CANbus 992
I/O Mapping of a CANopen module 993
I/O mapping of profibus devices 1037
I/O mapping of profibus modules 1037
I/O values
  force 954, 956
I/O values 954
icon for box in FBD/LD/IL 494
ID for images 112
idenitification of device 301
ident number 1032
identifier
  multiple use 1071
identifier 1071
identifier 1073
identify device 301
IEC 61131-3 1089
IEC 61131-3 extension 1139
IEC 61131-3 language 497
iec addressing mode 1220
iec language 67
IEC objects for device instance 956
IEC operators 1160
IEC1131-3 1139
iec-cycle count 864
iecsfc.library 823, 824
IECVarAccess.library 241
IF 842
IF instruction 842, 846
ignore library reference 487
ignore the device 358
ignore the library 358
IIoDrvAcyclicServices.library 1230
IIoDrvBusControl.library 1038
IIoDrvBusControl.Library 1230
IIoDrvDPV1C1.library 1230
IIoDrvDPV1C1.library 1038
IIoDrvDPV1C2.library 1230
IIoDrvDPV1C2.library 1038
IIoDrvProfibus.library 1038
IIoDrvProfibus.Library 1230
IL 757
IL commands 774
IL editor 754, 766
IL editor commands 324
IL editor options 494
IL editor view 347
IL modifier 759
IL operand 766
IL operator 759, 766
IL table editor 754, 766
image 112, 208, 593, 891
image files 394, 510
image for visualization 112
image in address 997
image out address 997
image pool 112, 198, 408
image pool commands 407
image pool directory 510

imagepool 408
images in visualization 198, 510
implement interfaces 81, 433
implementing interfaces 81
IMPLEMENTS 81
implicit variables for SFC 504, 604
implicit variables in SFC 824
import 437
import before compile 604
import from DXF file 287
import group 963
import log entries 951
import object 437
import PLCopenXML 440
import replacement file 567
import telists 567
import text list 105
import user 963
import V2.3 projects 358
Import/Export Textlists 567
inching mode in SFC 824
indent width 476
index
  object dictionary 984
index 406
index of a pou 1167, 1208
INDEXOF 1167, 1181, 1207, 1208
info 1101
info.cfg-file 115
information
  device 1038
information 1043
information message 622
information on a device 969, 1050
Information on CANbus node 974, 979, 992, 993
information on EtherCAT master 1000, 1012
inheritance 80, 81, 82, 85
inheritation of rights 186
inhibit-time 984
INI 1097
init method 88, 1097
init mode 1001
init step 538, 824
init_namespace 1112
Init_On_Onlchange 1113
initial step 814
initialization
  arrays 1146
  function block 1112
  online change 1113
  pragma 1106, 1109, 1119
  string 1113
  Structures 1149
initialization 88, 1071, 1079, 1080, 1097
initialization 1106
initialization 1125
initialization at reset 456, 457
initialization of GVLs 98
Initialize_on_call 1112
inline monitoring
  CFC Editor 668
  FBD/LD Editor 784
  ST-Editor 836
inline monitoring 836
InOutput variables 1088
input address 1220
input assistance during text editing 1135
input assistant
  hide variables 1110, 1111
input assistant 533
input assistant command 533
input channel 956
input dialog in visualization 236
input in CFC 659
input in FBD/LD 334
input length 1035

input variables 1087
input variables in visualization 193
inputs of EtherCAT master 1000
inputs of EtherCAT slave 1006, 1011
inputs on visualization elements 891
INS 174
insert 295
insert action association 543
insert action association after 543
insert assignment 326
insert box 328
insert branch 344, 541
Insert Branch above 345
insert branch below 345
insert branch right 541
insert circle (clockwise) 725
insert circle (counter-clockwise) 725
insert coil 335
insert contact 335
insert contact parallel above 339
insert contact parallel below 338
insert contact right 338
insert declaration 295
insert device 296, 298
insert element in CFC 665
insert element in FBD/LD editor 762
insert elements in FBD/LD/IL 774
insert empty box 331
insert file as text... 564
insert IL line 340
insert input 334
insert jump 332, 544
insert jump after 544
insert label 334
insert line 725
insert macro 546
insert macro after 546
insert negated contact 338
insert negated contact parallel below 338
insert network 324
insert network below 325
insert reset coil 335
insert return 334
insert set coil 335
insert spline 725
insert step 538
Insert step-transition 539
insert step-transition after 540
insert visualization element 885
insert with arguments 533
install device 308, 497
install element 415
install library 146, 409, 415
install visual element repository 415
Installation and Profile Manager 65
Installation commands 414
installation of libraries 408
installed elements 415
instance 78
instance name 194, 1124
instance of device 956
instance of EtherCAT master 1000, 1013
instance of EtherCAT slave 1011, 1016
instance path
  breakpoint definition 251
instance path 601
Instance-path 1113
instruction 757
Instruction 842
instruction list 757, 766
instruction list view 347
INT 1137
INT_TO_BOOL 1185
integer data types 1137
integrated visualization 203
interface

visualization 193
interface 433
Interface
add 93
extend 93
implementing 81
Interface 88
Interface 93
interface editor 193, 926
Interface Editor 591
interface in FBD/LD/IL 331
interface of a visualization 193
interfaces 80
Interfaces for Profibus DP master and slave 1038
interior rotation 890
internal library 146
international settings 484
interrupts 1005
interval for triggered tasks 858
invisibility of visualization elements 890
invisible library objects 146
IoDrvCifXProfibus.library 1037
IoDrvEtherCAT 1013
IODrvEtherCAT.library 1018
IP address 301
IP-address 944
IPM 65
isotropic 890
isotropic visualization elements 890
item background 486
item font style 486
item foreground 486
jitter 864
JMP 851
JMP instruction 851
JMP operator in IL 759
jump 332, 544, 851
jump in CFC 659
jump in IL 759, 766
jump in SFC 544, 816
jump label 851
jump to line if error 988, 1006
jumps 702
K 702
Keep checked out 556
keep current values 954
keep tabs 476
key
project properties 388
key configuration 890
key event in visualization 229
Keyboard
Load from file 518
Save to file 518
Keyboard 169
Keyboard 518
keyboard configuration 890
keyboard operation of visualization 229
keyboard usage
visuallization editor 887
keyboard usage in visualization 229, 595
keyboard usage in visualization editor 887
keypad 604, 890
keys 518
keywords
writing 485
keywords 1083
L 702
L_AddLog 23
L_DM_ParameterAccess 24
L_IO1000_EPMS640_RS232Read 48
L_IO1000_EPMS640_RS232Write 52
L_SMC_AbortTrigger 32
L_SMC_AbortTrigger (for EtherCAT) 32
L_SMC_AbortTrigger_FastIO 43
L_SMC_FastOutput 44

