

Form A6126

Part Number D301174X012

March 2009

DS800 Development Suite Software

User Manual

Revision Tracking Sheet

March 2009

This manual is periodically altered to incorporate new or updated information. The date revision level of each page is indicated at the bottom of the page opposite the page number. A major change in the content of the manual also changes the date of the manual, which appears on the front cover. Listed below is the date revision level of each page.

| Page | Revision |
|---|-----------------|
| All Pages | 05/02 |
| 55, 112-117, 119-120, 242, 419-425, 431-454, 538-540, 542, 546-549, 552-553, 556, 559-563, 566-570, 588, 594-5950 | 08/02 |
| All Pages | 07/06 |
| All Pages | 03/09 |

© 2002 - 2009 Remote Automation Solutions, division of Emerson Process Management. All rights reserved.

NOTICE

Remote Automation Solutions ("RAS"), division of Emerson Process Management shall not be liable for technical or editorial errors in this manual or omissions from this manual. RAS MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THIS MANUAL AND, IN NO EVENT SHALL RAS BE LIABLE FOR ANY INCIDENTAL, PUNITIVE, SPECIAL OR CONSEQUENTIAL DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PRODUCTION, LOSS OF PROFITS, LOSS OF REVENUE OR USE AND COSTS INCURRED INCLUDING WITHOUT LIMITATION FOR CAPITAL, FUEL AND POWER, AND CLAIMS OF THIRD PARTIES.

Bristol, Inc., Bristol Canada, BBI SA de CV and Emerson Process Management Ltd., Remote Automation Solutions division (UK) are wholly owned subsidiaries of Emerson Electric Co. doing business as Remote Automation Solutions ("RAS"), a division of Emerson Process Management. ROC, FloBoss, ROCLINK, Bristol, Bristol Babcock, ControlWave, TeleFlow and Helicoid are trademarks of RAS. AMS, PlantWeb and the PlantWeb logo are marks of Emerson Electric Co. The Emerson logo is a trademark and service mark of the Emerson Electric Co. All other trademarks are property of their respective owners.

The contents of this publication are presented for informational purposes only. While every effort has been made to ensure informational accuracy, they are not to be construed as warranties or guarantees, express or implied, regarding the products or services described herein or their use or applicability. RAS reserves the right to modify or improve the designs or specifications of such products at any time without notice. All sales are governed by RAS' terms and conditions which are available upon request.

RAS does not assume responsibility for the selection, use or maintenance of any product. Responsibility for proper selection, use and maintenance of any RAS product remains solely with the purchaser and end-user.

Table of Contents

Workbench 1

| | |
|--|----|
| Apearance | 3 |
| Title Bar | 4 |
| Menu Bar | 5 |
| Toolbars | 14 |
| Standard Toolbar..... | 15 |
| Debug Toolbar | 17 |
| Window Buttons Toolbar..... | 19 |
| Layers Toolbar | 19 |
| Version Source Control Toolbar..... | 20 |
| Options Toolbar | 20 |
| I/O Wiring Toolbar | 20 |
| Workspace | 22 |
| Zoom | 23 |
| Output Window | 24 |
| Contextual Menus..... | 25 |
| Status Bar..... | 25 |
| Customization | 26 |
| Directory Structure | 29 |
| Working with Projects | 32 |
| Creating Projects..... | 34 |
| Opening and Closing Projects | 36 |
| Saving Projects | 39 |
| Renaming Projects | 39 |
| Adding a Project Description..... | 40 |
| Printing Projects | 40 |
| Project Access Control..... | 41 |
| Importing and Exporting Workbench Elements | 43 |
| Uploading Workbench Elements from Targets | 46 |
| Link Architecture View | 48 |
| Resources | 48 |
| Resource Window Workspace..... | 49 |
| Creating Resources | 50 |

| | |
|---|----|
| Renaming Resources | 51 |
| Copying Resources | 51 |
| Pasting Resources | 52 |
| Deleting Resources | 53 |
| Editing Resource Properties | 54 |
| Resource Identification..... | 55 |
| Compilation Options..... | 55 |
| Run-time Settings | 59 |
| Resource Network Parameters..... | 62 |
| Custom Resource Parameters | 63 |
| Resource Access Control..... | 64 |
| Resource Description..... | 66 |
| Variable Bindings | 67 |
| Internal Bindings | 71 |
| Linking Resources | 74 |
| Deleting Resource Links..... | 76 |
| Viewing the Internal Bindings Defined for Resources | 77 |
| Hiding and Showing Resource Links | 77 |
| Defining Internal Variable Bindings..... | 78 |
| Editing Internal Variable Bindings | 80 |
| Deleting Internal Variable Bindings | 80 |
| External Bindings | 81 |
| Defining Producer Variable Groups | 83 |
| Editing Producer Variable Groups..... | 85 |
| Deleting Producer Variable Groups..... | 85 |
| Linking Resources for External Bindings | 86 |
| Editing External Resource Links | 87 |
| Defining External Variable Bindings..... | 88 |
| Editing External Variable Bindings..... | 89 |
| Deleting External Variable Bindings | 89 |
| Parameters | 90 |
| Variable Groups | 91 |
| Creating Variable Groups..... | 91 |
| Opening Variable Groups..... | 91 |
| Importing or Exporting Variables | 93 |
| POUs (Program Organization Units) | 96 |
| Programs | 96 |
| Functions | 98 |
| Function Blocks | 99 |

| | |
|--|-----|
| Creating POU..... | 99 |
| Manipulating POU..... | 100 |
| Creating FC Sub-programs | 102 |
| Creating SFC Child POU..... | 102 |
| Changing Hierarchy Level..... | 103 |
| Controlling Access to POU..... | 104 |
| Generating Debug and Monitoring Information..... | 107 |
| Editing a POU Description | 108 |
| Hardware Architecture View | 109 |
| Configurations | 110 |
| Creating Configurations..... | 110 |
| Deleting Configurations..... | 111 |
| Moving Configurations..... | 112 |
| Inserting Resources..... | 112 |
| Moving Resources Between Configurations | 113 |
| Configuration Properties..... | 114 |
| Configuration Link to ROCLINK Configuration File | 115 |
| Configuration Target Definitions..... | 117 |
| Target Access Control..... | 118 |
| Configuration Description | 119 |
| Networks..... | 120 |
| Creating Networks | 120 |
| Moving Networks | 121 |
| Connections | 123 |
| Creating Connections..... | 123 |
| Deleting Connections..... | 124 |
| Dictionary View..... | 125 |
| Appearance | 126 |
| Variables Tree..... | 127 |
| Parameters Tree | 128 |
| Types Tree | 129 |
| Creating Structures..... | 129 |
| Renaming Structures..... | 130 |
| Deleting Structures..... | 130 |
| Defined Words Tree | 130 |
| Working with the Grids | 131 |
| Resizing Columns | 132 |
| Selecting Rows and Elements | 132 |

| | |
|---|-----|
| Editing the Contents of the Grid..... | 133 |
| Adding or Inserting Rows..... | 134 |
| Moving Rows | 135 |
| Expanding or Collapsing Grid Components..... | 135 |
| Cutting, Copying, and Deleting Elements | 136 |
| Finding and Replacing Elements | 137 |
| Pasting Elements..... | 138 |
| Sorting the Grid | 138 |
| Duplicating Rows | 139 |
| Renumbering Addresses | 140 |
| Printing a Grid | 141 |
| Variables Grid | 142 |
| Parameters Grid..... | 143 |
| Types Grid..... | 144 |
| Defined Words Grid | 145 |
| Defining TLP Variables | 146 |
| Initial Values | 150 |
| Validation | 153 |
| Cell-level Validation..... | 153 |
| Row-level Validation..... | 153 |
| Database-level Validation..... | 154 |
| I/O Wiring View | 155 |
| Appearance | 156 |
| I/O Wiring Tree View | 157 |
| I/O Wiring Grid View | 159 |
| Working with the I/O Wiring Tool..... | 160 |
| TLP Devices (Automatic Wiring) | 162 |
| Analog Input - 4 Point | 163 |
| Analog Output - 4 Point..... | 163 |
| Discrete Input - 8 Point..... | 164 |
| Discrete Output - 5 Point | 164 |
| Multi-Variable Sensor Input - 6 Point | 165 |
| Pulse Input - 2 Point | 166 |
| RTD Input - 2 Point | 167 |
| System Analog Input - 5 Point | 167 |
| Thermocouple Input - 5 Point..... | 168 |
| Adding I/O Devices | 169 |
| Opening Devices..... | 170 |
| Deleting Devices and Conversions..... | 171 |

| | |
|---|-----|
| Setting the Real or Virtual Attribute | 171 |
| Wiring Channels | 172 |
| Mapping Channels | 172 |
| Freeing Channels | 174 |
| Run-time System Events..... | 175 |
| Logging Events | 175 |
| Viewing Events..... | 176 |
| Language Editors | 181 |
| Common Editor Features | 181 |
| Appearance | 182 |
| Menu Bar..... | 183 |
| Toolbars..... | 184 |
| Standard Toolbar | 185 |
| Options Toolbar..... | 186 |
| Debug Toolbar..... | 187 |
| SFC Breakpoints Toolbar..... | 189 |
| SFC Tools..... | 189 |
| Flow Chart Tools..... | 191 |
| ST Tools | 192 |
| IL Tools | 193 |
| LD Tools..... | 194 |
| FBD Tools | 195 |
| Workspace | 197 |
| Contextual Menus | 199 |
| Output Window | 199 |
| Status Bar | 200 |
| Inserting Identifiers..... | 201 |
| Inserting Blocks | 203 |
| Printing POU..... | 205 |
| Opening the Dictionary..... | 205 |
| Opening Another POU..... | 206 |
| Finding and Replacing in POU..... | 207 |
| SFC Editor | 209 |
| Appearance | 210 |
| Menu Bar | 211 |

| | |
|--|-----|
| Working with the Editor | 214 |
| SFC Elements..... | 215 |
| Initial Step | 215 |
| Step..... | 216 |
| Transition | 216 |
| Divergence/Convergence | 217 |
| Creating New Branches..... | 219 |
| Deleting Branches | 220 |
| Link | 221 |
| Jump | 222 |
| Managing Elements | 223 |
| Select..... | 223 |
| Rename..... | 224 |
| Move | 225 |
| Cut..... | 225 |
| Copy | 225 |
| Paste | 226 |
| Delete | 227 |
| Goto..... | 227 |
| Level 2 | 228 |
| Coding Action Blocks for Steps..... | 229 |
| Coding Conditions for Transitions..... | 231 |
| Moving Action Blocks Up or Down | 232 |
| Deleting an Action Block..... | 233 |
| Renumbering Charts | 233 |
| FC Editor | 235 |
| Appearance | 235 |
| Menu Bar | 236 |
| Working with Flow Charts | 239 |
| Flow Chart Elements..... | 240 |
| Action..... | 240 |
| Test..... | 240 |
| IF-THEN-ELSE | 241 |
| DO-WHILE..... | 242 |
| WHILE-DO..... | 242 |
| Flow..... | 243 |
| Connector | 244 |
| I/O Specific | 244 |

| | |
|--------------------------------|-----|
| Comment | 245 |
| Sub-Program..... | 245 |
| Managing Elements..... | 246 |
| Select | 246 |
| Cut | 247 |
| Copy | 247 |
| Paste..... | 248 |
| Delete..... | 248 |
| Move..... | 248 |
| GoTo..... | 249 |
| Renumber | 249 |
| Level 2..... | 250 |
| Level 2 Window | 251 |
| Edit the Level 2 | 252 |
| Multi-language Editor..... | 253 |
| Appearance | 254 |
| Menu Bar | 256 |
| Multi-Language Elements..... | 260 |
| ST/IL Elements | 260 |
| LD Elements..... | 261 |
| Contact on the Left | 261 |
| Contact on the Right | 261 |
| Parallel Contact | 262 |
| Coil | 262 |
| Block on the Left | 262 |
| Block on the Right | 262 |
| Parallel Block | 262 |
| Jump | 262 |
| Label..... | 263 |
| Return | 263 |
| Change Coil/Contact Type | 263 |
| Insert New Rung | 264 |
| Other Operations | 264 |
| FBD Elements | 265 |
| Variable | 266 |
| Function Block | 267 |
| Link | 267 |
| Corner | 267 |
| Jump | 268 |

| | |
|-----------------------------------|-----|
| Label | 268 |
| Return..... | 269 |
| LD Elements..... | 270 |
| Left Power Bar | 270 |
| Contacts | 270 |
| LD Vertical "OR" Connection | 270 |
| Coils..... | 271 |
| Right Power Bar | 271 |
| Comment..... | 272 |
| Managing Elements | 273 |
| Select..... | 273 |
| Resize..... | 274 |
| Undo/Redo | 274 |
| Move | 275 |
| Cut..... | 275 |
| Copy | 276 |
| Paste | 276 |
| Paste Special | 277 |
| Delete | 277 |
| Select All..... | 278 |
| Find Matching Name | 278 |
| Find Matching Coil..... | 278 |
| Go to Line | 279 |
| Display/Hide Comments..... | 279 |
| Libraries..... | 281 |
| Creating Libraries..... | 281 |
| Using Libraries in a Project..... | 282 |
| Debug..... | 289 |
| Status Information | 290 |
| Download | 293 |
| Debug/Simulate | 295 |
| Start / Stop a Resource..... | 297 |
| Resource Execution Mode..... | 298 |
| Real-time Mode | 298 |
| Cycle-to-cycle Mode..... | 299 |

| | |
|---|-----|
| Step-by-step Mode | 299 |
| Setting Breakpoints | 301 |
| Removing Breakpoints | 301 |
| Stepping in POU..... | 302 |
| Set Cycle Time..... | 303 |
| Write / Lock / Unlock | 304 |
| Diagnosis..... | 307 |
| SFC Breakpoints | 311 |
| Breakpoint on Step Activation | 312 |
| Breakpoint on Step Deactivation | 313 |
| Breakpoint on Transition..... | 314 |
| Transition Clearing Forcing | 315 |
| Spying Variables..... | 316 |
| Adding Variables to the Spy List | 316 |
| Selecting Variables in the Spy List | 317 |
| Removing Variables from the Spy List..... | 318 |
| Rearranging the Spy List..... | 318 |
| Saving a Spy List | 318 |
| Opening an Existing Spy List | 319 |
| Forcing the Value of a Spy List Variable..... | 319 |
| Simulate a Panel of I/Os | 320 |
| Appearance | 322 |
| Menu Bar..... | 323 |
| Toolbar | 324 |
| Contextual Menu..... | 325 |
| Displaying I/O Device Window Headers..... | 325 |
| Moving or Hiding the Browser | 326 |
| Online Changes..... | 327 |
| Code Sequences | 327 |
| Variables | 329 |
| Declared Variables | 329 |
| Function Block Instances | 330 |
| Compiler Allocated Hidden Variables | 330 |
| I/O Devices | 331 |
| Memory Requirements..... | 331 |
| Miscellaneous Limitations | 331 |
| Operations | 332 |
| Debug Function Block Instances | 334 |
| Clean Stored Code | 336 |

| | |
|---|-----|
| Document Generator..... | 337 |
| Table of Items..... | 338 |
| Printing Options | 340 |
| Preview | 342 |
| Code Generator..... | 345 |
| Build | 345 |
| Build a POU..... | 346 |
| Building Resources / Projects..... | 347 |
| Stopping Builds | 348 |
| Cleaning Projects | 348 |
| Compiler Options | 349 |
| C Source Code..... | 351 |
| Project Tree View | 353 |
| Cross References Browser..... | 355 |
| Calculating Cross References..... | 357 |
| Browsing the POU's of a Project..... | 357 |
| Defining Search Options | 358 |
| Version Source Control | 359 |
| Performing a Check in of a Workbench Element..... | 363 |
| Viewing the History of Workbench Elements | 364 |
| Getting a Previous Version | 365 |
| Comparing Current and Previous Versions | 365 |
| Accessing Details for a Previous Version | 366 |
| Creating a History Report..... | 366 |

Language Reference 367

| | |
|--|-----|
| Project Architecture | 368 |
| Programs..... | 368 |
| Cyclic and Sequential Operations | 369 |
| Child SFC POUs | 370 |
| FC Sub-Programs | 371 |
| Functions | 371 |
| Function Blocks..... | 373 |
| Description Language..... | 375 |
| Execution Rules..... | 376 |

| | |
|--|-----|
| Common Objects | 377 |
| Data Types | 377 |
| Standard IEC 61131 Types | 377 |
| User Types: Arrays | 379 |
| User Types: Structures | 380 |
| Constant Expressions..... | 381 |
| Boolean Constant Expressions..... | 381 |
| Short Integer Constant Expressions..... | 381 |
| Double Integer Constant Expressions..... | 382 |
| Real Constant Expressions..... | 382 |
| Timer Constant Expressions | 383 |
| String Constant Expressions | 383 |
| Variables | 385 |
| Reserved Keywords | 385 |
| Directly Represented Variables | 387 |
| Information on Variables | 389 |
| Boolean Variables (BOOL) | 390 |
| Short Integer Variables (SINT)..... | 390 |
| Double Integer Variables (DINT)..... | 390 |
| Real Variables (REAL)..... | 390 |
| Timer Variables (TIME) | 391 |
| String Variables (STRING) | 391 |
| Comments | 392 |
| Defined Words..... | 392 |
| SFC Language..... | 395 |
| SFC Main Format | 395 |
| SFC Basic Components | 396 |
| Steps and Initial Steps | 396 |
| Transitions..... | 397 |
| Oriented Links | 398 |
| Jump to a Step | 398 |
| Divergences and Convergences | 400 |
| Single Divergences (OR) | 400 |
| Double Divergences (AND) | 402 |
| Actions Within Steps | 404 |
| Boolean Actions..... | 404 |
| Pulse Actions | 405 |
| Non-stored Actions | 406 |
| SFC Actions | 407 |

| | |
|---|-----|
| List of Instructions | 408 |
| Calling Functions and Function Blocks..... | 409 |
| Conditions Attached to Transitions | 410 |
| Condition Programmed in ST | 410 |
| Condition Programmed in LD | 411 |
| Condition Programmed in IL..... | 411 |
| Calling Functions from a Transition | 412 |
| Calling Function Blocks from a Transition | 413 |
| SFC Dynamic Behavior..... | 414 |
| SFC Program Hierarchy | 415 |
| FC Language..... | 417 |
| FC Basic Components | 417 |
| FC BEGIN | 418 |
| FC END | 418 |
| FC Flow Links..... | 419 |
| FC Actions | 420 |
| FC Conditions | 420 |
| Other FC Components..... | 422 |
| FC Sub-Program | 422 |
| FC I/O Specific Actions..... | 423 |
| FC Connectors | 424 |
| FC Comments | 424 |
| FC Complex Structure Examples | 425 |
| FC Dynamic Behavior..... | 426 |
| FC Checking..... | 426 |
| FC Examples | 427 |
| FBD Language..... | 429 |
| FBD Diagram Main Format | 429 |
| RETURN Statement | 431 |
| Jumps and Labels | 431 |
| Boolean Negation | 433 |
| Calling Functions and Function Blocks | 433 |
| LD Language | 435 |
| Power Rails and Connection Lines | 436 |
| Multiple Connections | 437 |
| Basic LD Contacts and Coils..... | 439 |
| Direct Contact..... | 440 |
| Inverted Contact..... | 440 |

| | |
|--|---------|
| Contact with Rising Edge Detection..... | 441 |
| Contact with Falling Edge Detection..... | 442 |
| Direct Coil..... | 443 |
| Inverted Coil | 444 |
| SET Coil..... | 445 |
| RESET Coil | 446 |
| Coil with Rising Edge Detection | 447 |
| Coil with Falling Edge Detection | 448 |
| RETURN Statement | 449 |
| Jumps and Labels..... | 450 |
| BLOCKS in LD | 451 |
| ST Language | 453 |
| ST Main Syntax | 453 |
| Expressions and Parentheses | 455 |
| Functions or Function Block Calls | 456 |
| Calling Functions | 456 |
| Calling Function Blocks | 457 |
| ST Operators..... | 459 |
| ST Basic Statements | 459 |
| Assignment..... | 459 |
| RETURN Statement..... | 460 |
| IF-THEN-ELSIF-ELSE Statement | 461 |
| CASE Statement | 462 |
| WHILE Statement..... | 463 |
| REPEAT Statement | 464 |
| FOR Statement..... | 465 |
| EXIT Statement | 466 |
| ST Extensions | 467 |
| GSTART Statement in SFC Action | 468 |
| GKILL Statement in SFC Action | 469 |
| GFREEZE Statement in SFC Action | 470 |
| GRST Statement in SFC Action | 471 |
| GSTATUS Statement in SFC Action | 472 |
| IL Language | 473 |
| IL Main Syntax | 473 |
| Labels | 474 |
| Operator Modifiers..... | 474 |
| Delayed Operations..... | 475 |