L_SMC_FreeEncoder 46
L_SMC_ParameterAccess_FB 24
L_SMC_ReadAxisError 36
L_SMC_ReadParameter 33
L_SMC_ReadParameter (for EtherCAT) 33
L_SMC_SetTorqueLimit 26
L_SMC_SetTorqueLimit (for EtherCAT) 26
L_SMC_TouchProbe 28
L_SMC_TouchProbe (for EtherCAT) 28
L_SMC_TouchProbe_FastIO 38
L_SMC_WriteParameter 34
L_SMC_WriteParameter (for EtherCAT) 34
label
  branch 816
label 816
label for SFC branch 541
label for source control project 562
label in CFC 659
label in FBD/LD/IL 334, 778
label in instruction list 757
Labels only 558, 560
ladder diagram 755
ladder editor view 346, 347
language
  change by visualization input 890
  menu title 514
  switch in visualization 105
language 566
language 566
language 567
language 1070
language file 105
language files 510
language in visualization 194, 510
language model 174
language selection for online help 153, 484
language selection for user interface 484
last cycle time 864
last jitter 864
lastmessage 1013
latching 1050
latest version 554, 555
layout for printout 521
layout options for SFC editor 501
LD
  coil 782
  contact 781
  insert coil 335
  insert negated contact parallel below 338
  insert set coil 335
  negated contact 338
LD 755
LD commands 774
LD editor
  assignment 326
  cursor positions 772
  insert coil 335
  insert negated contact parallel below 338
  insert set coil 335
  negated contact 338
  paste 339
LD editor 340
LD editor 754
LD editor 762
LD editor commands 324
LD editor options 494
LD operator in IL 759
LE 1179
LE operator in IL 759
length of user parameters 1032, 1035
less than or equal to 1179
L-force Controller 3200C 1224
libcat.xml 146
libraries
  manage 791
  options 487

libraries for visualization 199
libraries menu 794
library
add 421
CAA_FILE.library 1232
container 146
conversion options 487
create and install 375
dependencies 409
details 409
device-specific 421
encoded format 146
external 146, 262
general information 146
hidden 146, 426
installation 409
internal 146
locations 409
mapping 487
namespace 426
properties 426
referenced 146
Standard.Library 1230
target-specific 421
version handling 426
Library
Bus specific Libraries 1230
CAA Libraries 1232
CAA_AsyncManager.library 1232
CAA_CallBack.Library 1232
CAA_CanL2.library 1232
CAA_CanOpenMan.library 1232
CAA_CanOpenStack.library 1232
CAA_CiA405.lib 1230
CAA_CiA405.library 1232
CAA_DTUtil.library 1232
CAA_Mathematics.library 1232
CAA_MemBlockMan.library 1232
CAA_Memory.library 1232
CAA_NetBaseServices.library 1233
CAA_RealTimeClock.library 1233
CAA_ResMan.library 1233
CAA_SegBufferMan.library 1233
CAA_SerialCom.library 1233
CAA_Settings.library 1233
CAA_Tick.library 1233
CAA_Timer.library 1233
CAA_Types.library 1233
CoDeSys V3.x System Libraries 1231
IIoDrvAcyclicServices.library 1230
IIoDrvBusControl.Library 1230
IIoDrvDPV1C1.library 1230
IIoDrvDPV1C2.library 1230
IIoDrvProfibus.Library 1230
Runtime Error Codes 1230
SM3_Basic.Library 1230
SM3_CNC.library 1230
SSysComAsync.library 1230
SysCom.library 1230
SysCPUHandling 1230
SysDir.library 1230
SysDirAsync.library 1230
SysEvent.library 1230
SysFile.library 1231
SysFileAsync.library 1231
SysInt.library 1231
SysMem.library 1231
SysPci.library 1231
SysPort.library 1231
SysPortAsync.library 1231
SysProcess.library 1231
SysSem.library 1231
SysSemProcess.library 1232
SysShm.library 1232
SysSocket.library 1232
SysSocketAsync.library 1232

SysTask.library 1232
System Libraries 1230
SysTime.library 1232
SysTimer.library 1232
SysTypes.library 1232
Util.library 1230
library category 146, 388, 409, 421
library database 409
library documentation 146, 791
library DUT name 1073
library encoding 375
library files 367, 375
library for SFC 824
library handling 497
library management 146, 497
library manager
commands 421
library manager 146
library manager 421
library manager editor 791
library manager object plug-in 408
library name 791
library namespace 791, 1210
library path 146
library placeholder 146, 421, 426
library POU name 1073
library reference 487
library reference conversion 358
library references 426
library repository
add 409
automatic installation 375
availability 497
edit 409
remove 409
library repository 146
library repository 409
library version 146, 386, 421
life time factor 980
LIMIT 1176
limiting 1176
line
simple line 890
line 208, 564, 686
line 890
line color 476
line numbering 476
line width in visualization elements 890
linear drive 1043
linedirection 890
linestyle 890
linewidth 890
link always 604
link to file 604
linkalways 1114
LINT 1137
list components 476
list components after typing a dot 485
List Components for text input 105
list components functionality 476, 485
List Components functionality 510, 1110, 1111
List Components Functionality 1135
list components immediately when typing 485
list identifier 604
ln 174
LN 1193
load a recipe from a file 890
load and save options 473
load from disk 963, 966
load last loaded project 473
load program to ASCII file 286
load recipe 797, 890
load trace 580
load trace configuration 580
LoadAndWriteRecipe 801
LoadFromAndWriteRecipe 801