| | |
|--|-----|
| IL Operators | 476 |
| LD Operator..... | 477 |
| ST Operator | 478 |
| S Operator..... | 478 |
| R Operator | 479 |
| JMP Operator..... | 480 |
| RET Operator..... | 481 |
|) Operator..... | 482 |
| Calling Functions..... | 483 |
| Calling Function Blocks: CAL Operator..... | 485 |
| Standard Operators | 487 |
| * | 488 |
| + | 489 |
| - | 491 |
| / | 492 |
| 1 GAIN | 494 |
| AND | 495 |
| ANY_TO_BOOL | 496 |
| ANY_TO_SINT | 498 |
| ANY_TO_DINT | 499 |
| ANY_TO_REAL | 501 |
| ANY_TO_TIME | 502 |
| ANY_TO_STRING..... | 504 |
| Equal..... | 505 |
| Greater Than or Equal | 507 |
| Greater Than | 508 |
| Less Than or Equal..... | 510 |
| Less Than | 511 |
| NEG..... | 512 |
| NOT | 514 |
| Not Equal..... | 515 |
| OR | 516 |
| TMR | 517 |
| XOR..... | 518 |
| Standard Functions | 521 |
| ABS | 522 |
| ACOS | 523 |
| AND_MASK..... | 524 |

| | |
|------------------|-----|
| ASCII | 525 |
| ASIN | 526 |
| ATAN | 527 |
| CHAR | 528 |
| COS | 530 |
| CURRENT_ISA_DATE | 531 |
| DELETE | 532 |
| EXPT | 533 |
| FIND | 534 |
| INSERT | 536 |
| LEFT | 537 |
| LIMIT | 539 |
| LOG | 540 |
| MAX | 541 |
| MID | 542 |
| MIN | 543 |
| MLEN | 544 |
| MOD | 546 |
| MUX4 | 547 |
| MUX8 | 549 |
| NOT_MASK | 550 |
| ODD | 551 |
| OR_MASK | 553 |
| POW | 554 |
| RAND | 555 |
| REPLACE | 556 |
| RIGHT | 558 |
| ROL | 559 |
| ROR | 560 |
| SEL | 562 |
| SHL | 563 |
| SHR | 564 |
| SIN | 565 |
| SQRT | 566 |
| SUB_DATE_DATE | 567 |
| TAN | 569 |
| TRUNC | 570 |
| XOR_MASK | 571 |

| | |
|-------------------------------|-----|
| Standard Function Blocks..... | 573 |
| ALARM..... | 575 |
| AVERAGE..... | 575 |
| BLINK..... | 577 |
| CMP..... | 578 |
| CONNECT | 579 |
| CTD..... | 581 |
| CTU..... | 582 |
| CTUD | 583 |
| DBG_CLR_GET_ERR | 585 |
| DBG_CLR_SET_ERR..... | 585 |
| DBG_GET_ERR..... | 586 |
| DBG_SET_ERR..... | 586 |
| DERIVATE..... | 587 |
| EVENT..... | 588 |
| F_TRIG | 588 |
| HYSTER | 589 |
| INTEGRAL..... | 590 |
| R_TRIG | 591 |
| REQUEST_LICENSE..... | 592 |
| RS | 593 |
| SET_PRIORITY | 594 |
| SIG_GEN | 595 |
| SOFT_POINT_READ..... | 596 |
| SOFT_POINT_WRITE..... | 597 |
| SR | 598 |
| STACKINT | 600 |
| TLP_GET_DINT..... | 601 |
| TLP_GET_REAL..... | 602 |
| TLP_GET_SINT | 603 |
| TLP_GET_STRING..... | 604 |
| TLP_GET_TLP | 605 |
| TLP_SET_DINT | 606 |
| TLP_SET_REAL | 607 |
| TLP_SET_SINT | 608 |
| TLP_SET_STRING | 609 |
| TOF | 609 |
| TON..... | 610 |
| TP | 611 |

| | |
|-----------------|-----|
| URCV_S..... | 612 |
| USEND_S..... | 613 |
| Glossary | 615 |
| Copyright | 661 |

Workbench

The DS800 software suite supports both the ROC800-Series and the FloBoss 107 flow computers from Remote Automation Solutions (RAS). To simplify usage, this documentation refers to both devices as the "RAS device." If there is a situation where we restrict functionality to either the ROC800-Series or the FB107, we note it.

The Workbench is the environment in which you develop multi-process control projects made up of virtual machines running on hardware components, called target nodes. The development process consists of creating projects made up of configurations, representing, individual target nodes, on which one or more instances of resources, i.e., virtual machines, are downloaded. At runtime, the virtual machines run on these target nodes.

Projects can be developed using any of the five languages of the IEC 61131 standard: SFC: Sequential Function Chart (or Grafcet), FBD: Function Block Diagram, LD: Ladder Diagram, ST: Structured Text, and IL: Instruction List. You can also use the Flow Chart language. When building, resources are compiled to produce very fast "target independent code" (TIC) or "C" code.

Within resources, you can declare variables using standard IEC 61131 data types (i.e., Boolean, integer, real, etc.) or user-defined types such as arrays or structures. For defined variables, you can set up alarms, events, and trending. Furthermore, field communications allow you to connect variables to field equipment. Resources can share variables using internal bindings or external bindings. Internal bindings are between resources within the same project. External bindings are between resources belonging to different projects.

You develop projects on a Windows development platform, in the Workbench and language editors. The Workbench graphically represents and organizes configurations, resources, POU s, and networks within a project from multiple views:

- link architecture
- hardware architecture
- dictionary
- I/O wiring
- bindings

Libraries made up of configurations and resources enable you to define functions and function blocks for reuse throughout projects.

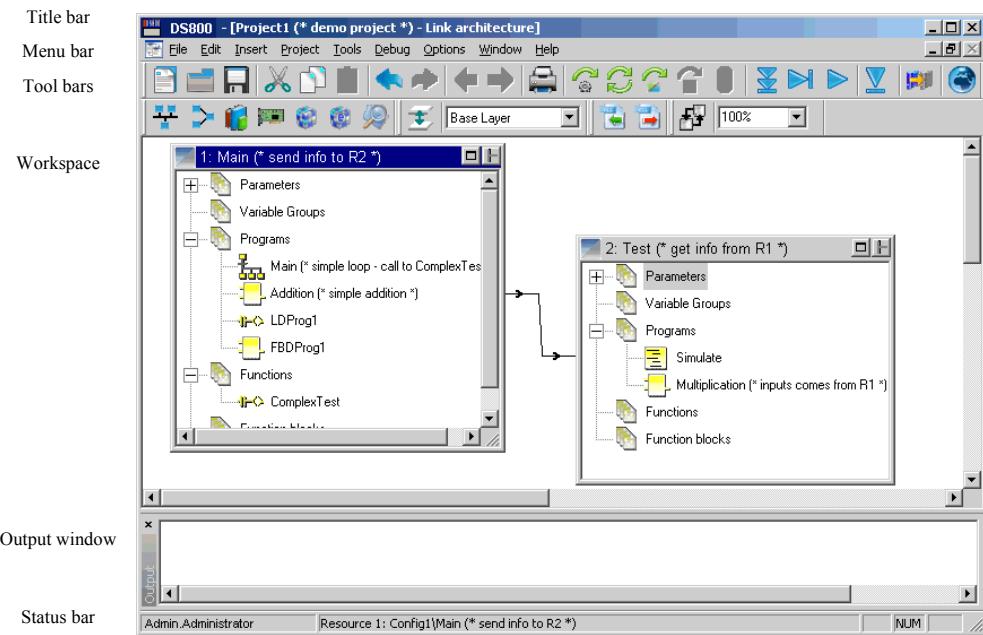
Individual resources, from the configurations making up a project, are downloaded, using the ETCP or ISARSI (serial link) network, onto target RAS device nodes running real-time operating systems. Communication between configurations can be implemented using the TCP\IP network. You can choose to implement any other network.

You can choose to simulate the running of a project, after building a project, using high-level debugging tools, before actually downloading the resources making up configurations to the target nodes.

You can set four levels of access control in a Workbench application:

- password protection and read-only mode for a complete project
- password protection and read-only mode for individual resources
- password protection for individual POU s
- password protection for a target

Appearance



Title Bar

For help locating the Title Bar, see the Appearance diagram. The Title Bar displays the application name and the filename of the active project, if any are open, along with the current view (Hardware Architecture, Link Architecture, Dictionary or I/O Wiring).

Control Icon

At the left end of the Title Bar is the Control Icon, which is used to access the Control Menu (see following section). Double-clicking on the Control Icon closes the Workbench.

Control Menu

Clicking on the Control icon opens the Control Menu. The Control Menu is used to position the Main Window or to exit.

Window Buttons

The standard window buttons appear at the right end of the Title Bar. Use these to resize or close the Window.

Menu Bar

The options available from the menu bar differ slightly for the hardware architecture and link architecture views of a project. Some options are available as keyboard commands.

| | | | |
|------|------------------------|--------|---|
| File | New Project/Library | Ctrl+N | creates a new project or library |
| | Open Project/Library | Ctrl+O | opens an existing project or library |
| | Save Project/Library | Ctrl+S | saves the current project or library |
| | Rename Project/Library | | renames the current project or library |
| | Project Properties | | sets project access control |
| | Import | | imports three types of information: - PLC definitions using text files generated with the Target Definition Builder - Workbench elements (projects, configurations, resources, and POU)s) - Variables data |
| | Export | | exports Workbench elements (projects, configurations, resources, and POU)s) or variables data |
| | Print | Ctrl+P | accesses the Document Generator |
| | Exit | Ctrl+Q | leaves the Workbench |

| | | | |
|------|------------------------|--------|---|
| Edit | Open | Alt+N | opens the item selected from a resource. This option is only available in the link architecture view. |
| | Undo | Ctrl+Z | cancels the last action |
| | Redo | Ctrl+Y | restores the last cancelled action |
| | Cut | Ctrl+X | removes the selected item and places it on clipboard |
| | Copy | Ctrl+C | takes a copy of the selected item and places it on the clipboard. For the link architecture view, this option appears as Copy Program where it copies an entire selected program. |
| | Paste | Ctrl+V | inserts the contents of the clipboard into the selected item |
| | Delete | DEL | removes the selected item from the selected item |
| | Find / Replace in POUs | Ctrl+F | finds and replaces text in a project, a configuration, a resource, or a POU |
| | Select All | Ctrl+A | selects all items in the active view |
| | Properties | | accesses the properties for the selected item |
| | Move to lower level | | sets the selected FC or SFC program as a sub-program of the next program in the resource. This option is only available in the link architecture view. |
| | Move to upper level | | sets the selected FC or SFC program as a parent program of the previous program in the resource. This option is only available in the link architecture view. |

| | | |
|--------|---------------------|--|
| Insert | Configuration | inserts a configuration in the workspace. This option is only available in the hardware architecture view. |
| | Resource | inserts a resource. For the hardware architecture view of a project, you insert resources in selected configurations. For the link architecture view, you insert resources in the workspace. |
| | Network | inserts a network in the workspace. This option is only available in the hardware architecture view. |
| | Add Variable Group | adds a variable group to the selected resource. This option is only available in the link architecture view. |
| | Add Program | adds a program to the selected resource. This option is only available in the link architecture view. |
| | Add SFC Sub-program | adds an SFC sub-program to the selected program. When an FC program is selected, adds an FC sub-program. This option is only available in the link architecture view. |

| | | | |
|---------|-------------------------------------|--------|---|
| Project | Types | Ctrl+3 | accesses the Types Tree of the Dictionary view |
| | Variables | Ctrl+G | accesses the Variables Tree of the Dictionary view |
| | Function /Function Block Parameters | | accesses the Parameters Tree of the Dictionary view. This option is only available in the link architecture view. |
| | External Binding List | Ctrl+0 | accesses the External Binding list window where you can define external variable bindings between producer variables of a source resource in a given project with consumer variables of a destination resource in a different project |
| | Internal Binding List | Ctrl+1 | accesses the Binding List window for the selected binding. This option is only available in the link architecture view. |
| | Defined Words | Ctrl+2 | accesses the Defined Words Tree of the Dictionary view |
| | I/O Wiring | | |
| | Build Project/Library | | compiles the current project or library |
| | Rebuild Project/Library | | recompiles the complete current project |
| | Clean Project/Library | | removes files created during the last build of the current project or library |
| | Build Resource | | compiles the selected resource |
| | Clean Resource | | removes files created during the last build of the selected resource |
| | Build Program | | compiles the selected program. This option is only available in the link architecture view. |
| | Stop Build | | stops a build in progress |

| | | | |
|-------|--------------------------|--------|--|
| Tools | Compact Database | | optimizes the current project's database |
| | Edit Project Description | Ctrl+K | accesses the description editor for the current project or library |
| | Edit Description | | accesses the description editor for the selected item |
| | Unlock Resource | | unlocks a resource currently locked by another user. This option is only available when editing a project in normal mode and one or more resources of the project are opened in single-resource editing mode by other users. |
| | Add/Remove Dependencies | | accesses the Add/Remove Dependencies window where you define the libraries used by a project |
| | Browser | Ctrl+B | accesses the cross references browser listing and localizing all instances of global variables (cross references) and I/Os declared in a project |
| | Check-in | | Checks in a project, configuration, resource, or POU definition not having the read-only attribute into a version source control database |
| | View History | | Views the history of a project, configuration, resource, or POU that has been checked in repeatedly to a version source control database |
| | Events Viewer | | accesses the Events Viewer |

| | | | |
|-------|---------------------------------|--------|--|
| Debug | Download | Ctrl+M | accesses the Download editor from where you download resources onto target nodes |
| | Debug Target | Alt+F6 | starts the project in debug mode |
| | Simulation | Alt+F7 | starts the project in simulation mode |
| | On-line Change: Download | | downloads only the changes made since the last download for the selected running resource. The download includes the symbol table (complete or reduced as selected in the resource's compilation options). |
| | On-line Change: Update | | updates a resource running on a target to use the latest on-line change download code. For use when you chose to update the resource code later. |
| | Start | | starts the selected resource, while in run mode |
| | Stop | | stops the selected resource, while in run mode |
| | Start from code saved on Target | | restarts the selected resource using the code saved on the target node |
| | Save Code on target | | saves the code of the running resource (including changes) |
| | Clean Stored Code | | removes code previously saved on a target |
| | Diagnosis | | accesses the Diagnosis window displaying general and status information for the selected resource |
| | Refresh Status | | updates the resource status information, appearing in the title bar, for all resources |

| | | | |
|-----------------|----------------------------|---------|---|
| Debug (cont) | Real Time / Cycle to Cycle | | switches between real time and cycle to cycle mode for the selected resource |
| | Execute one cycle | Alt+F10 | executes one cycle at a time, while in cycle to cycle mode |
| | Change Cycle timing | | accesses the Cycle Time editor where you set the cycle time for the selected resource |
| | Step | Alt+F8 | executes the current line then steps to the next line |
| | Step Into | Alt+F9 | executes the current line then steps into the next line |
| | Show Current Step | | shows the current step |
| | Layout | | accesses the Layout editor where you specify which toolbars to display and the magnification of the workspace area |
| Options | Customize | Ctrl+U | accesses the customization properties for Workbench views and editors |
| | Hide Links | | enables hiding or showing of the different types of binding links between resources. This option is only available in the link architecture view. |

| | | | |
|--------|-----------------------|--------|--|
| Window | Cascade | | sets the different views of the project to appear in a cascading manner |
| | Tile | | sets the different views of the project to appear in a tiled manner |
| | Show Spy List | | accesses the Spy List window where you specify variables whose values are displayed while in test mode |
| | Show Project Contents | | displays the project structure and enables accessing aspects of the currently opened project |
| | Show Output Window | Ctrl+4 | displays the output window below the workspace |
| | Clear Output Window | | clears the contents of the output window |
| Help | Contents | F1 | accesses the online help |
| | Search Help On... | | not currently supported |
| | About | | displays product and version information |
| | Support Info | | not currently supported |

Note: When no projects are open, only the File and Help menus are visible.

Using the Menus:

1. Open a menu by clicking on it, or by pressing (**Alt**) plus the letter that is underlined in the menu's title. For example, to open the File Menu, you press (**Alt**) + (**F**) (shown in this User's Guide as (**ALT+F**)).
2. Choose a menu selection by clicking on it, by pressing its underlined letter, or by using the cursor keys to highlight it and then pressing (**Enter**). Menu selections that appear in grey are not currently available.

Control Icon

When a project is open and not displayed in Cascade or Tile mode, the menu bar has a Control Icon on the left. This icon indicates the current view.

Control Menu

Clicking on the Control Icon opens the Control Menu. The Control Menu is used to position the Window or to alternate between views (see Window Buttons Toolbar).

Window Buttons

The standard window buttons appear at the right end of the menu bar.

Toolbars

Many toolbars performing different tasks are available for use in the hardware and link architecture views:

- Standard Toolbar
- Debug Toolbar
- Window Buttons Toolbar
- Layers Toolbar
- Version Source Control Toolbar
- Options Toolbar
- I/O Wiring Toolbar

While performing I/O wiring tasks in the I/O Wiring view, the I/O Wiring toolbar becomes available.

To show or hide toolbars

You can choose to show as many toolbars as required.

1. From the Options menu, choose **Layout**.

The Layout editor appears.

2. To show toolbars, check the required toolbars then click **OK**.
3. To hide toolbars, uncheck the toolbars then click **OK**.

To move a toolbar

Toolbars can be placed anywhere on the screen, their position is retained until the next change.

1. Point the cursor at the toolbar's title bar or main panel. Do not point at the control icon or one of the window's buttons.

2. Press and hold the left mouse-button.
3. Drag the toolbar by moving the mouse.
4. Release the mouse-button.

Docking toolbars

- Dock a toolbar to a side of the Workspace by positioning it at the Workspace's edge. Switch back and forth between a toolbar's floating and docked states.

Standard Toolbar



Creates a project



Opens a project



Saves the current project



Cuts the selection and places it on the clipboard



Copies the selection and places it on the clipboard



Pastes the contents of the clipboard



Undoes the last operation



Redoes the previously undone operation

-  Moves to upper level on currently selected SFC or FC program
-  Moves to lower level on currently selected SFC or FC program
-  Accesses the document generator where you can print different parts of a project
-  Builds the current project/library
-  Rebuilds the current project/library
-  Builds the current resource
-  Builds a program
-  Stops a build
-  Downloads resource code to targets
-  Switches an application to debug mode
-  Switches an application to simulation mode
-  Performs an Online Change: Download



Adds/removes dependencies



Accesses the web site

Debug Toolbar

The Debug toolbar is accessible when you run a project in either Debug or simulation mode.



Starts all stopped resources



Starts a stopped resource



Stops all running resources



Stops a running resource



Switches an application to real-time mode



Switches an application to cycle-to-cycle mode



Executes one cycle



Steps to the next **line of code or rung**



Steps into the next **line of code or rung**



Locates the current step



Sets the cycle timing



Sets or removes a breakpoint. For LD programs only.



Removes breakpoints. For LD programs only.



Shows/Hides output values. For FBD programs only.



Debugs a function block



Displays the spy variable list



Stops the debug/simulation mode



Refreshes the status of resources



Clears the output window

Window Buttons Toolbar



Switches the Workbench to the Hardware Architecture view



Switches the Workbench to the Link Architecture view



Switches the Workbench to the Dictionary view



Accesses the I/O Wiring view



Accesses the Binding window where you can create data links between resources and define the variable bindings using these links



Accesses the External Binding list window where you can define external variable bindings between producer variables of a source resource in a given project with consumer variables of a destination resource in a different project



Accesses the cross references browser

Layers Toolbar



Toggles between the link architecture view and the distribution view.



Sets the project layer to display. The available layers are Base Layer (link architecture view or hardware architecture) and 1499 Layer (distribution view).

Version Source Control Toolbar



Checks in a project, configuration, resource, or POU definition not having the read-only attribute into a version source control database

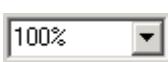


Views the history of a project, configuration, resource, or POU that has been checked in repeatedly to a version source control database

Options Toolbar



Shows or hides the data links between resources



Sets the magnification factor for the workspace

I/O Wiring Toolbar



Opens a device



Saves the I/O Wiring



Accesses the document generator



Adds a device



Deletes a device



Undoes the last operation



Redoes the last operation



Frees all I/O device channels



Frees an I/O device channel



Real/Virtual I/O device



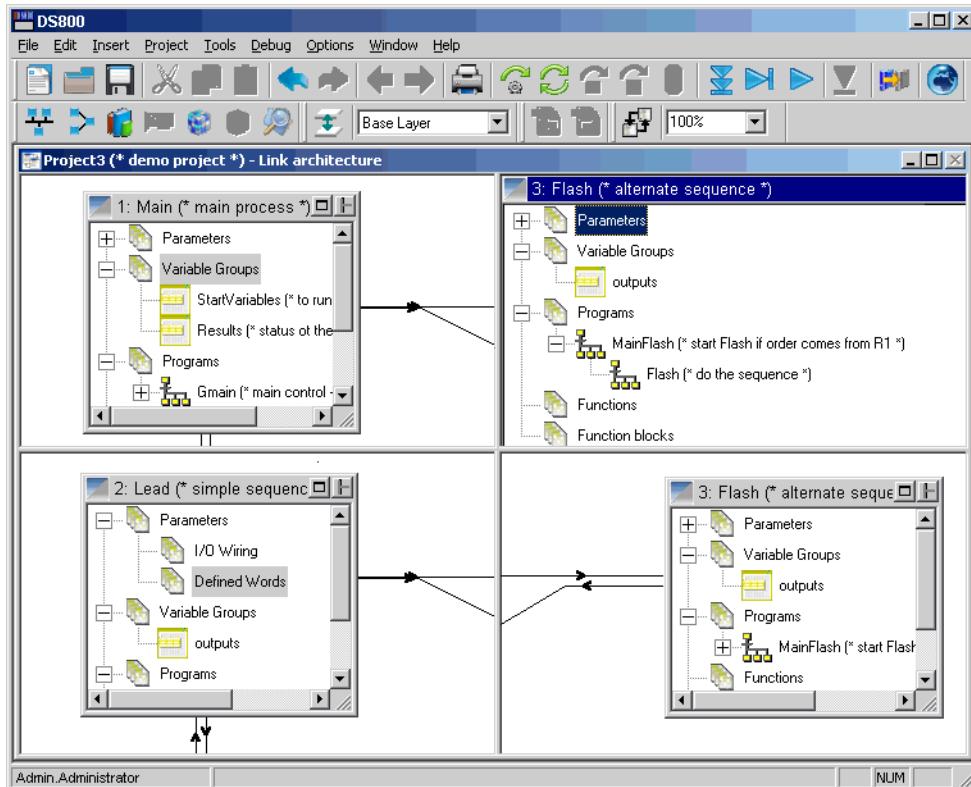
Maps logical and physical channels



Accesses help on selected I/O device in Tree view

Workspace

The Workspace can be split into a maximum of four simultaneous views:



Note: Sub-windows are zoomed independently.

To split the workspace

- Drag and drop the handles to the required positions:



Zoom

You can adjust the magnification factor, i.e., zoom, for the workspace. Elements appear with more detail as the zoom level increases. You can set the zoom from the Options toolbar or in the Layout editor. You access the Layout editor by choosing **Layout** from the Options menu.

When editing SFC, FC, LD, and FBD POU's, you can also adjust the magnification factor for the language editor's workspace.

To adjust the zoom level

1. On the Options toolbar, click the arrow of the magnification window  .
2. Choose a magnification factor from the list.

The workspace is displayed using the new magnification factor.

Output Window

The output window displays information resulting from builds of projects, resources, and programs. It also displays Workbench run-time errors. When building a program, the output window is automatically displayed. The Output window is also available from the language editors.

You can copy to the clipboard the information displayed in the output window.



To view the Output Window

- ▶ From the Window menu, choose **Show Output Window**.

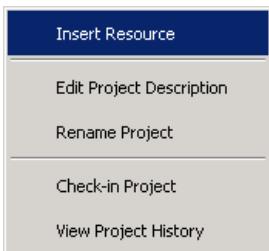
To clear the contents of the output window

- ▶ From the Window menu, choose **Clear Output Window**.

Contextual Menus

Contextual menus are displayed by right-clicking in the workspace of the various tools and applications. From the Hardware Architecture view, you can access a contextual menu for configurations or resources. For configurations, you access it by right-clicking a configuration's title bars. For resources, you access it by clicking a resource's name in the configuration window. From the Link Architecture view, you can access a contextual menu for resources by right-clicking a resource's title bar.

Example



Status Bar

A status bar appears at the bottom of the main window displaying information about commands, operations, and projects.

To show or hide the status bar

1. From the Options menu, choose **Layout**.
The Layout editor appears.
2. To show the status bar, check **Status Bar** then click **OK**.
3. To hide the status bar, uncheck **Status Bar** then click **OK**.

Customization

You can choose to customize the colors and fonts for many aspects of the Workbench as well as set working preferences. You can customize the colors and fonts for the following items:

- For the dictionary, you can set the font and the colors used for text, scope, and instances
- For the ST and IL editors, you can set the font and the colors used for background and text (basic syntax)
- For the FBD editor, you can set the font and the colors used for background, text, connection and element outline lines, line shadows, and selected elements as well as the fill for main elements
- For the LD, FC, and SFC editors, you can set the font and the colors used for background and text as well as the fill for main elements
- For the FBD and LD editors, you can set the color for comments and for Boolean values (TRUE and FALSE) displayed while in debug mode.

You can also set the colors for resource data links used with bindings.

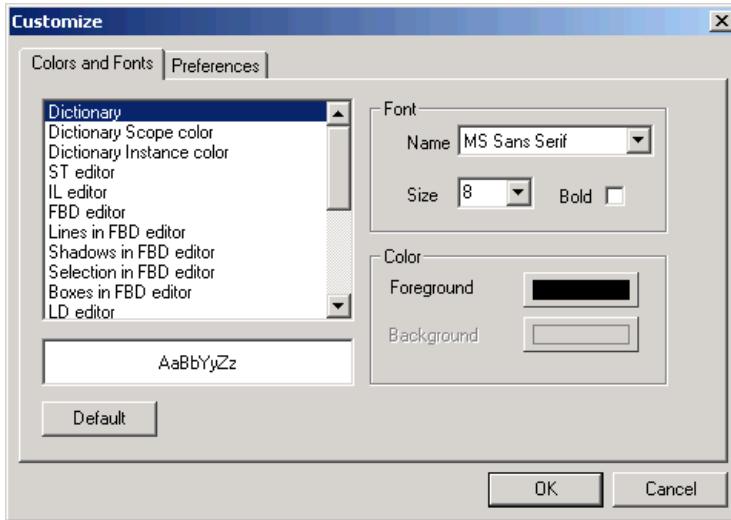
You can set the following working preferences:

- The number of recent project files to display in the File menu
- Reload the last project at startup
- Always prompt before saving changes to the project

To customize colors and fonts

Resetting the default for an item restores the colors and fonts to those when the Workbench was installed.

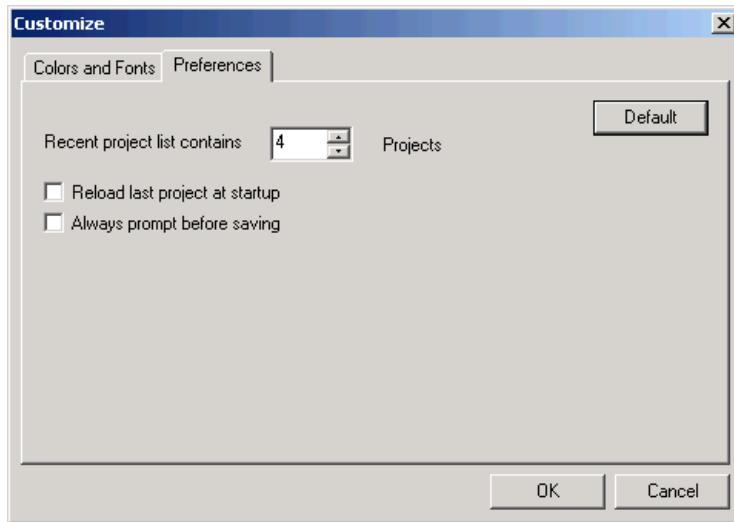
1. From the Options menu, choose **Customize**.



2. On the Customize editor, select the Colors and Fonts tab, then select the item to modify.
3. To change the font used, select a font and size. You can choose to bold the font.
4. To change the foreground or background colors, click the respective button, then from the color editor, choose a pre-defined color or specify a custom color.

To set working preferences

1. From the Options menu, choose **Customize**.
2. On the Customize editor, select the Preferences tab.



3. Make the desired changes.

Directory Structure

The installation process creates the following directory structure:

| | | |
|--|----------------------|--|
| | Emerson | Root directory for all Emerson products |
| | DS800 2.1 | DS800 Workbench files |
| | Bin | Executable files |
| | Simul | Simulator target files |
| | Tmp | Temporary files |
| | User | User profile files |
| | Projects | Emerson projects |
| | DS800 2.1 | DS800 Workbench projects |
| | Prj | Projects |
| | <project> | Individual Project Directories |
| | | A directory per hardware configuration |
| | | <resource> A directory per resource |
| | Tpl | DS800 templates |
| | EmptyLibmonoresource | EmptyLibmonoresource templates |
| | | <configuration> A directory per hardware configuration |
| | | <resource> A directory per resource |

| | | | |
|---|---|---------------------------------|--|
|  | EmptyLibmultiresource | EmptyLibmultiresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |
|  | EmptyPrjmonoresource | EmptyPrjmonoresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |
|  | EmptyPrjmultiresource | EmptyPrjmultiresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |
|  | Libmonoresource | Libmonoresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |
|  | Libmultiresource | Libmultiresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |
|  | Prjmonoresource | Prjmonoresource templates | |
| |  | <configuration> | A directory per hardware configuration |
| |  | <resource> | A directory per resource |

| | | |
|--|-----------------|---|
| | Prjmultiproject | Prjmultiproject templates |
| | <configuration> | A directory per hardware configuration |
| | <resource> | A directory per resource |
| | Shared | Common files shared by Emerson products |
| | Error Reporting | Solobug files for use when reporting errors on Emerson products |
| | Help 2.1 | Online help files for Emerson products |
| | Sentinel | Sentinel driver files for use with hardware keys |

Projects are stored in the Projects directory, as MS-Access database (.MDB) files:

<drive>:Emerson/Projects/DS800 2.1/Prj/<project name>/PRJLIBRARY.MDB

For details on the project architecture, see page 368.

Note: Existing projects can be manually moved or copied to the "tpl" directory to create new project templates.

Example

The *panel* resource in the *main* configuration within the *proj1* project is stored in the directory:

<drive>:Emerson/Projects/DS800 2.1/Prj/proj1/main/panel/

Working with Projects

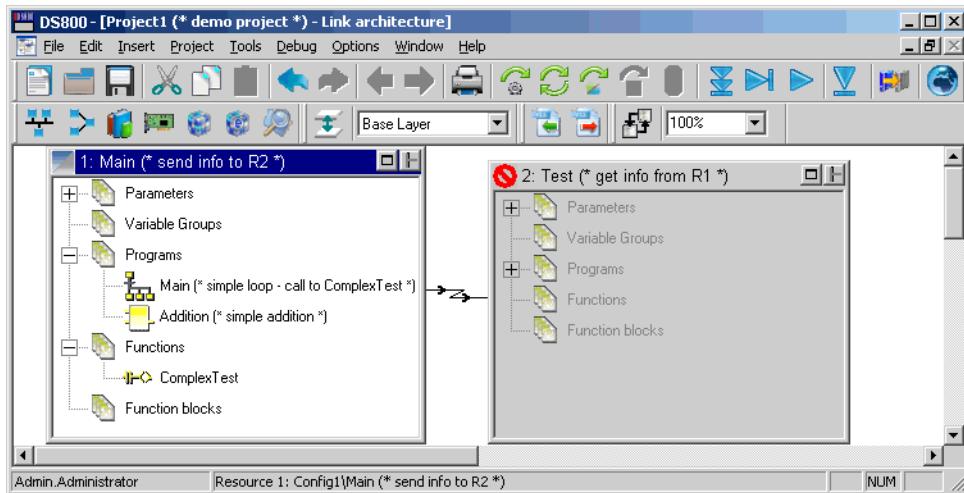
You can work with **DS800 Development Suite** projects in one of two project editing modes:

- Normal
- Single-resource

The normal mode provides access for a single user to all resources and POUs making up a project. While in the normal mode, no other users can access the project or its resources. Before opening a project in normal mode, multiple users can access the individual resources of the project for editing purposes, i.e., single-resource editing mode. The single-resource mode limits access for an individual user to one resource and its POUs. Other users can access other resources of the same project.

Note: Make sure to build the complete project in normal mode before editing single resources. Otherwise, a build while in single-resource mode may generate errors.

Only one user can access a resource at any given time; while in use, a resource remains locked to all other users. For instance, when editing a project in normal mode, all resources making up the project are automatically locked for your use except for those resources currently open in single-resource mode. The currently open resources are displayed in the workspace but remain locked. Locked resources appear gray with a lock symbol in their title bar.



You can unlock resources currently open in single-resource mode by another user by selecting the resource, then choosing **Unlock Resource** from the Tools menu.

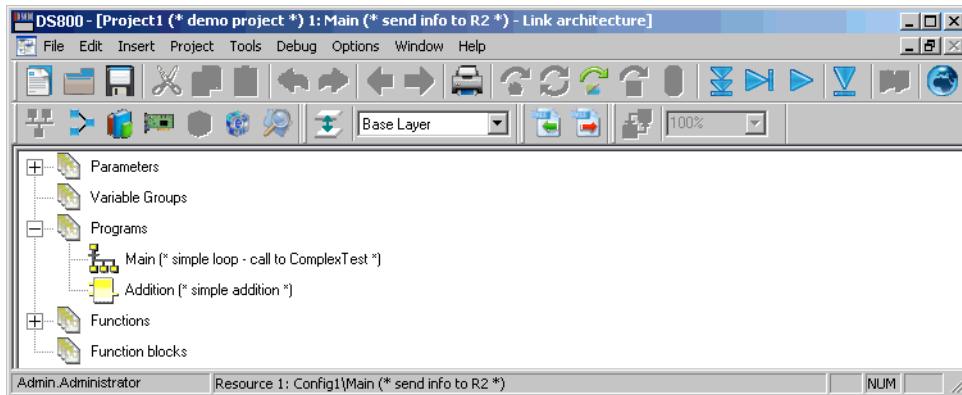
Warning: The Unlock Resource option should only be used when necessary. When unlocking resources currently opened by another user, make sure the remote Workbench is no longer running.

The Workbench automatically assigns a user name to a project, when running on a network. The user name is displayed in the status bar and in the access control properties of the resources. The assigned user name depends on the editing mode:

- In normal mode, the user name is always *UserName*.Administrator
- In single-resource mode, the user name is the Windows login user name of the user editing the resource

Resources currently opened by another user hold the name of that user in their properties.

In single-resource mode, a project is displayed in the link architecture view with the project and resource identification in the title bar of the single resource. The hardware architecture view and binding list are not available. In the dictionary view and the I/O wiring view, only the variables and wiring defined for the resource are displayed. Variables bound to other resources as well as types and defined words are in read-only mode. While in single-resource mode, you can switch a project to debug or simulation mode.



While in normal mode, you can perform the following tasks:

- Creating Projects
- Renaming Projects
- Adding a Project Description

While in the normal or single-resource project editing mode, you can perform the following tasks with limitations depending on the mode:

- Opening and Closing Projects
- Saving Projects
- Printing Projects

You can also control access to projects.

Creating Projects

When you create projects, you use one of four templates:

- SingleROC800, SingleROC800E, and SingleFB107, containing one resource in one configuration
- MultipleROC800E, containing two resources in two different configurations linked by an Ethernet network. This template is not available for use with non-networked versions of the Workbench.

The SingleROC800, SingleROC800E, and MultipleROC800E templates are all RAS device target specific. The LibSingleROC800, LibMultipleROC800, and LibSingleFB107 templates are available for use with libraries.

To create a new project

1. From the File menu, choose **New Project <Ctrl+N>**.
2. Enter the project name (max 128 characters).

3. Enter comments (optional).

4. Choose a template.

5. Click **OK**.

The project is created using the chosen **template** and the link architecture view is displayed.

You can only open one project at any given time. When changes have been made to an open project, you will automatically be prompted to save the changes before creating a new one.

Opening and Closing Projects

In the Workbench, you can only open one project at any given time. If changes have been made to an open project, the system automatically prompts you to save changes before closing a project or opening another. You can open projects in one of two project editing modes: normal and single-resource.

Project filenames are always PRJLIBRARY.MDB, the project directory name represents the given project name. When you open a project or create a new project, the hardware architecture view and the link architecture view are cleared. When a project has been relocated, you need to redefine its links within the Workbench before opening it.

When opening a project in single-resource editing mode, you need to select the project, then choose a resource from the list of resources defined for the project. In the list of resources, resources appear with icons indicating their security state or non-availability:



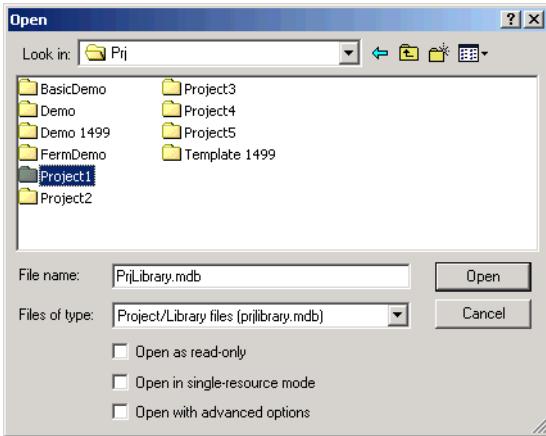
Resource unavailable, currently open by another user

When the advanced options are installed on your computer, you can choose to open a project without the advanced options features such as alarms and events, trends, field communications.

To open an existing project

1. From the File menu, choose **Open**.

The **Open Project** dialog box is displayed:

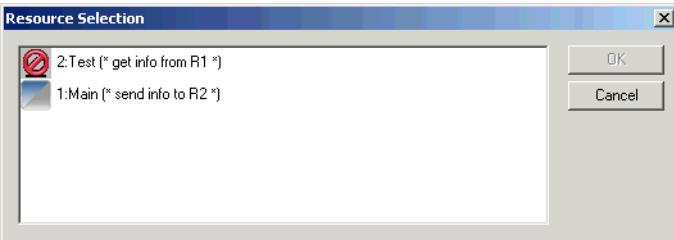


2. Locate the required project **file name**.
3. Do one of the following:
 - To open the project in normal editing mode, click **Open**.

The project is open in the normal editing mode having access to all resources and POU's.

- To open the project in single-resource editing mode, check **Open in single-resource mode**, then click **Open**.

The Resource Selection window is displayed with all project resources showing their security states or non-availability.



- From the list of available resources, select the resource to open, then click **OK**.

The project is open in the single-resource editing mode where only the selected resource is editable.

To open a project using a command line

You can open projects in single-resource editing mode or in read-only mode using a command line.

- To open the project in single-resource editing mode from a command line, use the following syntax:

DPM.exe project_path -res resource_name

Note: For command lines, resource names are case sensitive. You can also use the resource number to identify the resource.

When manually starting the Workbench, you may need to provide the location of the Workbench project. The Workbench needs to be started in its location directory. For example:

"C:\Program Files\Emerson\DS800\Bin\DPM.exe" "C:\Emerson\Projects/DS800\Prj\Project1" -res Lead

- To open the project in read-only mode from a command line, use the following syntax:

DPM.exe project_path -readonly

When manually starting the Workbench, you may need to provide the location of the Workbench project. The Workbench needs to be started in its location directory. For example:

"C:\Program Files\Emerson\DS800\Bin\DPM.exe" "C:\Emerson\Projects/DS800\Prj\Project1" -readonly

Saving Projects

The project name is used to create a unique directory structure. Saving the project saves it in the MS-Access database of the project root directory. Other files related to the project are also updated in this directory structure. When editing a project in single-resource mode, changes are only saved for the edited resource.

To save a project

- ▶ From the File menu, choose **Save Project**.

Note: When a project is saved, the undo/redo history is cleared.

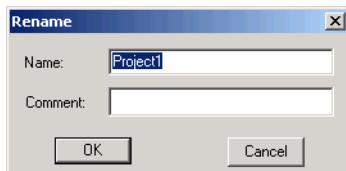
Renaming Projects

You can rename projects and edit their comments. You cannot rename projects while in single-resource editing mode. Before renaming projects, make sure to close all Workbench windows such as language editors and browsing tools.

To rename a project

1. From the File menu, choose **Rename Project**.

The Rename Project dialog box appears:



2. Change the name and comment as required.
3. Click **OK**.

The directory structure containing the project is renamed when you save changes to the project.

Adding a Project Description

You can include a text description for a project.

To edit the project description

- ▶ From the Tools menu, choose **Edit Project Description**.

Printing Projects

You can print projects using the Document Generator. For details on the Document Generator, see page 337.

To print the current Project

- ▶ From the File menu, choose **Print**.

The document generator appears with a standard list of elements to be printed for a complete project.

Project Access Control

For project security, you can control access using a password. You can also apply the read-only mode to the entire project. In read-only mode, users not having the password will have read-only access to the project. When opening a project in read-only mode, all resources and POU's making up the project are set to read-only mode. However, individual resources and POU's making up projects can have their own access control. For instance, a resource having its own password without the read-only option remains locked and cannot be viewed without its password. While in read-only mode, you cannot build a project. When importing and exporting projects having access control, password definitions are retained.