LoadRecipe 801
local address 124
local variables 1087
localization 1070
localized text for menu title 514
location
  breakpoint 251
  project 355
location 604
Location 549, 553
location of devices 308
location of POU 601
location of variables 258, 596
location of visual element repositories 415
location of visual element repository 415
lock 1032
Log
  clear list 951
  export and import 951
Log 951
LOG 1193
log file export 951
log file import 951
log of runtime 951
logarithm 1193
logger 951
logging offline 951
login 445, 451
Login 563
login configuration of a datasource 742
login dialog 139, 236
login to Application 445
Login to Source Control 563
login with download 139
login with online change 139
login without any change 139
Logoff 585
logoff dialog 585
Logon dialog 583
logon name 401
logout 448
logout from application 448
look variables for visualization elements 890
lower than 1178
lowercase 565
LREAL 1138
LREAL constants 1214
LREAL_TO conversions 1187
LT 1178
LT operator in IL 759
LTIME 1140
LWORD 1137
M 1219, 1220
m_references 890
m4 file for external library 262
MAC address 301
MAC address in EtherCAT network 997
macro in SFC 546, 547, 816
mainnet 123, 124
make lowercase 565
make uppercase 565
mapping
  automatic 979
  DP module 1037
mapping 1082
mapping device I/Os 1000, 1011
mapping for device conversion 358
mapping for library reference conversion 358
mapping of device I/Os 956
margin options 476
margins of printout 521
marker of a subnetwork 779
master 997
master cycle 997
Match case 348
Match whole word 348

matching brackets 476
MAX 1175
max. cycle time 864
max. jitter 864
max.number of tasks 857
maximum function 1175
maximum runtime of SFC step 814
MC_CAM_REF 650, 651
MC_CAMin 642
MC_CAMTableSelect 642
memberships 401
memory allocation 1198, 1200
memory area 1220
memory location address 1219, 1220
Menu
  standard 179
Menu 169
Menu 179
Menu 514
menu bar 174, 179
menu title 514
merge project 66, 375
message pragmas 1101
message view 428
message view plug-in 622
message window
  build 260
message window 258
message window 596
messages
  go to source position 428
  next message 429
  previous message 429
messages 622
messages view 174, 622, 1101
messages window
  pragma 1101
messages window 174
messages window 1101
meter element 211
meter element configuration 211
method
  call after initialization 1106
method 1097, 1099
Method
  additional outputs 73, 82
  exit 88
  init 88
Method 88
method call 82
method call in application stop 88
method call in IL 766
method invocation 82
method prototypes 93
methods 91
M-function 716
Migration of 22
MIN 1176
min. cycle time 864
min. jitter 864
minimum function 1176
minimum runtime of SFC step 814
MOD 1164
modes 712
modifier in instruction list 759
modify address 956
modulo division 1164
monitor 864
monitoring
  CFC Editor 668
  FBD/LD Editor 784
  ST-Editor 836
  tooltip 753
  watch list 941
monitoring 753
monitoring 1115

Monitoring 143
monitoring display mode 1107
monitoring format 469
monitoring options 476
motion axis control
  example 1058
motion control
  example 1053, 1055
mouse actions 476, 890
mouse scrolling 576, 580
mouse zooming 576
MOVE 1165
move down 273
move down declaration 295
move down POU in task 858
move element in CFC 665
move program 284
move the working sheet origin relatively 268
move up 273
move up declaration 295
move up POU in task 858
movement of visualization elements 890
MUL 1162
MUL operator in IL 759
multi channel 578
multi-device 115
multi-language support 105
multiple download 454
Multiple Download... 454
multiplexer operator 1177
multiplication 1162
multi-resource 115
multitasking 122
multi-user operation 243
MUX 1177
N modifier 759
name
  project 355
Name filter 348
name of device 969
namespace 388, 1071, 1209
namespace for library 146, 426, 791, 1210
namespace for strings 1112
naming convention 1073
navigatorcontrol 438
NE 1180
NE operator in IL 759
negate 269
negated contact 338
negated operation 759
negation 269
negation inFBD/LD 340
nested applications 119
nested comments 852
Net 972
net number 974
network
  subnetwork 779
network 324, 325, 776
network address 126
network comment
  options 494
network comment 776
network data exchange 130, 131
network label 334
network name EtherCAT 997
network number for CANbus 972
network path to controller 944
network properties 131, 604
network scan 944
network scan at login 445
network settings 131, 604
network title 494
Network topology 123
network type 604
network variables

export file 100, 131
network variables 100
network variables 131
network variables 604
network variables list 604
NetX
Profibus
check bus participants 1037
Profibus 1037
NetX Com Channel 1037
NetX configuration 1037
new breakpoint 251, 599
new horizontal tab group 630
new item in menu 514
new item in toolbar 513
new project 355, 553
new separator 514
new toolbar 513
new vertical tab group 630
newest version alwalys 426
news page 473
next bookmark 250
next editor 628
next message 429
next pane 629
NIC settings EtherCAT 997
no binding 604
No Binding 549
no initialization 980
No_check 1117
No_Copy 1117
No_virtual_actions 1120
node address 126, 312, 944
nodeguarding in CANbus 980
node-id 980
No-exit 1118
noinit 1119
nominate device 301
none 272
non-periodic 642
NOT 1169
not equal to 1180
number constants 1214
number of objects 388
number of syncs 984
number of undos 476
numberactiveslaves 1013
numpad 604, 890
NVL 604
object
add 431
object 68
object 1006h 977
object dictionary 984, 988
object history 558
object name 604
object orientated programming 60, 61, 80, 93
object orientation 60
object oriented programming 81, 82, 85, 88
object properties 68, 604
object type
statistics 388
object type 604
object types
availability 497
object types 431
Object types 348
object types availability 497
object types in project 60
objects
number of 388
objects commands 438
objects on device 966
obsolete pragma 1122
offline logging 951
offset 642

old password 401
ondialogclosed 890
one file per subtree 435
online
  Download Application 450
  login to Application 445
  logout from application 448
  multiple download 454
  secure online mode 443
  simulate device 464
  Start Application 453
  Stop Application 453
  write values 457
online change
  initialization 1113
  remanent variables 1090
online change 445, 451
Online Change 139
online commands 443
online help
  language 153, 484
Online Help
  index 406
Online Help 406
online menu 179, 443
online mode
  ST-Editor 836
  user interface 173
online mode 173
online status of application 174
online user management 963
online view 753, 836
onlinee
  force values 459
only look at recent libraries 415
onmousedown 890
onmouseenter 890
onmouseleave 890
onmousemove 890
onmouseup 890
open editor 624
Open From Source Control 553
open in editor 602
open POU
  from task editor 858
open project
  version handling 394
open project 358, 379
open targetvisu 604
Open view 169
operand
  options 494
operand in IL 766
operand in parenthesis in IL 759
operands 1137
operational mode 1001
operator
  assignment 844
  in pragmas 1128
operator 1160
operator in FBD/LD/IL 331
operator in IL 766
operator in instruction list 759
opt.keyb-file 518
opt.tbar-file 513
optional device 980
optional device default 977
options
  CFC 492
  CoDeSys 2.3 converter 490
  CoDeSys V2.3 converter 490
  declaration editor 493
  device editor 500
  features 497
  international settings 484
  libraries 487