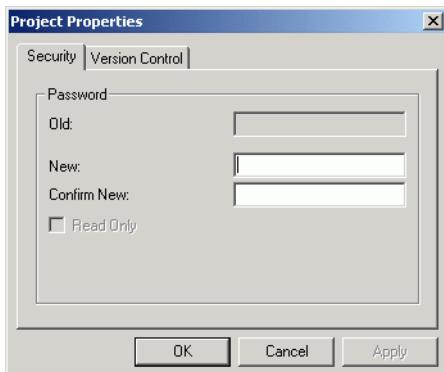
The DS800 Workbench enables monitoring a project, i.e., debugging as well as viewing TLP variables and parameters when a project is opened with access control or in read-only mode. You can open projects in read-only mode when opening the Workbench or open projects in read-only mode using a command line. While in monitoring mode, a banner indicating this mode is displayed above the status bar.

To set access control for a project

When a password is set for a project, you can choose to enable users not having the password to open the project in read-only mode. The read-only mode for a project is applied to all resources and POU's making up the project. However, individual resources and POU's may have their own access control.

1. From the File menu, choose **Project Properties**.

The Project Properties Security editor is displayed.



2. In the New field, enter the password for the project.
3. In the Confirm New field, reenter the password.
4. To enable users not having the password to open the project in read-only mode, check **Read Only**.

Importing and Exporting Workbench Elements

You can import and export Workbench elements, i.e., projects, configurations, resources, and POUs, from one project to another. When exporting an element, you copy the element from the project and create a compressed exchange file (.PXF) holding all data except for spy lists and step-by-step debug information. Therefore, enabling you to copy and paste elements from one project to another. When importing and exporting elements having access control, password definitions are retained.

To export a Workbench element

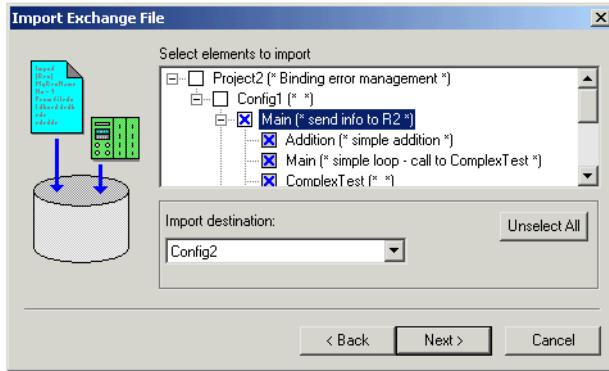
1. Depending on the element type, do one of the following steps:
 - For projects, from the File menu, choose **Export**, then **Project**.
 - For configurations, resources, and POUs, select the element (either from the link architecture or hardware architecture view), from the File menu, choose **Export**, then the element type.
2. In the Export window, select a directory in which to store the compressed exchange file, then click **Start**.
3. To close the window when the export is complete, click **Close**.

To import a Workbench element

You can only import Workbench elements that have previously been exported and stored as compressed exchange files. You cannot import elements having the same name as those in a project. Before importing an element, you can choose to create an automatic backup of your project.

1. From the File menu, choose **Import**, then **Exchange File**.
2. In the Import Exchange File window, select the *Import from file* option, then click **Next**.
3. Click **Browse** to locate the compressed exchange file (.PXF file), then click **Next**.
4. In the list at the top of the window, select the file name, then click **Next**.

- From the contents of the exchange file, select the element to import. For resources and POUs, you also need to select the import destination.



- Click **Next**.
 - Assign a name to the new element that will be created.
- Note:** Before importing elements, you should make a back-up copy of your project so that you could restore it if the resulting import is unsatisfactory.
- To create a backup copy of the project, check **Create a backup copy of the project before importing element**.

The <prjlibrary.BAK> file is created in the project folder. If the results of the import are unsatisfactory, you can choose to restore the project.

- Click **Next**.
- The element import begins.
- When the import is complete, do one of the following:
 - To import another element from the exchange file, click **Next**.
 - To exit the dialog, click **Close**.

To restore a project from a backup

1. Close the workbench.
2. Replace PrjLibrary.mdb with PrjLibrary.bak.
3. Remove (or rename) the *<element_name>* directory.
4. Rename *<element_name.BAK>* directory into *<element_name>* directory.

Uploading Workbench Elements from Targets

You can upload Workbench elements from any project into another when the resources' code has been stored on the target (if non-volatile storage exists for the platform). The element source file is compressed and contains all data for the element. The file is in the same format, compressed exchange file (.PXF), used when importing and exporting Workbench elements from one project to another. For details on importing and exporting elements, see page 43.

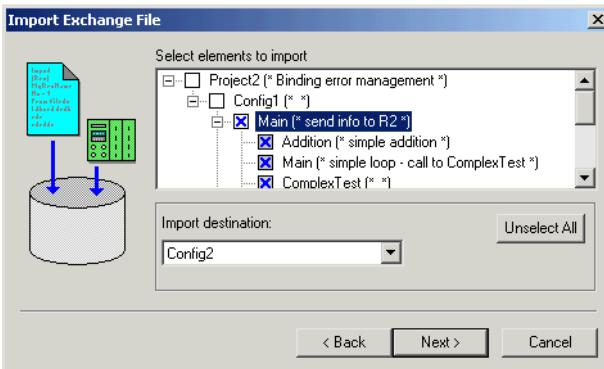
Before uploading an element's source file, you need to download its source code onto the target. Furthermore, when setting up the resource's Compilation Options properties, you need to check the **Embed Zip Source** option and select the element type.

To upload an element from sources on a target

The element upload process consists of uploading the source file containing the element from the target onto the local computer for access, then importing the element into the project from the source file. Before importing an element from an uploaded source file, you can choose to create an automatic backup for your project.

1. In the project, make sure that the configuration (in which to upload the element) is connected to the correct network with the correct connection parameters (IP Address for ETCP).
2. From the File menu, choose **Import**, then **Exchange File**.
3. In the Import Exchange File window, select the *Upload from target* option, then click **Next**.
4. From the list of available configurations, select the configuration where the required sources are located, then click **Next**.
5. From the list of available sources, select the one to upload, then click **Next**.
6. When the upload is complete, click **Next**.

7. From the contents of the exchange file, select the element to import (for resources and POUs, you also need to select the import destination), then click **Next**.



8. Assign a name to the new element that will be created.

Note: Before importing elements, you should make a back-up copy of your project so that you could restore it if the resulting import is unsatisfactory.

9. To create a backup copy of the project, check **Create a backup copy of the project before importing element**.

The <prjlibrary.BAK> file is created in the project folder. If the results of the import are unsatisfactory, you can choose to restore the project.

10. Click **Next**.

The element import begins.

11. When the import is complete, do one of the following:

- To import another element from the exchange file, click **Next**.
- To exit the dialog, click **Close**.

Link Architecture View



The **link architecture** view graphically displays the resources of a Project and the resource data links between them. This is the default view of the Workbench providing a main entry point to all editors. In the link architecture view, you manage many aspects of a project:

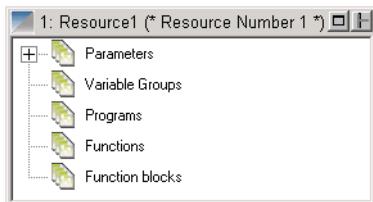
- creating resources
- linking resources (data links for bindings between resources)
- defining variable groups
- creating and manipulating POUs (Program Organization Units)
- setting up I/O wiring

To access the link architecture view

- ▶ From the Window menu, choose *project_name-Link Architecture*.

Resources

Each resource is displayed as a separate **window** within the link architecture view. The resource window title bar includes:



- An icon indicating the operative state and security state of the resource
- The resource name and comment

- A Windows control button to maximize or restore the resource window
- A Data Link button enabling you to graphically create data links between resources

The operative state of a resource is indicated by the color of the upper triangle in the resource icon:



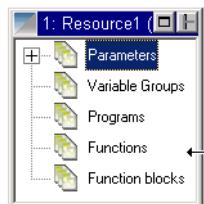
Blue. The resource is in editing mode.



Green. The resource is in real-time mode (running), debug mode, or simulation mode.

You can access the contextual menu from a resource by right-clicking in its title bar. Double-clicking the title bar minimizes/restores a resource window.

You can also resize resource windows by placing the cursor over an edge or corner until it shows double arrows and dragging:



Resource Window Workspace

The **Workspace** displays a graphical representation of the various components of each resource.

- +  Parameters
-  Variable Groups
-  Programs



Functions



Function Blocks

To expand / collapse any branch of the hierarchy

- ▶ Click '+' ('-' to collapse).

Note: Selecting one of the components changes the menu items available on the menus of the Workbench. For example, if a function is selected, the Insert menu includes the Add Function option. The contextual menus are also affected by selections within the resource window.

Creating Resources

When you create resources in the link architecture view, these resources are automatically assigned to the first configuration. You can also choose to create, i.e., insert resources directly in configurations while in the hardware architecture view. After having created resources, you can move them. For details on moving resources, see page 113.

To create a new resource

You can create resources using the main menu or a contextual menu, accessed by right-clicking the empty area of the link architecture view's workspace.

- ▶ From the Insert menu, choose **Resource**.

Renaming Resources

You can rename an existing resource by editing its properties. When a resource is selected, the Properties option is available from the main menu or a contextual menu. From the resource's Properties window, you can also edit the comments for the resource.

To rename a resource

1. Select the resource.
 2. From the Edit menu, choose **Properties**.
- The Resource Properties dialog box appears - on the General Tab.
3. Edit the resource name as required.
 4. Edit the comment as required.
 5. Click **OK**.

Copying Resources

You can copy a resource and place it on the clipboard. When copying resources, password definitions are copied, whereas, step-by-step debug information is not copied. When copying resources having The copy command is available from the main menu, the **Ctrl+C** keyboard command, the main toolbar, or a contextual menu.

Note: Before copying, click in a blank area inside the resource window to deselect individual programs or groups.

To copy a resource from the link architecture view

You can access the contextual menu by right-clicking the title bar of the resource.

1. Click on the title bar of the resource.
2. From the Edit menu, choose **Copy**.

To copy a resource from the hardware architecture view

You can access the contextual menu by right-clicking the resource in the configuration window.

1. Select the resource.
2. From the Edit menu, choose **Copy**.

Pasting Resources

You can paste a resource into the workspace of the link architecture view or into a configuration, in the hardware architecture view. When pasting resources, password definitions are also pasted, whereas, step-by-step debug information is not pasted. The paste resources using the main menu, the **Ctrl+V** keyboard command, the main toolbar, or a contextual menu.

To paste a resource in the link architecture view

You can access the contextual menu by right-clicking the title bar of the resource.

1. Click on an empty area of the link architecture view.
2. From the Edit menu, choose **Paste Resource**.

To paste a resource in the hardware architecture view

You can access the contextual menu by right-clicking the resource in the configuration window.

1. Select a configuration to paste the resource into.
2. From the Edit menu, choose **Paste Resource**.

Note: Links coming from or arriving to a resource are not copied and pasted.

Deleting Resources

You can delete a resource from the workspace of the link architecture view or from a configuration, in the hardware architecture view. The delete command is available from the main menu, the **Delete** keyboard command, the main toolbar, or a contextual menu.

Note: Before deleting, click in a blank area inside the resource window to deselect individual programs or Groups.

To delete a resource

You can access the contextual menu by right-clicking the resource title bar.

1. Select the resource.
2. From the Edit menu, choose **Delete**.

Editing Resource Properties

You need to define several properties at the resource level, intimately linked to targets (and their implementation). These properties determine the behavior of the programs and hardware such as the type of code generated, the timing, and Hardware specific properties. **To access the Resource Properties window**

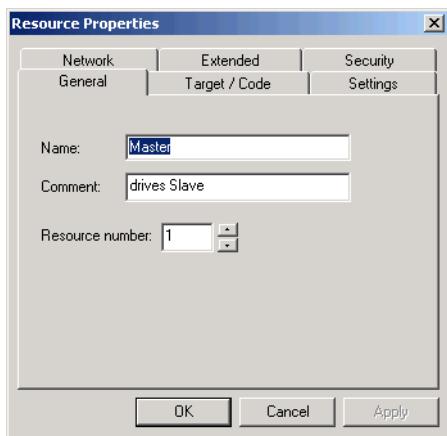
1. From the Window menu, choose *project_name*-Link Architecture.
2. Select a resource.
3. From the Edit menu, choose **Properties**.

The Resource Properties window is displayed.

Resource Identification

The resource identification properties of a resource include its name, comments, and a resource identification number. The resource number is automatically assigned. You can choose to change this number. However, when changing this number, you need to assign a number that is unique within the project. The resource number identifies the Virtual Machine that will run the resource code. Comments appear in the resource's title bar, next to its name.

You define resource identification properties on the General tab of the Resource Properties window:



Compilation Options

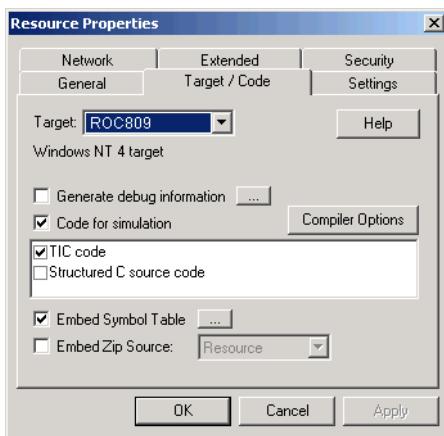
The compilation options of a resource define many aspects of a resource's run-time and simulation options:

- The target operating system on which the resource will run
- The generation of debug information
- The type of code to generate and compiler options
- The symbol table

- Embedded zip of the source files

You can also choose the target type at the configuration level. However, changing the target for a configuration affects all resources attached to it. For details on configuration properties, see page 114.

You specify a resource's compilation options on the Target/Code tab of the Resources Properties window:



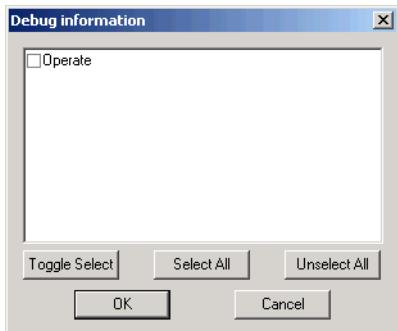
Generate Debug Information

When using the step-by-step mode, for debugging purposes, you need to generate debug information for a resource and its ST, IL, and LD POU. For details on step-by-step mode, see page 299. When setting up debug information, you also need to specify the individual POU for which to generate debug information. Debug information includes call stack information which tracks stepping between POU and called functions. You can only generate debug information for resources producing TIC code.

Note: The first time you choose to generate debug information for a resource, the complete resource is compiled when you build the resource. Subsequently, when you add or remove individual POU, only those POU are compiled when you build the project.

To generate debug information for ST, IL, and LD programs

1. To generate call stack information for use while stepping in POU, check **Generate debug information**.
2. To generate debug information, click .
3. In the Debug Information window, check individual POU, select all POU, or unselect all POU, then click **OK**.



Target Code and Compiler Options

You can specify the generation of three types of code for use in simulation or run time:

- Code for simulation, code required when running the application in simulation. To run the Simulator, you must check **Code for Simulation** to produce the application code.
- TIC Code, the indication of whether Target Independent Code is produced by the compiler. TIC code can be executed on **DS800 Development Suite** virtual machines.
- Structured C source code, the indication of whether structured C source code is produced by the compiler. Structured C source code can then be compiled and linked with **DS800 Development Suite** libraries to produce embedded executable code.

You can also define compiler options for individual resources. For details on compiler options, see page 349.

Symbol Table

You can specify whether to download the symbol table to the Virtual Machine with the resource code. The symbol table groups the variable names of the resource. You can also choose to download the complete symbol table or the reduced symbol table.

Note: The Complete Table must be selected. The **reduced symbol table** contains only names of variables for which the 'Address' cell had been completed. For details on the variables grid, see page 142.

To change the Build Symbol Table

1. Click Options.

The Build Symbol Table dialog box appears.



2. Choose the type of symbol table to download.

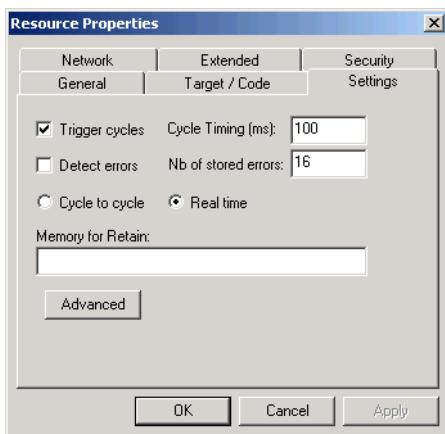
Embed Zip Source

You can embed, on the target, an exchange file (.PXF) holding all data from Workbench elements. This exchange file is the same as the file created when exporting an element.

Run-time Settings

The run-time settings include the cyclic and behavior definitions of a resource when the resource is executed. For information about execution rules, see page 376.

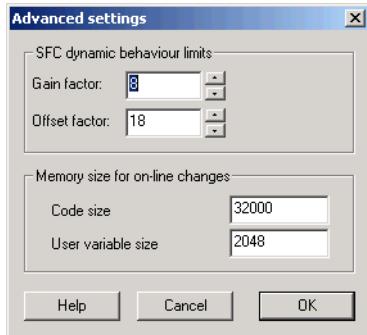
You specify a resource's run-time settings on the Settings tab of the Resources Properties window:



- Trigger cycles, enables you to specify the cycle timing, i.e., the amount of time given to each cycle. If a cycle is completed within the cycle time, the system waits until the cycle time has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the POU of the Workbench resource, then updating physical outputs. The virtual machine executes the resource code according to the execution rules. For details about the execution rules, see page 376.
- Detect errors, enables the storing of errors. You need to define the number of entries, i.e., the size of the queue (FIFO) in which detected errors are stored.
- Cycle to Cycle / Real Time, indicates whether programs are executed during the cycle or not. For Cycle to Cycle, inputs are read but the code is not executed during the cycle time. This option is useful for testing I/Os.
- Memory for Retain, indicates the location where retained values are stored (the required syntax depends on the implementation)

You can also specify advanced settings for resources:

- SFC dynamic behavior limits
- Memory size for online changes



To access advanced settings

- Click **Advanced**.

SFC Dynamic Behavior Limits

The SFC dynamic behavior limits determine the amount of memory, allocated by the target at initialization time, to manage SFC dynamic behavior (i.e. token moving). The amount of allocated memory is calculated as a linear relation with the number of SFC POU:

`Alloc Mem (bytes) = N * NbElmt * sizeof(typVa)`

`NbElmt = GainFactor * NbOfSFC + OffsetFactor`

Where:

$N = 5$ (constant linked to SFC engine design)

$typVa = 16$ bits in the medium memory model (32 bits in the large memory model)

$NbElmt$ represents for each executed cycle:

- The maximum number of transitions that can be valid. That is to say transitions with at least one of their previous steps being active.

A simpler, but more approximate definition is:

- The maximum number of steps that can be active.
- The maximum number of actions that can be executed.

Here, an action refers to an N, P1 or P0 action linked to a step.

If the available memory is not enough at a specific moment:

- If the target is generated with check mode (ITGTDEF_SFCEVOCHECK defined in dsys0def.h), The target kernel generates a warning to signal an SFC token moving error or an action execution error and the resource is set in ERROR mode (i.e. cycles are no longer executed). Otherwise, kernel behavior may be unpredictable.

Memory Size for Online Changes

The memory size for online changes defines the amount of memory that is reserved on the PLC for managing online changes:

- Code Size, the amount of memory reserved for code sequence changes
- User Variable Size, the amount of memory reserved for adding variables data. When generating symbol monitoring information for a POU, the same amount of memory is also reserved for the POU.

When performing downloads and online changes, parts of the User Variable Size memory space is used to store project data such as variables values. This memory space becomes free when you clean the project.

Resource Network Parameters

You need to define network parameters attached to the resource for each available network.

You specify a resource's network parameters on the Network tab of the Resources Properties window:



Note: The parameters appearing in the list reflect those attached to the resource. Some parameters are read-only. However, when a resource is attached to a network not requiring parameters, the list appears empty.

You can also access the online help by clicking **Help**.

For the HSD network, the current definition is the following:

The consumer computes the time elapsed between production and consumption and tests if it is less than the 'ValidityTime' parameter specified for the producer resource in the workbench. The user must be careful when specifying this value to take into account the cycle time of the producer resource. This resource cannot emit values at a period shorter than its cycle time.

If this time-out is detected, the consumer sets the error variable to ISA_HSD_KVB_ER_TIMEOUT value.

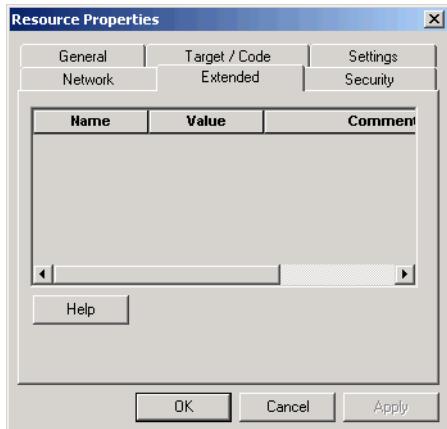
For the ETCP network, the current definition is the following:

On the consumer side, if no data is received during the time specified in the Timeout parameter value, then the error variable is set to ETCP_KVB_ERR_TIMEOUT value.

Custom Resource Parameters

You can define specific **OEM** options for a resource that may be implemented in your target.

Note: DS800 Development Suite standard targets do not have extended parameters. Contact your target supplier for specific details.



Resource Access Control

For resource security, you can control access using a password and you can choose to apply the read-only mode to an entire resource. When resources are password-protected, users not having the password can change resource properties, wire and bind variables, modify the memory for retain, and add devices to wired variables. POU's in a resource can have their own level of access control. For instance, a POU having its own password remains locked and cannot be viewed without entering its password. However, a POU using its resource's password also inherits the resource's security properties such as the read-only attribute.

The security state of a resource is indicated by the color of the lower triangle in the resource title bar icon. The resource can also be currently opened by another user.

| Resource Icon | Security State |
|---------------|--|
| | Gray. The resource has no access control. All users have read and write access in the resource. POU's in the resource may have individual access control. |
| | Red. The resource is locked. Users not having the resource password cannot access the resource or its POU's; these users do not have read or write capabilities. These users can change resource properties, wire and bind variables, modify the memory for retain, and add devices to wired variables. |
| | Cyan. The resource is in read-only mode. Users not having the resource password can view the resource and its POU's; these users only have read capabilities. These users can change resource properties, wire and bind variables, modify the memory for retain, and add devices to wired variables. POU's in the resource may have individual access control. |
| | Green. The resource is unlocked. User can access the resource and its POU's; this user has read and write capabilities. However, POU's in the resource may have individual access control. |
| | The resource is currently opened by another user in single-resource project editing mode. User can only access the resource properties in read-only mode. |

Note: While in debug mode or performing builds, unlocked resources as well as resources having no access control switch to read-only mode. Locked resources remain locked.

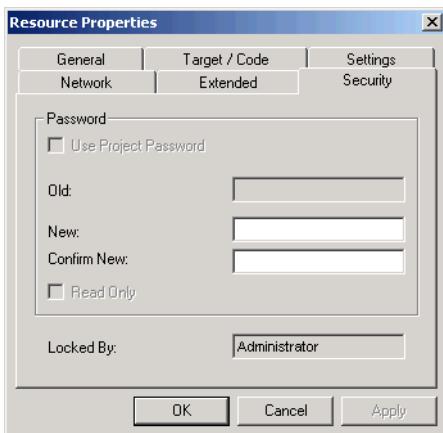
For projects having read-only access control, the resources and POU's making up the project are also set to the read-only mode except for those having individual access control.

When copying, pasting, importing, and exporting resources having access control, password definitions are retained.

When editing a project, resources making up the project are automatically locked by you except for those resources where another user set password protection or that are currently in use by another user in the single resource editing mode.

To set access control for a resource

You set access control for a resource in its properties' Security tab.



1. Specify a password:
 - To use an unique password, in the New field, enter a password then reenter it in the Confirm New field.
 - To use the same password as set for the project, check **Use Project Password**.
2. To enable all users to access the resource in read-only mode, check **Read Only**.

To unlock a resource

When entering a password while in debug mode or performing a build, the resource is only unlocked after stopping the debug mode or when the build is completed.

1. Right-click the resource's title bar, then from the contextual menu, choose **Enter Password**.
2. In the Security dialog box, enter the password for the resource.



The resource is unlocked.

Resource Description

You can include a free-format text description for a resource.

To edit the resource description

1. Select the resource.
2. From the Tools menu, choose **Edit Description**.
3. Edit the description as required.

Variable Bindings

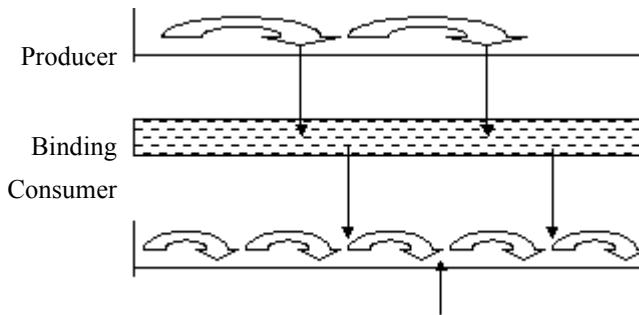
Bindings are directional links, i.e., access paths, between variables located in different resources. One variable is referred to as the producing variable and the other as the consuming variable. The value stored in the producing variable is transferred to the consuming variable. The Workbench enables two types of bindings: internal bindings and external bindings. Internal bindings are between resources within the same project. External bindings are between resources belonging to different projects.

Note: Online changes are possible as long as internal and external binding definitions remain the same.

Binding the variable V1 from resource R1 to the variable V2 of resource R2 means that V1 is periodically copied to V2 using memory sharing or network exchanges.

Variables coming from bindings (consumed variables) are refreshed in the resource at the beginning of the cycle, each time the producing resource sends them, i.e. on each end of the producing resource cycle.

The variable is not updated in the consuming resource until the producing resource sends them through the binding media. For example:



DS800 Development Suite does not impose the read-only accessibility for consumed variables. However, it is highly recommended to declare consumed variables with read-only attribute in order to avoid conflicts between Binding and execution of POU's.

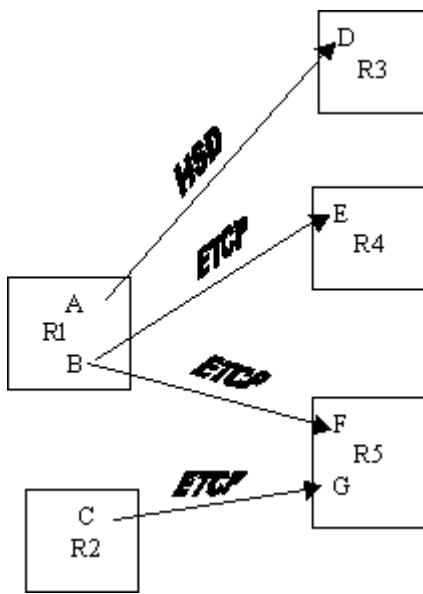
This behavior is applied in both HSD and ETCP Binding drivers. This behavior may change when using other network drivers implemented according to different conventions.

Binding error variables

Binding error variables enable the management of binding errors at the consumer resource level; one error variable for one consumer resource for each resource that produces to this resource. The virtual machine gives specific values to these error variables.

Note: DS800 does not support producer error variables.

Example



Production errors

The variable 'A' of the R1 resource represents the producer error variable for all binding links starting from R1 and using the HSD driver
(in the example only link from R1 to R3).

The variable 'B' of the R1 resource represents the producer error variable for all all binding links starting from R1 and using the ETCP network
(links from R1 to R4 and from R1 to R5).

Consumption errors

The variable 'F' of the R5 resource represents the consumer error variable for the unique binding link that comes from R1 and using ETCP.

The variable 'G' of the R5 resource represents the consumer error variable for the unique binding link that comes from R2 and using ETCP.

Depending on the driver used the error variables can take different values with different meanings.

Warning: Once the error variable is set to a non-zero value, it has to be reset to 0 by user or by Programs.

To test globally that there is a binding error, you can test the value of the following system variables:

- `_SYSVA_KVBPERR`: for a production error. It is a Boolean variable. If it is true it means there is a production error. DS800 does not support the `_SYSVA_KVBPERR` system variable.
- `_SYSVA_KVBCERR`: for a consumption error. It is a Boolean variable. If it is true it means there is a consumption error.

For HSD:

To test values of one binding error variable, you should create the following defined words in the dictionary of your project:

The 0 value in the error variable indicates there is no error.

| | | |
|--------------------------------------|---|--|
| <code>ISA_HSD_KVB_ER_MUTEX</code> | 1 | An error occurred with semaphore management |
| <code>ISA_HSD_KVB_ER_SPACE</code> | 2 | An error occurred with memory space access |
| <code>ISA_HSD_KVB_ER_NOKERNEL</code> | 3 | The bound producer is stopped (not running). This error happens only for consumer resources. |
| <code>ISA_HSD_KVB_ER_TIMEOUT</code> | 4 | Variable was not refreshed within the maximum time allowed (ValidityTime). This error happens only for consumer resources. |
| <code>ISA_HSD_KVB_ER_BAD_CRC</code> | 5 | Producer and consumer have different CRC. |
| <code>ISA_HSD_KVB_ER_INTERNAL</code> | 6 | Internal error |

For ETCP:

To test values of binding error variables, you should create the following defined words in the dictionary of your Project:

A value of 0 in the error variable indicates no error.

| | | |
|---------------------------------|---|---|
| ETCP_KVB_ERR_BINDING_IN_PROCESS | 1 | The binding initialization process is on its way. |
| ETCP_KVB_ERR_NO_PRODUCER | 2 | The remote producer is not currently running. This error happens only for consumer resources. |
| ETCP_KVB_ERR_BAD_CRC | 3 | Producer and consumer have different CRC. |
| Obsolete error value | 4 | The producer has been stopped. This error happens only for consumer resources. |
| ETCP_KVB_ERR_DATA_DIFFUSSION | 5 | Error during diffusion process. |
| ETCP_KVB_ERR_TIMEOUT | 6 | ETCP server does not answer quickly enough (TimeOut). This error happens only for consumer resources. |
| ETCP_KVB_ERR_IMPOSSIBLE_TO_BIND | 7 | Impossible to bind. |

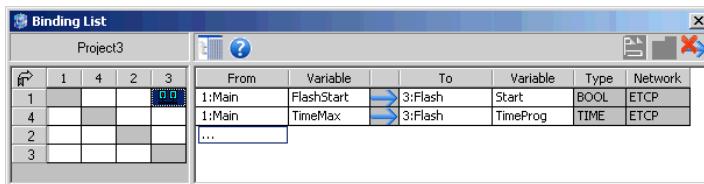
Internal Bindings

Internal variable bindings are bindings between variables of resources belonging to the same project. Before creating internal bindings for variables, you need to link the resources holding them using data links.

You manage resource data links and internal variable bindings in the Bindings List window. You can also manage resource links directly from the link architecture view.

The Bindings window displays the resource links and internal variable bindings defined for a project. The window is divided into three parts:

- The Resource-binding grid
- The Variable-binding grid
- The Binding List window toolbar



To access the Binding List window

1. From the Window menu, choose *project_name-Link Architecture*.

The link architecture view appears displaying existing resources and their data links.

2. Do one of the following:



- Click the icon.
- Double-click a data link joining two resources.

Resource-binding Grid

The Resource-binding grid, on the left side of the Binding List window, displays the data links between resources. The first column and the top row display the available resource's numbers. The resource display order depends on their configuration numbers.

| From | 1 | 2 |
|------|---|---|
| 1 | | 1 |
| 2 | 2 | |

When working in the Resource-binding grid, you can perform various tasks using the mouse or keyboard commands:

| Description | Mouse | Keyboard |
|-------------------------------------|---|----------------------|
| Move into the grid | Select cells | Arrows keys |
| Select an entire row | Select a row header | Shift+space bar |
| Select an entire column | Select a column header | Ctrl+space bar |
| Select the entire grid | Select an arrow on the top left of the grid | Shift+Ctrl+space bar |
| Switch to the Variable-binding grid | Click on the Variable-binding grid | Tab |

Variable-binding grid

The Variable-binding grid, on the right side of the window, enables you to manage variable bindings. The variable-binding grid manages the bindings between variables. The grid shows where a binding comes from and where it goes to, the type of the variable, and the network used for communicating.

| From | Variable | To | Variable | Type | Network |
|--------|------------|---------|----------|------|---------|
| 1:Main | LeadStart | 2:Lead | Start | BOOL | ETCP |
| 1:Main | TimeMax | 2:Lead | TimeProg | TIME | ETCP |
| 1:Main | FlashStart | 3:Flash | Start | BOOL | ETCP |
| 1:Main | TimeMax | 3:Flash | TimeProg | TIME | ETCP |

The column between the variable information indicates the status of the binding:



The binding does not have parameters and the status is OK



The binding does not have parameters but the status is bad

 The binding has parameters and the status is OK

 The binding has parameters but the status is bad

A bad status occurs when the types, string sizes, or dimensions of variables don't correspond or if the network, used for the binding, doesn't exist.

When working in the Variable-binding grid, you can perform various tasks using the mouse or keyboard commands:

| Description | Mouse | Keyboard |
|-------------------------------------|------------------------------------|-------------|
| Move into the grid | Select cells | Arrows keys |
| Switch to the Resource-binding grid | Click in the Resource-binding grid | Tab |
| Resource-binding grid | | |

Binding List Toolbar

The Binding List window toolbar enables you to perform many resource link and variable binding operations:



Hides the resource-binding grid



Accesses the online help



Creates a new binding variable



Edits an existing binding variable. This operation is only available for use in the variable-binding grid.



Deletes selected cells, rows, or columns

Note: If the two resources are distant, they must be located in configurations that are attached to the same target. Heterogeneous bindings are not yet supported.

Linking Resources

You need to link resources before binding variables belonging to them. Data links between resources are directional. All bindings using a data link must use the same network. You can link resources from the Binding List window. You can also link resources from the link architecture view.

In the Resource-binding grid, you create links between resources by locating the resource holding the producing variable in the first column and the resource holding the consuming variable in the top row, then selecting the grid cell where both meet.

| 1 | 1 | 2 |
|---|---|---|
| 1 | | |
| 2 | | |

In the grid, resource links appear as one of two types:



The linked resources belong to the same configuration



The linked resources belong to different configurations

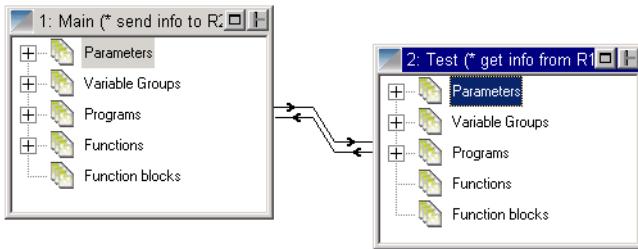
In the link architecture view, you create links by physically joining the resource holding the producing variable with the resource holding the consuming variable. In this view, data links appear as directional arrows linking the resources. The color of data links depend on the type of bindings using it:

Black The data link is only used for internal bindings

Blue The data link is used for internal bindings

When bindings have an error, the symbol is displayed on the data link used by the binding.

You can customize the colors of resource data links.



To link resources from the Binding List window

You can access the Binding list window from the main menu or the Windows toolbar.

1. From the Project menu, choose **Binding List**.

The Binding List window appears.

2. In the first column of the Resource-binding grid, locate the resource number of the resource holding the producing variable.
3. In the top row of the Resource-binding grid, locate the resource number of the resource holding the consuming variable.
4. Double-click the grid cell where both resource numbers meet.

An icon appears in the grid cell indicating whether the link is between resources from the same or different configurations.

To link resources from the link architecture view

1. On the resource holding the producing variable, click and hold the Data Link button , located on its title bar.
2. Drag the link to the resource holding the consuming variable.
3. Release the mouse button.

The data link is displayed graphically.

Deleting Resource Links

You can delete links between resources, i.e., data links, from within the Binding list window. You can also delete links from the link architecture view.

Note: Deleting a resource link also deletes the variable bindings using it.

To delete a resource link from the Binding List window

1. In the Resource-binding grid, select the grid cell holding the resource link to delete.
2. Do one of the following:

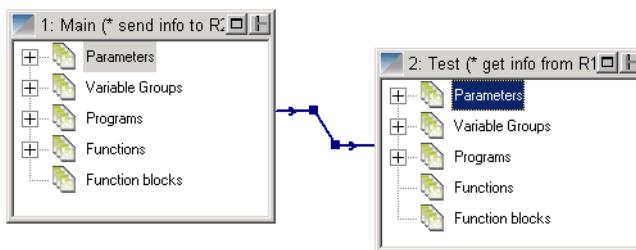
- From the Binding List window's toolbar, click .
- Press **Delete**.

The grid cell appears blank.

To delete a resource link from the link architecture view

1. In the link architecture view, click on the resource link.

The selected data link appears highlighted:



2. Do one of the following:

- From the Edit menu, choose **Delete**.
- Press **Delete**.

Viewing the Internal Bindings Defined for Resources

You can view all producing variable bindings or all consuming variables defined for a resource at the same time. You can also choose to view all bindings defined for all resources, i.e., the entire project. However, when viewing bindings, you cannot edit their definitions.

To view the producing variable bindings for a resource

- In the first column of the Resource-binding grid, click the corresponding resource number.

To view the consuming variable bindings for a resource

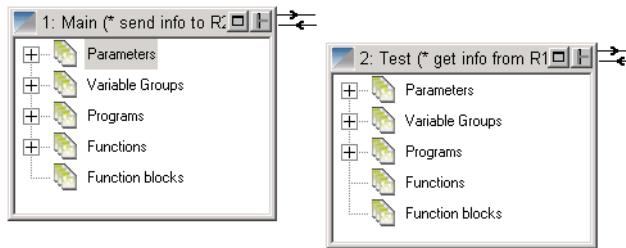
- In the top row of the Resource-binding grid, click the corresponding resource number.

To view the bindings defined for a project

- In the Resource-binding grid, click .

Hiding and Showing Resource Links

In the link architecture view, you can choose to show or hide the data links between resources. In hidden mode, links cannot be activated or selected. Links appear as short arrows, indicating their direction, sticking out from the top right corner of resources:



To hide or show data links

You can hide or show data links using the main menu or the Options toolbar.

- From the Options menu, choose **Hide/Show Links**.

Defining Internal Variable Bindings

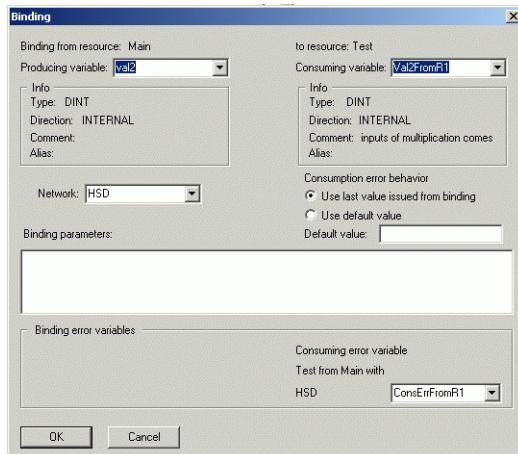
Before defining bindings between variables, you must first link the resources to which they belong. For bindings, one variable is known as the producing variable and the other as the consuming variable.

You can only define bindings between variables of a same type. Producing variables can have any direction attribute, i.e., input, output, and internal. Whereas, consuming variables can only have the output or internal attribute and must also have the Free attribute.

You instantiate variable bindings in the Variable-binding grid of the Binding List window.

| From | Variable | To | Variable | Type | Network |
|--------|------------|---------|----------|------|---------|
| 1:Main | LeadStart | 2:Lead | Start | BOOL | ETCP |
| 1:Main | TimeMax | 2:Lead | TimeProg | TIME | ETCP |
| 1:Main | FlashStart | 3:Flash | Start | BOOL | ETCP |
| 1:Main | TimeMax | 3:Flash | TimeProg | TIME | ETCP |

You define variable bindings in the Binding editor. When defining a binding, you need to indicate a producing variable and a consuming variable, and the network used for communicating. The producing variable serves as input for the binding. Whereas, the consuming variable serves as output. You can choose to specify a default value to use in the event of a communication error. You can also choose to specify a binding error variable (producer error variables are not supported in DS800).



The Producing Variable selection list contains all variables of the producer resource. The Consuming Variable selection list contains all variables which do not have the INPUT direction and are not already used as a consumed variable in an existing binding.

The network selection list contains the networks that are supported by the target of the configuration of the first resource and the target of the configuration of the second resource.

The Binding Parameters list displays the parameters to be defined for the variables bound on the selected network. This list may be empty depending on the network used by the binding. For example: ETCP does not need any parameters at this level. You define parameters by double-clicking on a parameter line to display the Binding Parameter dialog box (available only when the parameter is editable). Some parameters are read-only.

The Binding Error Variables section contains two selection lists for selecting a variable (Global / Memory of the DINT type) in each resource to receive binding error values (producer error variables are not supported in DS800). Producer error variables and consumer error variables can be used in the resource's POUs to trap and act upon errors. The default value is None.

To create a binding between variables

1. In the Variable-binding grid, select the next available field.
2. From the Binding List window's toolbar, click .
The Binding editor appears.
3. In the Producing Variable field, select the producing variable.
4. In the Consuming Variable field, select the consuming variable.
5. In the Network field, select the network used for communicating.
6. To set a default value for use in the event of a communication error, select **Use Default Value**, then enter a value in the field.
7. To use a binding error variable, indicate it in the Binding Error Variables section (producer error variables are not supported in DS800).
8. Click **OK**.

Editing Internal Variable Bindings

You can change the contents of existing variable bindings. You edit bindings from the Variables-binding grid of the Binding List window.