load and save 473
SFC 501
smartcoding 485
syntax highlighting 486
text editor 476
visualizations 510
options 169, 472
Options
FBD LD and IL 494
Langage FBD
Langage LD et IL 494
options CFC editor 492
Options dialog 472
options for SFC 504
OR 1168
OR operator in IL 759
order 889
order by data flow 274
order by topology 275
order number of device 969
order of visualization elements 594, 889
ORN 1168, 1209
out of data 576, 580
output address 1220
output channel 956
output in CFC 659
output in FBD/LD/IL 783
output length 1035
output variables
additional 73
output variables 73
output variables 1087
outputs of EtherCAT master 1000
outputs of EtherCAT slave 1006, 1011
overwrite 366
overwrite mode 564
OVR 174
owner 186
Owner 401
pack variables 604
pack_mode 1123
page layout 521
page setup 521
pager setup 394
paper format 521
pApplicationInfo 88
parabel 696
parabolic 1043
parallel branch 541, 816
parallel connection 339
parallel in sfc 540
parallel SDO action limit 977
parameter dialog in task configuration 863
parameter editor for CNC 738
parameter update for boxes 328
parameter update in FBD/LD/IL 345
parameters
constant inputs in CFC 277
parameters of a device 948
Parameterstringof 1124
PARAMS OF 73, 1097
password
user 401
password 152
password 583
password 963
Password
ENI Server 563
password for project 394
Paste 293
paste above 340
paste after 339
paste below 339
paste contacts 339, 340
paste device object 115
paste element in SFC 812

paste in IL 766
paste right 339
path for image files 510
path for language files 510
path preprocessing online 1067
pause in SFC 824
PDO
  add 984
  enable 983
  mapping 984, 988
  properties 984
PDO Assignment download 1005
PDO Configuration download 1005
PDO mapping 983, 984, 988
PDO properties 984
PDO sequence 988
pending checkins 624
periodic 642
periodic CAM 642
periodic transition 604
permissions 586
permissions on device 966
PERSISTENT 1090
persistent variables 102
Ph.Start Address 1005
pins in CFC 278, 279
placeholder
  visualization 193
placeholder 193
placeholder 194
placeholder 926
placeholder for library 146, 421, 426
placeholder library for visualization 199
placeholders for page setup 521
placeholders in menus 514
placeholders in toolbars 513
plc configuration 115, 123
PLC service 153
PLC settings 954
Plc-Logic 115
PLCopenXML
  export 439
  import 440
PLCopenXML 439
plug device 296, 299
plug-in 65
Pointer 1143, 1157
pointer check function 1143, 1157
POINTER TO 1143, 1157
poll timeout 1028
polygon 208, 890
polyline 208, 890
port 604
Port 549, 553
position
  breakpoint 251
position 889
position 890
position of cursor 174
position of visualization elements 890
Positioning 684
positioning of views 169
postfix in IL 766
POU
  assign to task 858
  browse for 257
  open from task editor 858
POU 68, 71
POU execution order in tasks 858
POU name 1073
POU properties 68
POU Windows 174
POUs for implicit checks 114
POUs tree
  build 264
POUs view 68

POUs View 602
POUs window 602
power rail 755
pragma
attributes 1103, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1117, 1118, 1119, 1120, 1123, 1124, 1125
conditional pragmas 1128
define 1128
defined 1128
displaymode 1107
error 1101
hastype 1128
hasvalue 1128
info 1101
noinit 1119
obsolete 1122
symbol configuration 241
symbolconfiguration 1127
text 1101
undefine 1128
warning 1101
xref 1128
pragma 604
pragma for datatypes 1122
pragma instruction
message 1101
pragma operator 1128
precompile information 394
precompile message 622
precompile process 139
preferences 470
prefix in IL 766
preoperational mode 1001
prepare value dialog 459
prepared value 457, 753, 836
preprocessor 705
prevent processing 711
previous bookmark 250
previous editor 628
previous message 429
previous pane 629
print 248, 521, 531
print commands 519
print layout 521
print margins 521
print visualization 890
printout 520
priority 858
process data 1006
process data of EtherCAT slave 1005
processing order in CFC 274, 275, 277
Product ID
check EtherCAT configuration 1001
professional feature set 60
professional features set 497
profibus
DP master 1028
DP parameters 1035
DP slave 1032
DP-Module 1035
I/O mapping 1037
profibus 1037
profibus device
information dialog 1038
Profibus DP Configuration 1036
profibus I/O mapping 1037
profibus I/O mapping master 1037
profibus master
NetX 1037
profibus master 1037
profibus master 1038
profibus module configuration 1036
Profibus Slave 1038
profile 65
profile for visualization 394
profile for visualization project 200

profile selection 415
PROFINET IO
  sync config 306
  synchronize configuration 306
PROGRAM 71
program call 71
program call in IL 766
program in FBD/LD/IL 331, 778
program loaded 174
program modified 174
program name 1073
program status 174
program unchanged 174
programming language
  CFC 658
programming language 67
programming language 755
programming objects 60
project
  automatically at startup 473
  close 365
  CoDeSys V2.3 66
  compare 379
  download 452
  example 153
  location 355
  merge 375
  new 355
  new project from database 553
  open 358
  recent projects 379
  save 366
  save as 367, 375
  standard project wizard 355
  start page 622
  statistics 388
  summary 388
project 66
project 379
project archive 369, 372
project compilation 394
project compression 473
project conversion 375
project documentation 248, 520
project encryption 394
project environment 358, 386
project file only 394
project files 367
Project History 560
project information
  namespace 791
project information 66, 146, 388
project menu 179
project name 355
project organization 60
project security 394
project settings
  compile options 394
  page setup 394
  replace constants 394
  security 394
  source download 394
  users and groups 394
  version handling 394
  visualization 394
  visualization profile 394
project settings 66
project settings 394
project settings 401
project template 66, 355
project templates 355
project upload 452
project version 358
project version handling 394
prompt network path at login 944
properties