To edit the contents of an existing binding

1. In the Variable-binding grid, select the variable binding.
2. From the Bindings List window's toolbar, click  .
The Binding editor appears.
3. Make the necessary changes, then click **OK**.

Deleting Internal Variable Bindings

You delete variable bindings from the variable-binding grid of the Bindings List window.

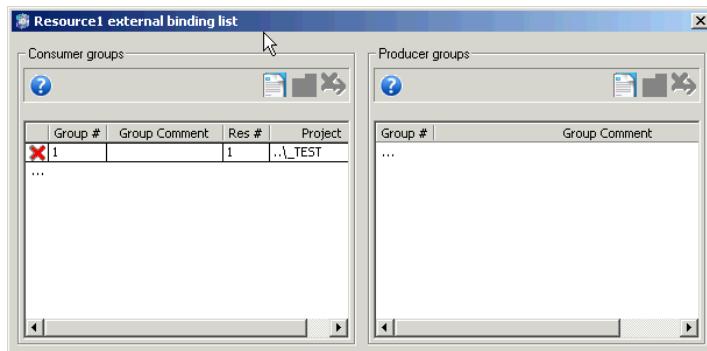
To delete a variable binding

1. In the Variable-binding grid, select the variable binding.
2. Do one of the following:
 - From the Binding List window's toolbar, click  .
 - Press **Delete**.

External Bindings

External variable bindings are bindings between variables of resources belonging to different projects. When defining external variable bindings, you need to define groups of producer variables in the producer project, then in the consumer project, link the consumer resource with the producer resource, then define the external bindings between the producer variables and the consumer variables.

You define external bindings from the External Bindings List. This list is made up of the Consumer groups and Producer groups sections. The Consumer groups section lists the groups holding external producer variables having bindings with consumer variables defined in the project. The Producer groups section lists the groups holding outgoing producer variables for consumption in external bindings defined in another project.



When defining producer groups of variables, you group variables of a resource to be consumed by consumer variables of one or more resources located in other projects. Individual variables of a resource can belong to a one or more producer groups.

For producer groups and external bindings, indicates errors that can occur for different situations such as the following:

- | | |
|-------------------|--|
| Producer groups | <ul style="list-style-type: none">- The project of a producer group cannot be found- The producer group cannot be found within the specified project- A conflict exists between the consumer and producer resources- One of the bound producer variables no longer exists |
| External bindings | <ul style="list-style-type: none">- The producer variable used in the binding no longer exists- The project holding the producer variable cannot be accessed |

External Binding List Toolbar

The External Binding List window toolbar enables you to perform many external binding operations. The operations performed by the toolbar items differ depending on whether these are in the Consumer or Producer groups sections.



Accesses the online help



In the Consumer groups section, accesses the Consumer Binding list where you define links between resources and define external bindings from the Bindings editor.

In the Producer groups section, accesses the Producer Binding list where you define groups of producer variables for use in bindings with consumer variables of other projects.



In the Consumer groups section, edits an existing external variable binding.

In the Producer groups section, edits an existing group of producer variables.



In the Consumer groups section, deletes an external variable binding.

In the Producer groups section, deletes a group of producer variables.

To access the External Binding list

1. From the link architecture view, select the resource for which to create producer groups.
2. From the Project menu, choose External Binding List or click , from the Window Buttons toolbar.

Defining Producer Variable Groups

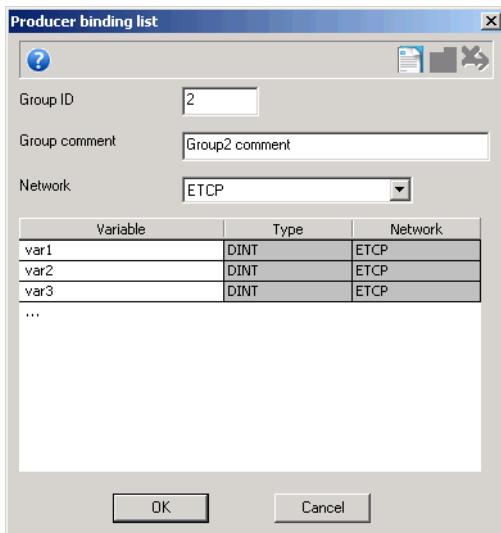
You define producer groups of variables for resources of a project to enable accessing them when defining external variable bindings in consumer resources of other projects. Producer groups hold producer variables of a resource to be consumed by consumer variables of one or more resources located in other projects.

To define a producer variable group

You define producer variable groups from the External Binding List.

1. In the Producer Groups list, double-click ...

The Producer Binding List editor is displayed:

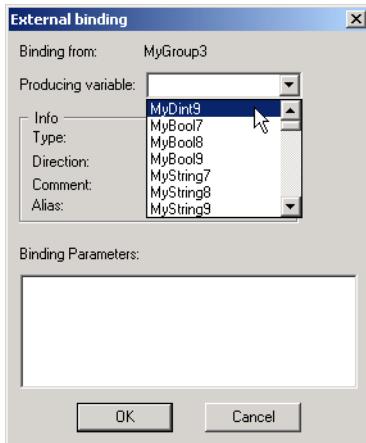


2. Enter a unique Group ID and optional comment.
3. Specify the network used for the bindings.

4. Specify each variable making up the producer variable group:

- a) In the variable list, click ...

The External Binding editor is displayed.



- b) In the Producing variable field, select the variable to include in the group.

The selected variable's information is displayed as well as the default network binding parameters. You can edit the values for the network binding parameters.

- c) Click **OK**.

Editing Producer Variable Groups

You edit the contents of an existing producer variable groups from their originating resource.

To edit the contents of an existing producer variable group

1. In the Producer Groups list, select a producer group.
2. From the Producer's Group binding list toolbar, click .
3. In the Producer Binding List editor, make the necessary changes, then click **OK**.

Deleting Producer Variable Groups

When deleting producer groups having producer variables used in external bindings, in the consumer project, the link between the consumer and producer resource shows an error ().

To delete a producer group

- In the Producer Groups list, select the producer group to remove, then from the Producer's Group binding list toolbar, click .

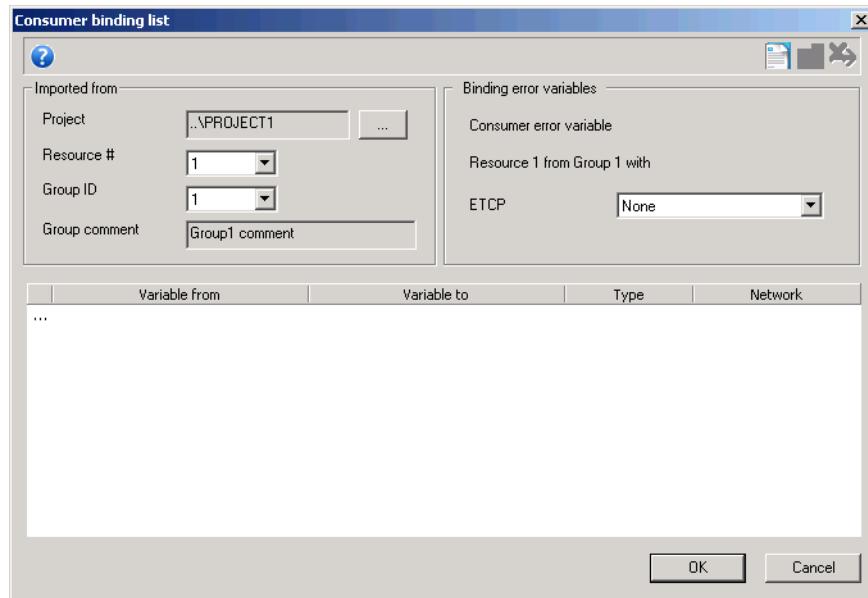
Linking Resources for External Bindings

Before defining external variable bindings between resources, you need to link the consumer and producer resources. You link these resources from the consumer resource by identifying the project, resource, and producer group of variables, holding the producer variables whose values are conveyed to the consumer variables. A link between resources flows in one direction. You can choose to use binding error variables.

To link resources for external variable bindings

1. From the External Binding List window, double-click ... in the Consumer Groups section.

The Consumer Binding List is displayed.



2. Define the source of the producer group of variables used for the external bindings.
 - a) Browse for the project holding the producer group to include in the bindings.

- b)** Specify the resource and producer group ID of the producer group.

The resource number, producer group ID, and network used for linking the resources are displayed in the Binding error variables section.

3. To indicate a binding error variable, select one from the available variables.

Editing External Resource Links

You can edit resource links for existing external bindings from the consumer resource.

Warning: Editing the linking information for a specified producer group, resource, or project sets all defined external variable bindings in the list in error (X).

To edit a link between resources of different projects

You edit links between resources from the Consumer Binding List editor.

- In the Consumer Binding List, make the necessary changes to the project, resource, or producer group information, then click **OK**.

Defining External Variable Bindings

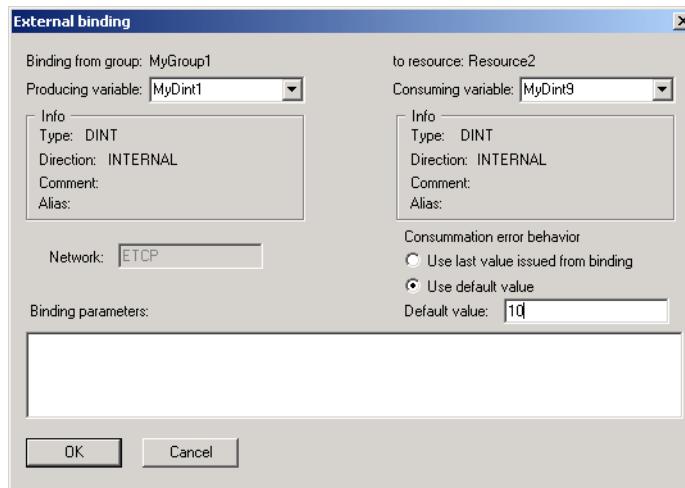
Before defining external variable bindings, for the producer resources, you need to define producer groups, holding the producer variables for consumption by the consumer resources, and link the consumer and producer resources.

To define an external variable binding

You define external variable bindings from the consumer resource.

1. In the Consumer Binding List, double-click ...

The External Binding editor is displayed.



2. Specify the producing variable and consuming variable from their respective lists of available variables.
3. Indicate whether to use the last value issued from the binding or a default value. When using a default value, specify the value to use.
4. Click **OK**.

Editing External Variable Bindings

You edit the contents of existing external variable bindings from the consumer resource.

To edit an existing external binding

1. In the External Binding List window, select the binding link to edit in the Consumer groups section.
2. From the Consumer Group's binding list toolbar, click .
3. In the Consumer Binding List editor, select an external binding from the list, then click .
4. In the Binding editor, make the necessary changes.

Deleting External Variable Bindings

You delete external bindings from the consumer resource.

To delete an existing external binding

1. In the External Binding List window, select the binding link to edit in the Consumer groups section.
2. From the Consumer Group's binding list toolbar, click .
3. In the Consumer Binding List editor, select an external binding from the list, then click .

Parameters

The 'Parameters' component contains the IO Wiring and 'Defined words' sub-components. For details on I/O Wiring, see page 155. For details on the Defined Words Tree, see page 130.

-  Parameters

-  I/O Wiring

-  Defined Words

I/O Wiring Double-clicking on this item opens the I/O Wiring Tool to select I/O devices and connect variables to them.

Defined Words Double-clicking on this item opens the Dictionary on the Defined Words Tree.

Variable Groups

Variables Groups provide a method of managing variables and logically sorting them within a resource. The variable groups are shown in the Variables Tree, their contents are defined within the Dictionary Variables grid. For information on the variables tree, see page 127.

Creating Variable Groups

You create variable groups from within the link architecture view. You can rearrange the order of defined variable groups by dragging and dropping within the variable groups section of a resource window. The group order affects the printing order.

To create a new variable group

1. From the Window menu, choose *project_name-Link Architecture*.

The link architecture view appears displaying all resources and data links defined for a project.

2. Select a resource.
3. From the Insert menu, choose **Add Variable Group**.

Opening Variable Groups

Opening a variable group opens the Dictionary with the grid showing variables of that group. You open variable groups from within the link architecture view. For information on the Dictionary, see page 125.

To open a variable group from the link architecture view

1. Select a group.

2. Do one of the following steps:
 - From the Edit menu, choose **Open**.
 - Within a resource window, double-click on the required variable group name.
 - Select a group name then press **Enter**.

To open a variable group from the Dictionary view

1. Select the Variables Tree.
2. Double-click the resource name to which the group belongs.
3. Click on the variable group name.

The grid displays the variables for that particular group.

Importing or Exporting Variables

You can choose to import or export variables data such as hundreds of trends, alarms, or events using either a comma-separated (CSV) file in a text editor or a Microsoft Excel spreadsheet. To include comments in your data, surround them with quotation marks ("").

When using a text editor, you must separate each piece of information from the others with a comma; each line must end with a carriage return; the resulting file can have either the .csv or .xls extension. When using a spreadsheet, enter each piece of information in a separate cell; leave empty cells if an item is to be omitted; save the file under the CSV or XLS format. These requirements are automatically followed by the export facility; you must respect them if you build a file to be imported. For variables data, i.e., alarms or events, and trends, imported data must include the configuration, resource, and variable names to which it belongs; default values will appear for all other values that remain empty. You need to define variables before proceeding to defining alarms, events, and trends.

Note: The XLS file format is only available when Microsoft Excel is installed on your computer.

An example of an Excel file holding variables data is:

| A | B | C | D | E | F | G | H | I | J | K | | |
|----|---------|-----------|------------|-----------|-------|------------|-----------|-----------|-------|-----------|--------|-----|
| 1 | Config | Resource | Name | Data Type | Alias | StringSize | InitValue | Dimension | Group | Attribute | Scope | Dir |
| 2 | Config1 | Resource1 | VarDInt | DINT | vdinx | 0 | | [1..5] | | Free | Global | Int |
| 3 | Config1 | Resource1 | VarDInt[4] | | | | | | | | | |
| 4 | Config1 | Resource1 | VarDInt[1] | | | | | | | | | |
| 5 | Config1 | Resource1 | VarDInt[2] | | | | | | | | | |
| 6 | Config1 | Resource1 | VarReal | REALI | vrea | 0 | 12.34 | | | Free | Global | Int |
| 7 | Config1 | Resource1 | fldint | FLDIODINT | | 0 | | [1..3] | | Read | Global | Inp |
| 8 | Config1 | Resource1 | fldint2 | FLDIODINT | | 0 | | [1..4] | | Read | Global | Inp |
| 9 | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | |

An example of a CSV file holding variables data:

The first line holding the title of each column (headings) in the same order as they are defined for each variable:

```
Config,Resource,Name,Data Type,Alias,StringSize,InitValue,Dimension,  
Group,Attribute,Scope,Direction,Retain,Address,Comment,  
IODriverIDNum,IOFieldEquipIDNum,IODeviceIDNum,IOChannel,DriverWiring  
AlarmEnable,AlarmTransFilter,AlarmDeadBand,AlarmHHEnable,  
AlarmHHValue,AlarmHEnable,AlarmHValue,AlarmLEnable,AlarmLValue,  
AlarmLLEnable,AlarmLLValue,AlarmHysteresis,AlarmROCEnable,  
AlarmROCValue,AlarmROCPeriod,AlarmHHDevEnable,AlarmHHDevValue,  
AlarmHDevEnable,AlarmHDevValue,AlarmTransient0-1,EventEnable,  
EventAutoDetection,EventDeadBand,EventTransient0-1,  
EventTransient1-0,TrendEnable,TrendCABSize,TrendFileSize,  
TrendHistorical
```

The next line holds the data for the first variable, in the same order as the columns in the first line. Empty spaces between commas indicate that no information is present for the specific field. The data for each variable starts at the configuration level:

```
Config1,Resource1,VarDint,DINT,vdinx,0,,[1..5],,Free,Global,Internal,  
No,,dint  
var,,,,,No,No,0,0,No,0,No,0,No,0,No,0,0,0,No,0,00:00:01:000,No,0,  
No,0,No,No,0,No,No,0,100,10000,Yes
```

The data for the second variable:

```
Config1,Resource1,VarDint [4],,,,,,,,,,,No,No,,,No,,No,,No,,  
,No,,00:00:01:000,No,,No,,No,No,,No,No,yes,100,10000,Yes
```

The data for the third variable:

```
Config1,Resource1,VarDint [1],,,,,,,,,,,No,Yes,0,0,Yes,90,Yes,80  
,Yes,70,Yes,60,0,No,1,00:00:01:000,No,2,No,1,Yes,No,Yes,0,Yes,No,yes,  
100,10000,Yes
```

To import variables data

1. From the Tools menu, choose **Import Variables**.

The Import Type window appears.

- 2.** Do one of the following steps:
 - To add the imported data to the Workbench project's database, click **Append**.
 - To replace the contents of the Workbench project's database with the imported data, click **Replace**.

Warning: When replacing, the Workbench deletes all existing variables for the given resource before importing the data. When appending, the Workbench adds all new variables and inquires about replacing existing variables for the given resource.

The Import window appears.

- 3.** Click **Set File** to locate the file to import, then click **Open**.

The contents of the selected file are displayed in the browser.

To export variables data

- 1.** From the Tools menu, choose **Export Variables**.

The Export window appears.

- 2.** Do one of the following:

- If you are using a template, load the template.
- In the browser, check the data fields to export. To create a template using the selected fields, in the Export Templates section, click **Save**.

- 3.** Click **Set File** to locate the file (.xls or .cvs format) to which to export the data, then click **Open**.

- 4.** Select the resource from which to export the fields. You can also select all resources.

- 5.** Click **Export**.

The defined field data is stored in the specified file.

POUs (Program Organization Units)

A POU (Program Organization Unit) is a set of instructions written in one of the following languages: SFC, FC, IL, ST, FBD, and LD. POUs can be programs, functions, or function blocks.

You can perform many tasks when managing POUs:

- Creating POUs
- Manipulating POUs
- Creating FC Sub-programs
- Creating SFC Child POUs
- Changing Hierarchy Level
- Controlling Access to POUs

Programs

Programs constitute the target Cycle. Programs are also known as POUs. POUs defined as Programs are executed on the Target system respecting the order shown in the Program section. You need to respect the hierarchy of programs within resources.

Available graphical programming languages are Sequential Function Chart, Flow Chart, Functional Block Diagram, and Ladder Diagram. Available literal programming languages are Structured Text and Instruction List. The language of each program is shown as an icon beside the program name:



Sequential Function Chart (SFC)

SFC Editor



Flow Chart (FC)

FC Editor

| | | |
|--|------------------------------|-----------------------|
| | Structured Text (ST) | Multi-language Editor |
| | Ladder Diagram (LD) | Multi-language Editor |
| | Function Block Diagram (FBD) | Multi-language Editor |
| | Instruction List (IL) | Multi-language Editor |

Within a resource there are certain restrictions on the relative positions of programs within the hierarchy:

- All SFC and FC programs must be adjacent within the hierarchy.
- SFC Child or FC Sub-programs must use the same language as their parent.

When using SFC programs in a resource, you may need to change the SFC dynamic behavior factors defined for the resource. For details on the SFC dynamic behavior factors, see page 60.

You can move or copy programs written in ST, LD, and FBD to the Functions section and programs written in SFC, ST, LD, and FBD to the Function Blocks section. You can also move or copy functions and function blocks to the Programs section. When moving or copying a program to the Function or Function Blocks sections, all local variables defined in the program are converted to function or function block parameters respectively.

Note: To call a POU written with a different language from SFC or FC program, call a function or function block (written in ST, LD, FBD or IL).

Functions

Any program can call a Function. Functions are also known as POUs. Functions can only be programmed in ST, LD, or FBD. In all cases, the return parameter of a function must be assigned. You can only declare local variables in functions. However, these local variables cannot be function block instances. Also, you cannot retain the values of variables declared in functions.

Each time a function is executed, its local variables are reset to their initial values (zero when none is provided in the dictionary). When a large structure or array is declared as local variable for a function, the compiler generates code to reset the initial values of each simple variable contained in the structure or array.

The order in which functions appear within their section is not important; functions are called from a POU.

You can move or copy functions to the Function Blocks and Programs sections. You can also move or copy function blocks and programs written with languages supported by functions to the Functions section. When moving or copying a program to the Functions section, all local variables defined in the program are converted to function parameters.

Example

if F1 is programmed as:

```
if (in1) then  
F1 := 10;  
end_if;
```

in the case in1 is FALSE, F1 will not be assigned, and it can take any value.

in the calling program:

```
MyVar := F1 (TRUE) ; leads to MyVar := 10; this is OK  
MyVar2 := F1 (FALSE) ; you can not predict what will be the value of MyVar2
```

Function Blocks

Any program or function block can call a function block. A function cannot call a function block. Function blocks are also known as POUs. Function blocks are written in SFC, ST, LD, or FBD. SFC function blocks can have SFC child function blocks. The order in which function blocks appear within their section is not important; function blocks are called from a POU.