access control 604
bitmap 604
boot application settings 604
breakpoint 599
build
compiler defines 604
enable system call 604
exclude from build 604
external implementation 604
link always 604
build 604
CAMEditor 604
CNC Editor 604
CNC Settings 604
common 604
dynamic memory settings 604
external file 102
file 102
functions 388
key 388
link to file 604
network properties 604
network settings 604
object 68, 604
POU 604
project file 388
SFC settings 604
source control 604
task 858
visualization 604
visualization element 890
properties 604
properties of a library 426
properties of a task configuration 857
properties of a visualization element 890
properties of CAM elements 654
properties of CNC object 730
properties of path element 724
properties of SFC elements 623, 814
properties of visualization elements 625
property
method 91
monitoring 1115
property 91
property width 501
propeties for SFC steps 623
provider data items 742
pTaskInfo 88
publish all IEC symbols to that project... 426
Q 1220
Qualified_Only 1125
qualifier width 501
qualifiers for SFC actions 823
R 272
R operator 844
R operator in IL 759
ramp_in 642
ramp_in_neg 642
ramp_in_pos 642
read CAM data from ASCII table 266
read CAM online file 266
read EtherCAT parameter 1018, 1021, 1022
read recipe 797, 890
ReadAndSaveRecipeAs 801
read-only projects 473
ReadRecipe 801
REAL 1138
REAL constants 1214
REAL to INT conversion 1190
REAL_TO conversions 1187
reboot
remanent variables 1090
rebuild 261
receive PDO mapping 984
recent projects 379
recipe

add 532
remove 532
recipe 797
recipe 890
recipe commands 531
recipe definition
add recipe 532
remove recipes 532
recipe definition 797
recipe management 890
recipe manager 531
recipe manager commands 890
recipe name 797
recipe_management.library 797
recipemancommands 797
rectangle 208, 890
redo 582
REF 1141, 1155
reference
__ISVALIDREF 1141, 1155, 1200
check operator 1141, 1155, 1200
cross reference 258, 596
data type 1141, 1155
initialization 1141, 1155
visualization 193
reference 193
reference 1141
reference 1155
referenced libraries 146
references in visualization frame elements 890
references of libraries 426
Reflection 1125
refresh data items 742
refresh list of applications 949
refresh status 562, 624
register image files without path 510
register text list files without path 510
regular expressions 348
reinit drive 1050
reinit method 88, 1097
reinitialization 88
re-initialization 456
relative 642
relative address 126
relative addressing 124
relative movement of visualization elements 890
relative path 604
release and restore forced values 459
release forcelist 464
release forcing 459
release the force without modifying the value 459
ReloadRecipe 801
remanent variables
remove from device 470
remanent variables 102
remanent variables 1090
remember declaration editor format 493
Remember my credentials 563
remember this mapping... 358
remove applications from PLC 949
remove language 567
remove library repository 409
remove recipes 532, 797
remove unused pins 279
Remove Unused Text Ids 573
remove variable from recipe 797
rename object 431
render pixels of this color transparently 604
renumber CNC program 286
Repair 153
REPEAT 850
REPEAT instruction 842
REPEAT loop 850
Replace 348, 352
replace constants 394
replacement file 567

repository
visual elements 415
repository 146
repository for devices 308
repository for libraries 409
repository locations dialog 415
reset 272, 343, 844, 1090
reset coil 335, 783
reset cold 457, 1090
reset in SFC 824
reset origin 457, 1090
reset origin device 470
reset output 272
reset pins 278
reset version 558
reset warm 456, 1090
reset window layout 630
resource 119
resource objects 60
restart EtherCAT master 1013
restore forced values 459
RET network 776, 784
RET operator in IL 759
RETAIN 1090
retain variables 1090
retry limit 1028
return 334
RETURN 842, 846
return in CFC 659
RETURN in FBD/LD/IL 779
reverse direction 286
revision control 243
right
deny 586
grant 586
rights 186
rights for online device actions 966
ROL 1172
ROR 1173
rotate program 285
rotation 890, 1172, 1173
rotation of visualization elements 890
rotatory drive 1043
rounded rectangle 208, 890
Routing 124
RUN 174
run to cursor 256
Runtime Error Codes 1230
runtime logger 951
runtime objects 966
runtime system 206
runtime system files 950
S 272
S operator 844
S operator in IL 759
safe operational mode 1001
same as Microsoft Windows 484
save a recipe in a file 890
save archive 372
save as 366
save parse trees to file 394
save project 366
Save Project And Save Into Library Repository 375
save project as 367
save project as compiled library 375
save recipe 797, 890
save to disk 963, 966
save trace 580
save trace configuration 580
SaveRecipe 801
scale program 285
scaling 642
scaling of visualization elements 890
scan applications on PLC 949
scan devices
PROFINET IO 306

scan for devices 301
scan hardware 301
scan network 944
scan of hardware 115
Scheme 348
scope operator 1209, 1210
scopes in device tree 115
scroll path view 283
SDO
  create all 980
  new 988
SDO 988
SDO 1006
SDO Access timeout 1001
SDO action limit 977
SDO parameters 1006
search 407
Search 348
search for target device 742
search in online help 407
Search scheme 348
Search string 348
Search up 348
search-type 742
secure online mode 443
security 152, 394
SEL 1174
select 725
select a target device 358
Select all 294, 589
select element in CFC 665
select element in SFC 812
select image dialog 112
select item from object dictionary 984, 988, 1006
select network by MAC 997
select network by name 997
Select none 589
select online state 433
select paste position 115
select source device 490
select target device 358, 490
select the network path to the controller 944
select to matching bracket 565
select visualization elements 887
selecting elements in FBD/LD/IL editor 762
selection color 476
selection of visualization elements 229, 595
Selection only 348
selection operator 1174
selector 659
send archive 372
Send one to back 594
send PDO mapping 988
send to back 273
Send to back 594
send to front 273
sender 604
sensitivity of watchdog 858
separator 514
separator in menus 514
sequence numbers in CFC 273
Sequential Function Chart 810
service data object 988
Service Data Object 1006
service data object dialog 1006
set 272, 343, 844
Set Active Application 437
set active path 324, 944
set all outputs to default 954
set coil 335, 783
set defaults 604
set execution order 277
set next statement 256
set output 272
set output connection 343
set target library 487