When using SFC function blocks and SFC child blocks, you need to specify the maximum number of tokens for each one in their individual properties.

You can move or copy all function blocks to the Programs section and all but the SFC function block to the Functions section. You can also move or copy functions and programs, written with languages supported by function blocks, to the Functions section. When moving or copying a program to the Function Blocks section, all local variables defined in the program are converted to function block parameters.

Creating POUs

You create, i.e., add, POUs (programs, functions, and function blocks) in resources while in the link architecture view. You add POUs using the main menu or a contextual menu accessed by right-clicking the respective component (Program, Function, or Function Block) within a resource. After having created a POU, you can drag and drop it to a new position in its section, to another section, or to another resource. POUs belonging to a same section must have different names. POU names must begin with a letter.

For SFC programs and SFC child programs, you may need to change the SFC dynamic behavior factors for the resource. For details on the SFC dynamic behavior factors, see page 60. For each SFC function block and SFC child block, you may need to adjust the maximum number of tokens.

To create a POU

1. In the resource window, select the POU component to create.

2. From the Insert menu, choose **Add Program**, then the desired language.

The new component appears at the end of its respective section with its name ready to edit.
3. Type a name for the component.
4. For SFC POU, do one of the following:
 - For an SFC program or SFC child program, make sure the dynamic behavior factors defined for the resource are sufficient by selecting the resource, then from the Edit menu, choosing Properties, then the Settings tab, then clicking **Advanced Settings**.
 - For an SFC function block or SFC child function block, specify the maximum number of tokens by selecting the block, then from the Edit menu, choosing Properties, then the Settings tab.

Manipulating POU

You can move, cut, copy, paste, and delete POU, with certain exceptions, within their sections, to other sections, and from one resource to another. You can only move or copy POU between sections supporting the same language. For instance, you cannot move or copy an SFC program or function block to the Functions section.

You can move programs to change their order of execution or to change them to functions or function blocks. You can move functions to change them to programs or function blocks and move function blocks to change them to programs or functions. Changing a function or function block's order within its section has no effect on its execution since it is called.

Note: Before manipulating POU, you should save the changes made to your project.

To move a POU

1. Select the POU in the resource window.
2. Drag and drop the POU to its new location.

Note: You can only move POU s between sections supporting the same language. You cannot move a program (Child SFC or FC) to change its hierarchy level; you can only move it to change its position as a child within the same level. To change the hierarchical level of an SFC or FC program to become a child, see “Changing Hierarchy Level” on page 103.

To cut, copy, or paste a POU

The cut, copy, and paste commands use the clipboard as temporary storage. Once copied (or cut), a POU can be pasted more than once. You can only paste POU s between sections supporting the same language. SFC programs are pasted at the same hierarchical level as the selected program. When copying and pasting POU s having access control, password definitions are retained.

1. In the resource window, select the POU.
2. From the Edit menu, choose **Cut <Ctrl+X>** or **Copy <Ctrl+C>** (or use the contextual menu).
3. Select the new location, i.e. the Program, Function, or Function Block section within the same or different resource.
4. From the Edit menu, choose **Paste <Ctrl+V>** (or use the contextual menu).

To delete POU s

1. Select the POU.
2. From the Edit menu, choose **Delete **.

To copy POU s from a project to another

1. In the destination project, create a program having the same name and language as the program in the original project.
2. From the original project directory of the program's resource, copy the *POU_name.stf* file, then paste the file in the destination project's resource directory.
3. In the destination project, redeclare local and global variables needed for the POU.

Creating FC Sub-programs



Flow Chart (FC)



Flow Chart (FC) Sub-program

To create an FC sub-program

You can create FC sub-programs using the main menu options or a contextual menu accessed by right-clicking the FC program component within a resource.

1. In the resource window hierarchy, select an existing FC program .
2. From the Insert menu, choose **Add FC Sub-Program**.

Creating SFC Child POUs



Sequential Function Chart (SFC)



Child Sequential Function Chart (SFC)

To create a child SFC POU

You can create child SFC POUs using the main menu options or a contextual menu accessed by right-clicking an SFC POU within a resource.

1. Select the existing SFC POU  in the resource window hierarchy.
2. From the Insert menu, choose **Add Child SFC**.

Changing Hierarchy Level

You can promote or demote child SFC (FC) POUs, depending on their relative position in the hierarchy.

To change the level of an SFC (FC) POU

1. Select the SFC (FC) POU.
2. Do one of the following:
 - From the Edit menu, choose **Move to lower Level** or **Move to upper Level**.
 - From the Main toolbar, click  to move the program to a lower level or  to move it to the upper level.

Example

Consider the following two SFC POUs:



Sequential Function Chart (SFC)



Sequential Function Chart (SFC)

Selecting the second SFC POU and moving it down a level would produce:



Sequential Function Chart (SFC)



Child Sequential Function Chart (SFC)

Selecting the Child SFC POU and moving it up a level would result in:



Sequential Function Chart (SFC)



Sequential Function Chart (SFC)

Controlling Access to POUs

You can control access to user-defined POUs using a password. When you set a project with the read-only access control, the resources and POUs making up the project are also set to the read-only mode except for those having individual access control. For instance, a POU having its own password remains locked and cannot be viewed without entering its password. When moving or copying a POU using its resources password, the POU retains this password.

The security state of a POU is indicated by its icon color in the resource:

| POU Icon Color | Security State |
|-------------------|--|
| Yellow | Yellow. The POU has no access control. All users have read and write access in the POU. In the dictionary view, local variables and parameters are visible and editable. |
| Red | Red. The POU is locked. Users not having the POU password cannot access the POU; these users do not have read or write capabilities. In the dictionary view, local variables and parameters are visible but not editable. |
| Blue | Blue. The POU is in read-only mode. Users not having the resource password can view the POU; these users do not have write capabilities. The read-only mode for the POU is inherited from the resource to which it belongs. In the dictionary view, local variables and parameters are visible but not editable. |
| Green | Green. The POU is unlocked. User can access the POU; this user has read and write capabilities. In the dictionary view, local variables and parameters are visible and editable. |

Note: While in debug mode or performing builds, unlocked POUs as well as POUs having no access control switch to read-only mode. Locked POUs remain locked.

You can build POU s of all security states.

When copying, pasting, importing, and exporting POU s having access control, password definitions are retained.

To set access control for a POU

You set access control for a POU by setting a password.

1. In the resource window, select the POU for which to set access control.
2. From the Edit menu, choose **Properties**.

The Program Properties window is displayed showing the Security tab.



3. Specify a password:
 - To use a unique password, in the New field, enter a password then reenter it in the Confirm New field.
 - To use the same password as set for the resource to which the POU belongs, check **Use Resource Password**.

To unlock a POU

When entering a password while in debug mode or performing builds, the POU is only unlocked after stopping the debug mode or when the build is completed.

1. In the resource window, right-click the POU, then from the contextual menu, choose **Enter Password**.
2. In the Security dialog box, enter the password for the POU.

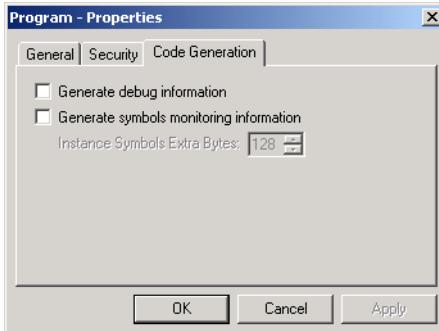


The POU is unlocked.

Generating Debug and Monitoring Information

You can choose to generate debug and symbols monitoring information for POU s. Debug information is available for ST, IL, and LD POU s (programs, functions, and function blocks) for use when debugging using the step-by-step mode. Symbols monitoring information is available for ST, IL, FBD and LD programs and function blocks for use when debugging or simulating to graphically display the output values of functions and function blocks. For details on the graphical display of output values, see page 295.

You set the generation of debug and symbols monitoring information for a POU on the Code Generation tab of the Program Properties window:



When generating symbols monitoring information for function blocks, you also need to specify the instance symbols extra bytes. This indicates the size of memory reserved for each function block instance for adding symbols monitoring information during online changes. Note that a string-type output takes up 260 bytes.

You can change the default value for the *Generate symbols monitoring information* option as well as the *Instance Symbols Extra Bytes* size. Their values are specified in the FunctionMonitoringSupportDefault and MonitoringSpaceDefault parameters of the Settings section of the Diamond.ini file, located in the Bin folder. For details on the location of the bin folder, see page 29.

The symbols information generated for graphically monitoring output values requires a significant amount of memory space. Therefore, when compiling, an error message stating that the memory limit has been reached may be displayed in the output window. In such a case, to enable compiling, you need to either disable monitoring for the POU, remove elements from the POU, or clean the project.

To generate debug information for a POU

You can also generate debug information for POUs at the resource level.

1. In the resource window, select the POU for which to generate debug information.
2. From the Edit menu, choose **Properties**.

The Program Properties window is displayed showing the Code Generation tab.

3. Check **Generate debug information**.

To generate monitoring information for a POU

1. In the resource window, select the POU for which to generate monitoring information.
2. From the Edit menu, choose **Properties**.

The Program Properties window is displayed showing the Code Generation tab.

3. Check **Generate symbols monitoring information**.
4. For function blocks, specify the size of Instance symbols extra bytes.

Editing a POU Description

You can add a free-format text description for a POU.

To edit the POU Description

1. Select a POU.
2. From the Tools menu, choose **Edit Description**.
3. Edit the description as required.

Hardware Architecture View



The hardware architecture view graphically displays the configurations of a Project and the network links between them. From the hardware architecture view, you manage many aspects of a project:

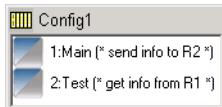
- creating configurations
- attaching targets to configurations
- inserting resources into configurations
- moving resources between configurations
- creating networks
- connecting configurations and networks
- defining configuration connection properties
- defining resource network properties
- setting up I/O wiring

To switch to the hardware architecture view

- From the Window menu, choose *project_name-Hardware Architecture*.

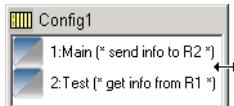
Configurations

A configuration represents a hardware definition:



When creating a new project, a default configuration is automatically created. Subsequent configurations must be manually inserted.

You can resize configuration windows by placing the cursor over an edge or corner until it shows double arrows and dragging:



Creating Configurations

You can create configurations using the main menu or a contextual menu, accessed by right-clicking within the workspace. Following the creation of a configuration, the Configuration Properties dialog box automatically appears where you attach it to a target. Choosing a target leads to the accessibility of network, I/O devices, and RAS device functions and function blocks supported by this target.

To create a configuration

1. Switch to hardware architecture View .
2. From the Insert menu, choose **Configuration**.

An empty configuration is created using a default name, then the Configuration Properties dialog box appears:



3. On the Hardware Tab, choose a Target to attach to the configuration:



Deleting Configurations

You can delete configurations using the main menu or a contextual menu, accessed by right-clicking a configuration's title bar. You cannot delete the last configuration of a project; projects must have at least one configuration.

To delete a configuration

1. Select the hardware architecture view
2. Select a configuration.

Note: To deselect resources in the configuration window, click an empty space in the configuration window.

3. From the Edit menu, choose **Delete **.

Moving Configurations

When you move configurations, the hardware architecture view is re-drawn to tidy-up the display. Fixed-sized gaps are placed between network and configurations.

To move a configuration

1. Select the configuration.

The selected configuration's title bar is highlighted.

2. Drag and drop the configuration as desired.

Inserting Resources

You can choose to insert, i.e., create, resources directly in a configuration while in the hardware architecture view of your project. You can also create resources in the link architecture view. However, in the link architecture, new resources are automatically assigned to the first configuration.

To insert a resource in a configuration

You can insert resources using the main menu or a contextual menu, accessed by right-clicking the empty space in the configuration's window.

1. Select a configuration.
2. From the Insert menu, choose **Resource**.

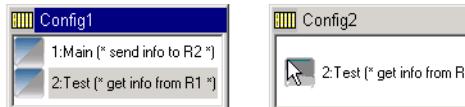
Moving Resources Between Configurations

When moving resources from one configuration to another, you need to make sure several aspects of the destination configuration are compatible with those of the source configuration:

- Network Information, when both configurations are connected to the same networks, resource information remains intact. Otherwise, you will need to change the binding network information for the moved resource.
- C function or C function block calls, when the list of available C functions or function blocks is different for both configurations, when proceeding to build the resource, some errors may occur when the functions called do not point to the functions declared in the target.
- I/O Wiring, when the I/O device list is different for both configurations, the I/O wiring of the moved resource is deleted.

To move a resource from one configuration to another

1. Click and hold the mouse button on the required resource.
2. Drag and drop the resource to the new configuration.



Configuration Properties

Configuration properties are defined from the hardware architecture view.

To access the Configuration Properties window

1. From the Window menu, choose *project_name-Hardware Architecture*.

The hardware architecture view appears displaying all configurations defined for a project.

2. Select a configuration.
3. From the Edit menu, choose **Properties**.

The Configuration Properties window appears.

Configuration Link to ROCLINK Configuration File

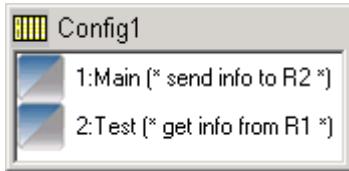
The configuration general properties enable you to assign a meaningful name to a configuration and link the configuration to a ROCLINK 800 configuration file. You need to link a configuration to a ROCLINK 800 configuration file before defining TLP variables in the dictionary. When the configuration is linked to a configuration file, the name and location of the configuration file is automatically entered in the Comment field. Comments appear within (* *) next to the name of the configuration in its title bar.

Note: The ROCLINK 800 Configuration Software must be installed on the same computer running the DS800 Workbench.

You can choose to replace the configuration representation in the hardware architecture view with a custom bitmap by checking the Use bitmap option, then browsing to locate the bitmap.

Standard Configuration Representation

Sample Bitmap Representation



When using a custom bitmap for configurations, a copy of the bitmap is automatically placed in the configuration folder and renamed to use the configuration's name.

You specify the configuration general properties in the General tab of the Configuration Properties window:



To link a configuration to a ROCLINK 800 configuration file

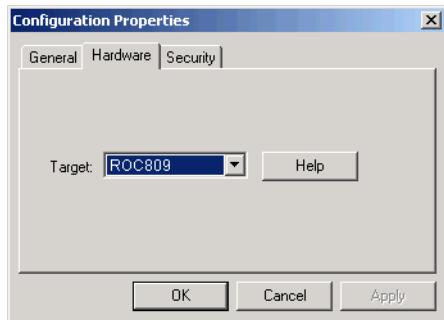
1. Click .
2. In the *Select ROCLINK 800 Configuration File* browser, locate the configuration file to link.
3. Click **Open**.

DS800 extracts the IP address of the RAS device target from the ROCLINK 800 configuration file and automatically creates the network connection. The configuration file name and location is indicated in the Comment field.

Configuration Target Definitions

The configuration target definition property enables you to attach a target to the configuration. Changing targets for a configuration affects all resources attached to the configuration.

You specify the configuration target definition property in the Hardware tab of the Configuration Properties window:



The selection of the target determines:

- the network on which you can connect the configuration and that can be used in Binding definition.
- the I/O devices that you will be able to use in the I/O Wiring Tool
- the list of C functions and function blocks that you will be able to call in your programs.

Warning: Changing the target of a configuration may lead to the destruction of the I/O wiring of all resources within the configuration and connections to networks. You should assign targets to configurations as a first step in your project development.

When the advanced options are installed, you can choose whether to download the advanced options features such as alarms and events definitions, trends definitions, events server configuration, and trends server configuration.

You can also choose to add a help file using the Help button.

Target Access Control

For configuration security, you can control access to a target by setting a password. This password is embedded on the target and can only be set or changed while running in real-time or debug mode. The configuration access control prevents the connection of all IXL clients not having the target's password.

At run time, the security state of a configuration is indicated by its title bar icon:

| Configuration Icon | Security State |
|--------------------|---|
| | The configuration has no access control. All IXL clients can access the target. |
| | The configuration is not accessible; the target does not recognize the password. IXL clients not having the target password cannot access the target. |
| | The configuration is accessible; the target recognizes the password. IXL clients having the target password can access the target. |

To specify access control for a configuration

You set access control for a configuration in the configuration's Security properties.



- In the Password field, enter the password for the configuration, then reenter the password in the Confirm Password field.

Note: You can only change a password while in real-time or debug mode. Otherwise, the password embedded on the target remains unchanged.

Configuration Description

A free-format text description of the configuration.

To edit the configuration description

- 1.** Right-click on the configuration title bar.

The contextual menu appears.

- 2.** Choose **Edit Description**.

- 3.** Edit the description as required.

Networks

Networks provide the means for communication between configurations. Configurations need to communicate when bindings have been defined within them. Configurations are connected to the network. The target attached to the configuration must support the network the configuration is connected to. You define network properties when you create them.

When you create DS800 projects, networks are automatically inserted.

Networks are represented in the hardware architecture view as a horizontal 'bar':



Notes:

- A project can contain an unlimited number of networks.
- If the network is not implemented in the target, it is the responsibility of the integrator to develop and implement a driver for that particular network. Creating a Network.
- The default network is ETCP.
- ISaRSI is only used to connect the Workbench to the RAS device target. ISaRSI does not support projects with multiple configurations.

Creating Networks

You define network properties at the time of creation. You need to specify the protocol (also called Network Driver) to use for communications between configurations when bindings are defined. The parameters defining the network appear in the grid. Some parameters may be read-only (greyed). Not all networks require parameters at this level, e.g., ETCP for Ethernet.

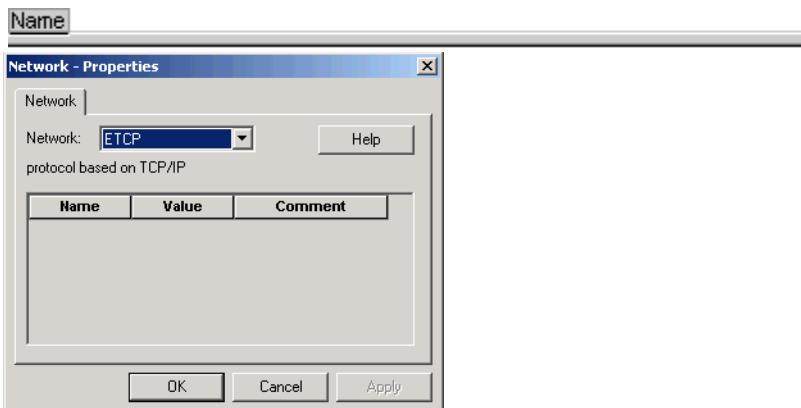
When you create DS800 projects, networks are automatically inserted.

You can choose to integrate help using the Help button.

To create a network

1. Switch to the hardware architecture view .
2. From the Insert menu, choose **Network**.

A new Network is created and the Network Properties dialog box from which you select a protocol. The available protocols are ETCP and ISaRSI.



Note: ISaRSI is only used to connect the Workbench to the RAS device device. ISaRSI does not support projects with multiple configurations.

Moving Networks

The Network can be moved vertically within the workspace. This facility is simply a method of providing a preferred view for the user, usually the default view is preferred.

To move a network

1. Select the network.

The selected network is highlighted.

2. Drag and drop the network as required.

Note: The hardware architecture view is re-drawn to 'tidy-up' the display. Fixed-sized gaps are placed between network and configurations.

Connections

Connections between networks and configurations enable communications to flow. DS800 automatically creates connections when creating projects. When you link a configuration to a ROCLINK 800 configuration file, the IP address is automatically assigned. A configuration can be linked to many networks. Similarly, a network can be linked to many configurations.



Creating Connections

When creating a connection, make sure to not select the configuration or network. Click elsewhere in the workspace to deselect these items. In the connection's properties, you need to specify the IP address of the target, for example:

192.168.2.36

Note: DS800 automatically creates connections when creating projects. When you link a configuration to a ROCLINK 800 configuration file, the IP address is automatically assigned.

The list of available parameters depends on the network to which the configuration is connected. This list may be empty. Some parameters may be read-only (displayed greyed). For the ETCP (Ethernet) network driver, only the IP address of the configuration is required.

Note: A connection may fail if the network protocol is not supported by the configuration's target.

To connect a configuration and network

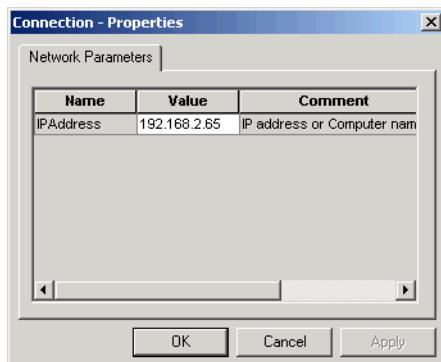
1. Click and hold the mouse button on the title bar of the configuration to connect.



The mouse becomes a network connection cursor:

2. Drag and drop the mouse cursor to the required network.

The connection is created and the Connection Properties dialog box is displayed.