set variables 291
set/reset in FBD/LD 343
Set/Reset in IL 783
Set/Reset outputs in FBD/LD/IL 783
set-property 91
SetRecipeManager 801
SetRecipeValues 801
settings dialog 401
settings for network variables 604
SFC
  cursor positions 811
  editing 812
  element properties 623
  flags 824
  implicit variables 824
  online mode 834
  qualifier 823
  step attributes 814
SFC 810
SFC action 543
SFC attribute 501
SFC editor commands 538
SFC editor options 501
sfc element properties 623
SFC Element Properties 814
SFC elements 816
SFC flags
  access 824
SFC flags 504, 604
SFC flags 824
SFC library 504, 604, 823, 824
SFC macro 546, 547
SFC options 501, 504, 604
SFC properties 504
SFC settings 504, 604
sfc settings properties 604
SFC step 538
SFC toolbox 816
SFCActionControl 823
SFCCurrentStep 824
SFCEnableLimit 824
SFCErrorAnalyzationTable 824
SFCErrorPOU 824
SFCErrorStep 824
sfciec.library 504, 604
SFCInit 824
SFCPause 824
SFCReset 824
SFCTip 824
SFCTipMode 824
SFCTrans 824
shift 1170, 1171, 1172, 1173
Shift-Click 476
SHL 1170
shortcut keys 518
shortcut mode 1081
show current statement 256
show cursor 217
show differences 624
Show Differences 556, 557
show documentation
  data source 742
show documentation 533
show empty environment 473
show end points 283
show frame 890
show generic device configuration views 500, 948, 1036
show grid 283
show interpolation points 282
show network comment 494
show network title 494
show 'new project' dialog box 473
show next lodation 596
show 'open project' dialog box 473
show operand comment 494

show preprocessed path 282
show previous locatoin 596
show step time 501
show symbol address 494
show symbol comment 494
showcursor 890
SHR 1171
sigmoidal 1043
signature mismatches 949
simple line 890
simulate device 464
simulation 115, 144, 464
simulation device 115
simulation mode 464
SIN 1194
sine 1194
single cycle 453
singleton event 858
SINT 1137
size 889
size of a variable 1165, 1208
size of CFC 268
size of memory for visu 604
SIZEOF 1165, 1208
slave 1001, 1032
slave interval 1028
slot 299
slot for NetX communication 1037
SM3_Basic.library 650, 1039
SM3_Basic.Library 1230
SM3_CNC.library 1230
SM3_CNC.library 1039
smart coding
  auto declare 535
smart coding 1135
smart coding commands 533
smartcoding 1135
smartcoding options 485
SMC_CAMTable 652, 653
SMC_CAMXYVA 652
softmotion
  drive configuration 1042
  drive interface 1042
  Drive Logical 1047
  example application 1052
softmotion 1039
Softmotion Device Editor
  Survey on special use cases 1050
softmotion drive
  basic 1043
  scaling/mapping 1043
softmotion drive interface 1042
sort by 602
sort order 602
source address mac 997
source control
  differences to v2.3 246
source control 243
source control 246
source control 246
source control 604
source control 624
Source Control
  Get Latest Version 554, 555
  Label 562
Source Control 549
Source Control commands 548
Source Control configuration 549
source control database connection 553
source control for a project 549
source control login 563
source control options 508
source control project label 562
source control properties 604
source control status 562, 624
source device 490

source download
  content 394
source download 394
source download 452
source download to connected device 452
source library 487
Source upload 452
sourcedownload
  destination device 394
  timing 394
special binding 604
Special binding 549
specific culture 484
specific profile 394
specific version 426
spline 693
SQRT 1192
square root 1192
ST
  assignment 844
  Comments 852
  Expression 841
  Instruction 841
ST operator in IL 759
standard commands 179
standard feature set 60
standard features set 497
standard menu 179
standard project wizard 355
standard.library 355
Standard.Library 1230
start 153
start / stop trace 575
start application 453
start page 473, 622
start trace 575
start visualization 192, 936
Start with a little project 153
startconfigwithlessdevice 1013
startup
  DP master 1028
startup behaviour 473
startup page 473
startup parameters EtherCAT 1006
startvisualization 604
state variables for visualization elements 890
static id 890
static ID 112
static image 112
static text 105, 194
static variables 1089
station address 1028, 1032
station name 301
statistics
  project 388
status
  task 864
status 974
status 979
status 992
status 993
status 1000
status 1011
status 1043
status information 174
status line 174
status of a device 1038
status of EtherCAT slave 1016
status of SFC actions 824
status of SFC steps 824
ST-Editor
  breakpoint positions 836
  JMP 851
  monitoring 836
  online 836
ST-Editor 836

step 538, 539, 540, 816
step action 816
step attributes 501, 814, 834
step height 501
step in SFC 539, 540
step into 255
step name 814, 824
step out 256
step over 255
step status 824
step suppression 283
step time 501, 824
step width 501
stepping
  run to cursor 256
  set next statement 256
  show current statement 256
  step into 255
  step out 256
  step over 255
stepping 144, 250
stepping 601
stepping in FBD/LD/IL 784
step-transition 539
step-transition after 540
STOP 174
stop application 453
stop EtherCAT bus 1013
stop on BP 864
stop on error 864
stop trace 575
stop watchdog 864
storage format 358
stretch 577
string
  instance path 1113
  namespace 1112
STRING 1138, 1141, 1155
STRING_TO conversions 1189
STRING_TO_BOOL 1185
STRUCT 1149
structure
  pack mode 1123
structure 1216
Structure of addresses 126
structure visualization 223
structured view
  data source 742
structured view 533
Structures
  Initialization 1149
Structures 1149
SUB 1162
SUB operator in IL 759
subindex
  object dictionary 984
subnet 123, 124
subnet index 124
subnet mask 301
subnetwork in FBD/LD 344
subnetwork in FBD/LD/IL 779
subnetwork marker 344, 779
Subrange types 1152
subtraction 1162
summary
  project 388
SUPER 80
suppress 1006h object 977
Survey on special use cases 1050
switch between controller modes 1050
switch FBD/LD 346, 347
switch framevisualization 590, 890
switch point 714
switchable between textual and tabular 493
switching 642
symbol 241, 1127

symbol configuration 241
symbolconfiguration 1127
symbolic values 1032, 1035
sync 1028, 1032
sync config 306
sync manager 1005
sync manager type 1005
sync unit cycle 1001
sync. COB-ID 980
Sync0 1001
synchronization in EtherCAT 1001
synchronization manager 1005
synchronize configuration of PROFINET device 306
Synchronize Contents 407
synchronous transmission 984
synchronous window length 980
syntax check 260
syntax highlighting 486
SysCom.library 1230
SysComAsync.library 1230
SysCPUHandling 1230
SysDir.library 1230
SysDirAsync.library 1230
SysEvent.library 1230
SysFile.library 1231
SysFileAsync.library 1231
SysInt.library 1231
SysMem.library 1231
SysPci.library 1231
SysPort.library 1231
SysPortAsync.library 1231
SysProcess.library 1231
syssem.library 122
SysSem.library 1231
SysSemProcess.library 1232
SysShm.library 1232
SysSocket.library 1232
SysSocketAsync.library 1232
SysTask.library 1232
system call
  build properties 604
system call 604
system events 863
System Libraries 1230
system time 194
SysTime.library 1232
SysTimer.library 1232
SysTypes.library 1232
t# 1211
tab group of windows 630
tab width 476
table 223
table editor 719
table of contents 520
table view 719
tabular declaration editor 294, 493
tabular declaration editor commands 294
tabular only 493
TAN 1195
tangent 1195
tap false 890
tap on enter if captured 890
tap variable 890
Target 115
target device 490
target ID 944
target library 487
target name 944
target search for data server 742
target type 944
target vendor 944
target version 944
target-specific library 421
targetvisu activation 604
Target-Visualization
  settings 936

Target-Visualization 188, 201
Target-Visualization 936
Target-Visualization editor 936
TargetVisualization object 936
task
  add POU 858
  configuration dialog 858
  execution order 864
  idle 864
  jitter 864
  monitor 864
  POUs execution order 858
  priority 864
  running 864
  status 864
  stop 864
  time flow 864
  trace 871
task 855
task configuration
  maximum number of tasks 857
  monitoring tasks 864
  parameter dialog 863
  properties dialog 857
  system events 863
  watchdog 858
task configuration 122, 855, 856
task editor
  online view 864
task editor 856
task for network variables 604
task name 858
task parameters 863
task priority 858
task properties 858
TaskConfig editor plug-in 856
tasks
  max.number 857
tbit 1028, 1032
TCP/IP address 553
templates
  categories 355
  project 355
templates 355
terminate 379
testkeyword 143
testkeyword1 143
text
  Intellisense 105
  visualization 887
text 105
text 194
text 1101
text area options 476
text color 486
text editor 677, 1135
text editor basics commands 564
text editor options 476
text editor smartcoding option 485
Text file for textual List Components functionality 510
text font 486
text for visualization element 887
text formatting 105, 194
text in visualization 105, 194, 890
text list 105, 510
text list commands 566
text list directory 510
text list files 510
text properties 890
text properties of visualization elements 890
text variable 194
text variables for visualization elements 890
textlist
  import/export 567
textlist 194, 566, 567
textlist files 394

texts 890
textual declaration editor 493
textual only 493
the handling 186
THIS pointer 88
TIME 1138
time attributes of SFC step 814
TIME constants 1211
Time Data Types 1138
time format 194
time of a SFC step 501
time status of SFC step 824
TIME_OF_DAY 1138
TIME_OF_DAY Constants 1138, 1213
TIME_TO conversions 1187
TIME_TO_BOOL 1185
timeout in SFC 814, 824
timing for source download 394
tip mode in SFC 824
title page 520, 521
TO_BOOL conversions 1185
TOD 1138
tod# 1213
TOD_TO conversions 1187
toggle a variable 890
toggle bookmark 249
toggle breakpoint 251
toggle network comment state 325
toggle variable 890
tool bar 174
toolbar
  activation 513
  definition 513
  delete 513
  load from file 513
  save to file 513
Toolbars 169, 513
toolbox 625, 885
ToolBox 174
toolbox for visualization 885
toolbox in CFC 659
toolbox in SFC 816
tools menu 179
tooltip 890
tooltip in FBD/LD/IL editor 762
tooltip text 105
topological order of processing 275
TQUI 1028
trace
  compress 577
  cursor 575
  default appearance 577
  download 574
  load configuration 580
  mouse scrolling 576, 580
  mouse zooming 576
  multi channel 578
  Online List 579
  out of data 576, 580
  start / stop 575
  stretch 577
trace 217, 574
trace 866
trace 890
trace commands 574, 577, 578, 580
trace configuration
  activate maximum warning 871
  appearance 871
  axes for trace display 871
  backcolor 871
  continue after trigger 871
  critical lower limit 871
  critical upper limit 871
  description of trace axis 871
  fill arrow 871
  font 871

graph color 871
graph type 871
grid 871
invert display 871
length of trace time 871
major tick distance 871
measure in every x-th cycle 871
minor tick distance 871
paint arrow 871
task for tracing 871
time span 871
trigger 871
trigger edge 871
trigger level 871
trigger position 871
variables 871
trace configuration 871
trace configuration file 580
trace editor
appearance 880
configuration of new variables 869
graph color of new variables 869
graph type of new variable 869
maximum warning of new variable 869
minimum warning of new variable 869
online mode 881
overview 866
trace editor 866
trace editor plug-in 574
trace element configuration 217
trace visualization 217
trace-file 580
transfer of files 950
transition
multiple use 816
transition 539, 540
transition 816
transition actuation 824
transition code 504, 604
transmission of network variables 604
transmission type 984
transmit on change 604
transmit on event 604
transparency of a bitmap 604
transparent 890
transparent color 890
trapezoid 1043
triggered by event 858
triggered by external event 858
TRUNC 1190
TRUNC_INT 1191
TSDR 1028, 1032
TSET 1028
TSL 1028
TTR 1028
TYPE 1139
type of device 969
type of object 431
type of visualization element 890
typed literals 1093, 1215
UDINT 1137
UDP network 604
UINT 1137
ULINT 1137
undefine 1128
Undo 581
undo checkout 557, 624
undo/redo commands 581
Undocking views 169
unforce values 463, 941
unforce values all applications 463
ungroup 590
unicode 105
uninstall device 308
uninstall library 409
uninstall visual element repository 415

Uninstallation 153
UNION 1139
unlock 1032
unresolved references 949
Update 153
update all variables in all devices 954
update anything 415
update box parameters 328
update code 415
update device 296, 299
update io while in stop 954
update of compiler version 358
update of library version 358
update of storage format 358
update of visualization profile 358
update parameters 328, 345
update rate for data server 742
update variables 956
update version 386
Update Visualization Text Ids 573
updated extended 415
upload from device 963, 966
upload of project 452
upload of source code 452
uppercase 565
uppercase for keywords 485
usage of variables 718
use automatic detected client size 936
use defaults 1028
use newest profile 394
use newest version 394
Use regular expressions 348
Use soruce control for this project (ENI) 549
use specified client size 936
use the following device 358
use the following dimensions 268
use the following library that has already been installed 358
use visualization as 604
user account 394, 401, 583, 963
user and access rights management 186
User and Access Rights Management 401
user defined data type 1096
user group 963
user groups 394
user information 174
user interface
  customizing 169
  language 484, 1070
  reset window layout 630
user interface 169
user interface 174
user logoff 585
user logon 583
user management
  store 963
user management 186, 394
User Management 401, 582
User Management commands 582
user management for device 963
user name 583, 963
User name
  ENI Server 563
user Owner 186, 401
user parameters 1035
user paramters 1032
userdefined attribute 1103
users
  user management 394
users and groups 394, 401, 963
users dialog 401
USINT 1137
Util.library 1230
V2.3 project import 358
V2.3 projects 66
value 753

value of a variable 939
value of device parameter 948
VAR 1087
VAR CONSTANT 1093
VAR PERSISTENT 102, 1090
VAR RETAIN 1090
VAR_CONFIG 1095
VAR_EXTERNAL 1089
VAR_GLOBAL 1088
VAR_IN_OUT 278, 1088
var_in_out in FBD/LD boxes 328
VAR_INPUT 1087
VAR_INPUT CONSTANT 277
VAR_OUTPUT 1087
VAR_STAT 1089
VAR_TEMP 1089
variable
  accessing 1216
  constant 1093
variable comment 939
variable configuration 1095
variable initialization 1079
variable name 1073
variable size 1165, 1208
variable value 939
variable values 797
variables
  insert via input assistant 533
  positions 596
  positions of use 596
variables categories 533
variables declaration 1071
variables initialization 1080
variables references 258, 596
variables update 956
vendor id
  check EtherCAT configuration 1001
vendor name 969
version
  library 421
  project 358
version 65
version compatibility 358, 386
version constraint 65
Version Details 558
version handling 60, 394
version handling for library 426
version information 407
version of a device 969
version of library 146
version of programming system 367
version update 358
vertical tab group 630
view
  auto hide 628
  cross reference view 596
  devices 602
  messages 622
  POUs 68, 602
  Source Control 624
  toolbox 625
  WatchList 626
view 560
View
  Move 169
  Open and close 169
View 169
View 174
view as function block diagram 346
View as instruction list 347
view as ladder logic 346, 347
view commands
  breakpoints 598
  messages 622
view commands 598
view commands 598

view commands 623
view commands 625
View commands 603
view docking 627, 628
view menu 179
view options for sfc editor 501
Views
  arrangement 169
  Arranging 169
virtual mode 1043
visibility of library references 426
visu_usewstring 105
Visual Commands 588
visual element properties 625
visual element repository 200
Visual Element Repository 415
visual element trace 217
visualization
  array as input 1103
  background 593
  bar display 214
  bitmap 593
  change by visualization input 890
  data source 742, 748
  display size 604
  edit control 229
  element list 593
  expandfully 1103
  external device 206
  frame selection 590
  grid 510
  group 589, 889
  HMI 206
  hotkeys 592
  image 593
  integrated mode 203
  key event 229
  keyboard configuration 890
  keyboard usage 229, 595
  meter 211
  options 510
  overview 188
  preconditions 191
  running in programming system 203
  table 223
  text input 890
  textlists 105
  toggle 590
  trace 217
  ungroup 590
  use as 604
visualization 188
visualization 188
visualization as dialog 236, 890
visualization background 112
visualization directories 394
visualization editor 192
Visualization Editor 588
visualization element
  adding 589
  complex control 211, 214
  complex controls 217
  insert 885
  invisible 890
  position 889
  selection 229, 595, 887
  size 889
  text 887
  visual 208
  windows controls 223
visualization element 890
visualization element properties 625, 890
visualization elements
  grouping 589
  ungrouping 590
visualization elements 890

visualization elements for recipe handling 797
Visualization elements of type Visual 208
visualization files 394, 510
visualization instance 193
visualization interface 193
visualization interface editor 926
visualization libraries 199
Visualization Manager
  Target-Visualization 936
visualization modes 188
visualization object
  create 191
visualization object 191
visualization object properties 604
visualization printout 890
visualization profile
  define 415
  name 415
visualization profile 191, 200, 386, 394
visualization profile 415
visualization profile 885
visualization profile update 358
visualization reference 193
Visualization running on external devices 206
visualization settings 394
visualization size 604
visualization template
  example 1055
visualization toolbox 885
visualizaton
  color 593
visuallization editor
  keyboard usage 887
visudialogs.library 199, 236
VisuElemBase.library 199
VisuElemMeter.library 199
VisuElems.library 199, 200
VisuElemWinControls.library 199
VisuInputs.library 199
visumanager 236
W 1220
warm start 456
warning 1101
warning for data types 1122
warning message 622
watch 626
watch all forces 626, 941
watch list
  create 939
  online mode 941
watch view 941
watch window 174
watchdog
  profibus slave 1032
  task 858
  task configuration 858
watchdog control 1032
watchlist
  add all forces 464
watchlist 174
WatchList 626
web browser 202
webvisu.htm 202
Web-Visualization 188, 202
WHILE 849
WHILE instruction 842
WHILE loop 849
width of CFC 268
width of operand field 494
width of SFC action 501
width of SFC attribute 501
width of SFC qualifier 501
width of SFC steps 501
window
  auto hide 628
window 174

window <n> 631
window commands 627
window docking 627, 628
window layout 630
window menu 179
windows
  arrangement 169
windows controls 223
windows tab group 630
with arguments 71, 78
With name 501
Within the following objects 348
WORD 1137
word addressing 1220
word wrap 476
working sheet in CFC 268
working sheet size in CFC 665
wrap guide 476
wrap margin 476
write a variable 890
write CAM 266
write CAM data into ASCII table 266
write CAM online file 266
write DCF 980
write EtherCAT parameter 1023, 1024, 1025
write program to ASCII file 287
write recipe 797, 890
write values
  prepare value 459
  watch list 941
write values 457
write values all applications 457
write variables in visualization 229
WriteRecipe 801
writing values 836
WSTRING 1141
X 1220
x offset of CFC 268
x/y-position of a visualization element 890
xml export of objects 435
xml file for loggings 951
XOR 1169
XOR operator in IL 759
XORN 1169, 1209
xref 1128
y offset of CFC 268
zoom 169
zoom in 722
zoom into macro 547
zoom out 722
zoom out of macro 547
zoom to POU 257

© 03/2011

Lenze Automation GmbH
Hans-Lenze-Str. 1
D-31855 Aerzen
Germany

+49 (0)51 54 / 82-0

+49 (0)51 54 / 82-28 00

Lenze@Lenze.de

www.Lenze.com

Service Lenze Service GmbH
Breslauer Straße 3
D-32699 Extertal
Germany

00 80 00 / 24 4 68 77 (24 h helpline)

+49 (0)51 54 / 82-11 12

Service@Lenze.de

SHPPLCD ▪ 13373836 ▪ EN 4.1 ▪ TD29

10 9 8 7 6 5 4 3 2 1