3. In the Value field, enter the IP address, then click **OK**.

Deleting Connections

You can remove existing connections between configurations and networks.

To delete a connection between a configuration and network

1. Select the connection.
2. Do one of the following:
 - From the Edit menu, choose **Delete Connection**.
 - Press **Delete**.

Dictionary View



The Dictionary is an editing tool using tree views and grids for the declaration of the variables, functions, and function block parameters, user types and defined words of the project.

The various components are sorted in a tree-like hierarchy, e.g., by resource or by Type. The Tree name is displayed on the window title bar. The four dictionary tree views are:



Variables Tree



Parameters Tree



Types Tree



Defined Words Tree

Note: You need to declare variables before proceeding with the I/O Wiring process.

To switch to the Dictionary view

- ▶ Do one of the following steps:
 - From the Project menu, choose either **Types**, **Variables**, **Function/Function Block Parameters**, or **Defined Words**.

Note: The choices available differ depending on whether you are in the hardware architecture or link architecture views.

- On the Window Buttons toolbar, click
- Open a variable group.

To switch to the Dictionary view from a language editor

Opening the Dictionary from an Editor opens the Variable Tree and grid for the POU being edited.

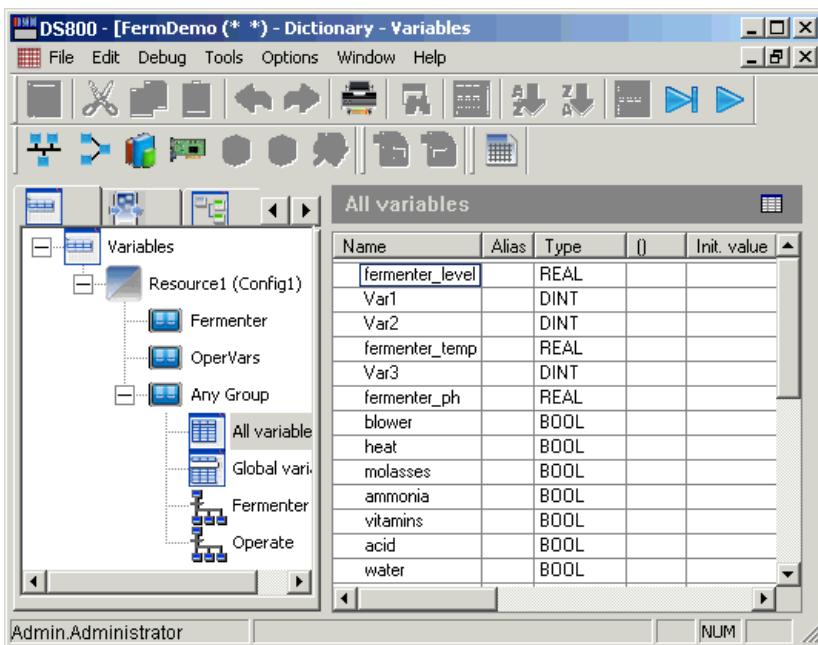
- ▶ Do one of the following steps:
 - From the File menu, choose **Dictionary**.
 - On the Standard Buttons toolbar, click .

Appearance

The Dictionary view is displayed maximized in the workspace. The menus and toolbar now reflect Dictionary options only.

The left of the dictionary workspace is a tree-like hierarchical structure of either variables, parameters, types, or defined words. The right side of the workspace displays a grid-like table.

Titlebar



Menu Bar

Toolbars

Workspace

Status Bar

Variables Tree

The branches provide different ways to access the variables of each resource:



Top Level



Resources



Variable Group

Grid displays only variables in that group.



Any Group



All Variables

Grid contains all variables in the resource



Global Variables

Grid contains all global variables



Programs

Grid contains global variables and variables local to the program



Functions

Grid contains global variable and variables local to the function

Note: When the cursor is positioned over an item, the full name and comments are displayed in the ToolTip.

Parameters Tree

The branches in each resource show all functions and function blocks, in order to define their parameters in the corresponding grid.

 Top Level

 Resources

 Functions

 Function

Grid displays the parameters of the function

 Function Blocks

 Function Block

Grid displays the parameters of the function block

Note: When the cursor is positioned over an item, the full name and comments are displayed in the ToolTip

Types Tree

The various tree levels are represented using the following icons:



Top Level



Arrays



Structures

Level



Structures

Individual Structures

When the cursor is positioned over an item, the full name and comments are displayed in the ToolTip.

Types have a Common Scope, they can be used as a type or any variable of any resource.

Creating Structures

To create a structure

1. Right-click on the 'Structures' top of tree.
2. From the Edit menu, choose **Add Structure**.

A structure has been created at the end of the tree. Its name is displayed and ready for editing.

Note: In addition to a name, you can include a comment, e.g., StructName (* comment *).

Renaming Structures

You can rename a structure using the main menu or a contextual menu, accessed by right-clicking a structure.

To rename a structure

1. Right-click on the structure to rename.
2. From the Edit menu, choose **Rename Structure**.
3. Enter a name and comment in the dialog box.

Deleting Structures

You can delete structures using the main menu or a contextual menu, accessed by right-clicking the structure.

To delete a structure

1. Select the structure in the tree.
2. From the Edit menu, choose **Delete Structure**.

Defined Words Tree

There is no Tree for defined words, these are entered in the grid. Defined words have a Common Scope, they can be used in any POU of any resource. For information on the Defined Words grid, see page 145.

Working with the Grids

Grids display characteristics and values for components corresponding to the selected Tree View. You create, manipulate, and make changes for variables, functions, and function block parameters, user types and defined words directly in the grids. The grid is a table formatted database. You can use one of two editing modes while working in the grids:

- Grid, where you can access individual cells. In this mode, a grid outlines individual cells:

| Name | Alias | Type | 0 | Init. value |
|-------|-------|------|---|-------------|
| + din | | BOOL | | |

- Line, where you can access complete rows, i.e., lines. The information contained in the line appears in a dialog box where you can change it. In this mode, no grid appears:

| Name | Alias | Type | 0 | Init. value |
|-------|-------|------|---|-------------|
| + din | | BOOL | | |

Keyboard shortcuts enable navigating throughout the grid. The behavior of the shortcuts differs depending on the editing mode of the grid.

| Shortcut | Grid Mode | Line Mode |
|-----------|---|---|
| Tab | Moves from one grid cell to the next from left to right. When editing the contents of a cell, the edition mode is retained in the next cells. | Moves from one line to the next from top to bottom |
| Shift+Tab | Moves from one grid cell to the next from right to left. When editing the contents of a cell, the edition mode is retained in the next cells. | Moves from one line to the next from bottom to top |
| End | Moves to the bottom of the variables list | Moves to the line at the bottom of the variables list |
| Home | Moves to the top of the variables list | Moves to the line at the top of the variables list |

Note: When defining TLP variables, you need to set the variables grid to the line editing mode.

To switch editing modes

- Directly above the grid, click .

Resizing Columns

You can resize columns or rows.

To resize a column (or row)

1. Click and hold a cell header divider:

| Name | Alias | Type | 0 | Init. value |
|------|-------|------|---|-------------|
| | | BOOL | 0 | |

2. Drag and drop it as required (drag to the left in the above example to shrink the Name column).

Selecting Rows and Elements

You can select either rows or individual cells in the grid depending on the selected editing mode:

To select rows

- ▶ While in the Line editing mode, click on the row.
- ▶ While in the Grid editing mode, click the left-most edge of the row.

To select items in the grid

While in the Line editing mode, you can select one or more items in the grid.

1. To select a single item, click the item.
2. To select more than one consecutive item, click the first item, then while holding down the <SHIFT> key, click the last one.

All the elements between the first and last are selected.

3. To select many individual items, click each one while holding down the <Ctrl> key.

Editing the Contents of the Grid

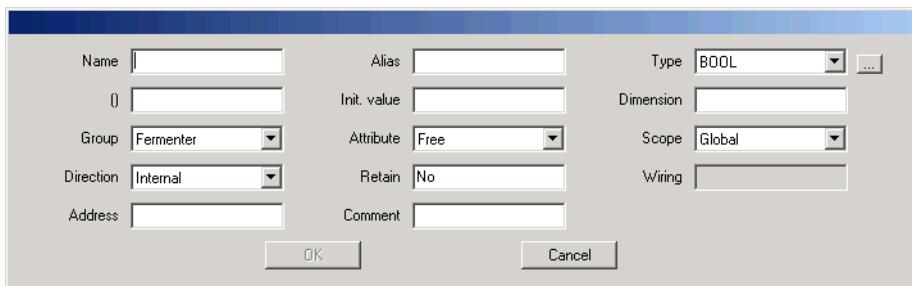
You can edit the contents of individual cells or complete rows depending on the selected editing mode. **To edit the contents of a cell**

- While in Grid mode, double-click an element within the row.

To edit the contents of a row

1. While in Line mode, double-click a row.

The variable dialog is displayed.



2. Make the necessary changes to the variable fields. For the Type field, you can also access the Select Data Types browser by clicking

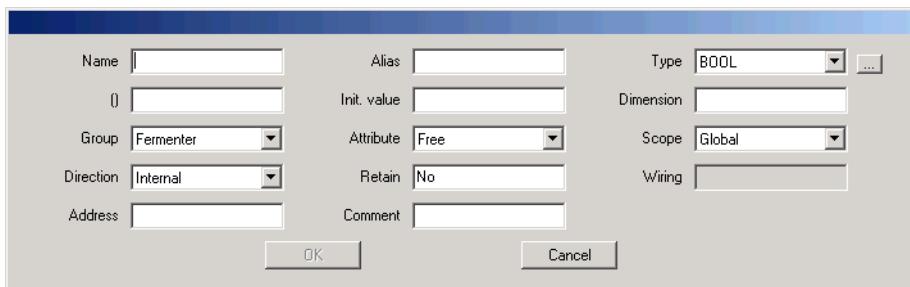
Adding or Inserting Rows

You can edit the contents of existing rows, add rows at the end of the grid, or insert rows at a specific location in the grid. You can perform these tasks from the main menu or a contextual menu, accessed by right-clicking in the grid.

To add a row

- ▶ From the Edit menu, choose **Add Row**.

The grid dialog box appears:



Some fields have pull-down menus, showing the options available for that field. The Type field also enables access to the Select Data Types browser.

Note: The group name is automatically asserted when a variable group is selected in the Variable Tree.

To insert a row in the grid

1. Select a row in the grid.
2. From the Edit menu, choose **Insert Row**.

When the Line editing mode is selected, a row is inserted in the grid. When the Grid editing mode is selected, grid dialog box appears.

Moving Rows

You can change the position of a variable or a parameter in the grid, by dragging the line to a new position.

Note: You cannot undo row-moving operations.

Expanding or Collapsing Grid Components

Variables with user types (Arrays and Structures) are initially displayed 'collapsed', i.e. only the variable definition row is displayed, with a + sign in the row header cell. Clicking on the row header cell expands or collapses that variable.

For example, the variable In1 of type arr1, where arr1 is defined in the Dictionary as an Array of [1..3] Booleans, is initially displayed as:

| | | | | | | | | | | |
|---|-----|--|--|------|------|--|--|------|----|--|
| + | in1 | | | None | Arr1 | | | Free | Gk | |
|---|-----|--|--|------|------|--|--|------|----|--|

When expanded, the complete definition of in1 is shown:

| | | | | | | | | | | |
|---|--------|--|--|------|------|--|--|------|----|--|
| - | in1 | | | None | Arr1 | | | Free | Gk | |
| | in1[1] | | | | BOOL | | | | | |
| | in1[2] | | | | BOOL | | | | | |
| | in1[3] | | | | BOOL | | | | | |

Cutting, Copying, and Deleting Elements

You can cut, copy, or delete either rows or individual cells in the grid depending on the selected editing mode. The Cut command removes selected elements and places them on the clipboard. The Copy command places the selected item on the clipboard. The clipboard holds only one item at a time.

To cut elements

1. Select an element.
2. From the Edit menu, choose **Cut <Ctrl+X>**.

To copy elements

1. Select an element.
2. From the Edit menu, choose **Copy <Ctrl+C>**.

Deleting elements

- Select an element then press **Delete**.

Finding and Replacing Elements

You can search for and replace elements in the grid. , however, you can only replace the following elements in the respective grids:

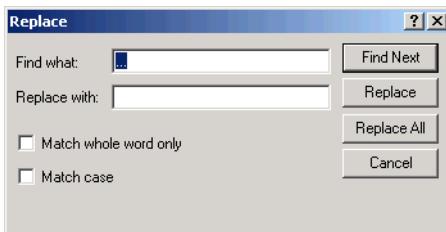
| Variables | Parameters | Types | Defined Words |
|------------|------------|---------|---------------|
| Name | Name | Name | Word |
| Alias | Short Name | () | Equivalent |
| () | Comment | Comment | Comment |
| Init.Value | () | | |
| Dimension | Dimension | | |
| Address | | | |
| Comment | | | |

To differentiate between upper and lower case characters during a search, check **Match Case**.

To search or replace an element (a character, word or phrase):

1. From the Edit menu, choose 'Find / Replace' <Ctrl + F>.

The Find / Replace dialog is displayed.



2. To search for an element, in the Find what field, enter the element to search for, then click **Find Next**.
3. To replace an element, in the Replace with field, enter the element, then click **Replace**. To replace all occurrences of the element, click **Replace All**.

Pasting Elements

You can paste the contents of the clipboard above the currently selected row(s), if one or more rows have been copied or cut.

To paste

1. Click on the required insertion point.
2. From the Edit menu, choose **Paste** <Ctrl+V> or on the Standard toolbar, click ..

OR

1. Right-click the required insertion point.
2. From the contextual menu, choose **Paste**.

Sorting the Grid

You can sort the contents of individual columns of the grid from the main menu, from the toolbar, or by clicking the individual column headers.

To sort the grid

1. Do one of the following:
 - From the Tools menu, choose **Sort Ascending** or **Sort Descending**.
 - On the toolbar, click  to sort in an ascending manner or  to sort in a descending manner.
2. In the Sorting dialog box, choose the criteria (column) to use for sorting, then click **OK**.

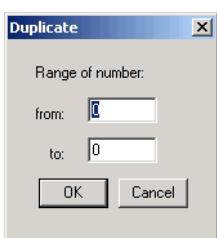
Duplicating Rows

You can duplicate rows, automatically generating sequentially numbered 'name' copies.

To duplicate a row

1. Select the row to Duplicate.
2. Do one of the following:
 - From the Tools menu, choose **Duplicate <Ctrl+U>**.
 - On the Standard toolbar, click .

The Duplicate dialog box is displayed:



3. Enter the From and To numbers to use for the automatic generation of names.
4. Click **OK**.

The newly created rows are inserted below the selected row.

Renumbering Addresses

Renumbering addresses automatically generates contiguous addresses within a selected range of the grid. Renumbering is only available in the variables grid. When renumbering, certain cells such as a function block instances are ignored since they have no address.

The reduced symbol table contains the set of variables with addresses.

To renumber addresses

1. Select the rows to renumber their address
2. From the Tools menu, choose **Renumber Addresses**.



3. In the Renumber Addresses dialog, enter the 'From Address' (hexadecimal value ranging from : 1 to FFFF).
4. Click **OK**.

Example

- If A1 is entered as a Start Address, A1, A2, A3, A4... are generated.
- If AA is entered as a Start Address, AA, AB, AC, AD... are generated.

Printing a Grid

You can choose to print the current grid. This command launches the Document Generator with the standard list of elements to be printed for a grid. For information on the Document Generator, see page 337.

To print the current grid

- From the File menu, choose **Print**.

Variables Grid

The variables grid allows the definition of variables for each resource created in the project.

The columns of the variables grid are:

| Column | Details |
|------------|---|
| Name | Variable name: limited to 128 characters, conforming to the IEC 61131 standard |
| Alias | Any name. Used in LD Editor |
| Group | Group name or "None" |
| Type | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array Types, Structure Types, Function Blocks. See Glossary. |
| () | if Type is STRING this represents the string length (max. 255 characters) |
| Dimension | For example: [1..4,1..7]. See Glossary. |
| Attribute | For example: READ-ONLY. See Glossary. |
| Scope | Global or local to a program or function. see Glossary. |
| Direction | of I/O Wiring; Input, Output or Internal. |
| Init.value | Numeric or Textual. See Glossary. |
| Wiring | Read-only cell, generated by the I/O Wiring tool. Uses syntax of Directly Represented Variable |
| Comment | User comments: Free format |
| Retain | Yes or No. See Glossary and Resource Settings Properties. |
| Address | Hexadecimal value in the range 1 to FFFF. |

Parameters Grid

The Parameters grid defines the interface of the functions and function blocks created in the project resources. The columns for parameters are:

| Column | Details |
|------------|--|
| Name | Parameter name: Limited to a maximum of 128 characters and must conform to the IEC 61131 Standard. |
| Short Name | Short name used in the FBD and LD Editors for display only (max. 4 chars). |
| Type | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array Type, Structure Type, Function Block Type. see Glossary |
| () | If Type is STRING, () is the length (max. 255 chars). |
| Dimension | Example [1..4,1..7] for a two dimensional Array. see Glossary |
| Direction | Input Parameter, Output Parameter or Local |
| Comment | User comments: Free format |

Note: Parameters are sorted within the database; "Input", then "Output", then "Local". Functions have only one output parameter which must be a simple type (i.e., no arrays or structures). Function block instances can only be defined as local parameters of function blocks. To call a function block in a function block (nested function blocks), you may create the instance of the called function block as a local parameter of the calling function block. This enables you to spy the local parameters of the called block .

Types Grid

In the Types grid, you create complex types that will then be available for variable declaration, i.e., new types will appear in the 'Type' selection in all grids. The columns for types are:

- Arrays:

| Column | Details |
|--------------|--|
| Name | Array name: maximum 128 characters, conforming to the IEC 61131 Standard |
| Element Type | Array Element Type: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, User Arrays, Structures |
| () | If Type is STRING, this represents the length (maximum 255 characters) |
| Dimension | Example: [1..10] for a one dimensional Array, [1..4,1..7], for a two dimensional Array. The dimension must be defined as a positive double integer (DINT) value. |
| Comment | User comments: Free format |

- Structures:

| Column | Details |
|--------------|--|
| Name | Element name: maximum 128 characters, conforming to the IEC 61131 Standard |
| Element Type | Element Type: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, User Arrays, Structures |
| () | If type is STRING, this represents the length (maximum 255 characters) |
| Comment | User comments: Free format |

Notes:

- To create a structure with an element with a dimension, first create an array, then create a structure with an element of type <Array name>.
- Type recursive use is not allowed, e.g., one field of 'str1' cannot use the 'str1' type

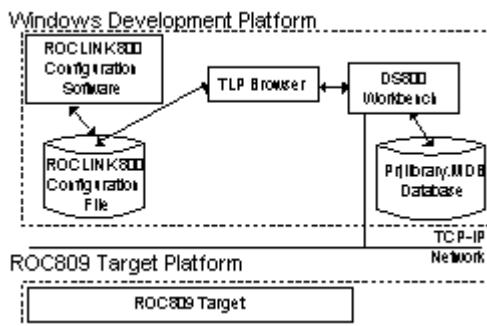
Defined Words Grid

The columns for defined words are:

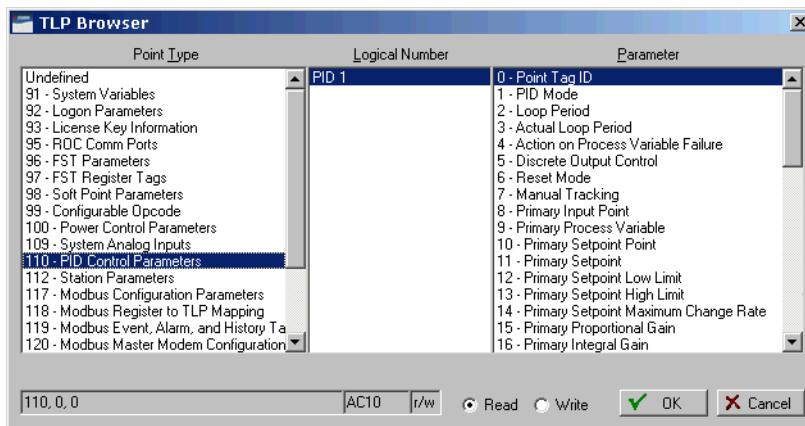
| Column | Details |
|---------------|--|
| Word | Name used in ST source files: first character must be a letter, following characters must be letters, digits or underscore ('_'). |
| Equivalent | String according to ST syntax, that replaces the defined word during compiling. Example: Word = PI, Equivalent = 3.14159 |
| Comment | User comments: Free format |

Defining TLP Variables

The TLP browser enables the selection of TLP numeric point locations from the ROCLINK 800 configuration file for use as TLP variables in the DS800 environment. Before defining TLP variables, you need to link the configuration to a ROCLINK 800 configuration file and set the configuration target definition property to the applicable RAS device target. DS800 automatically performs the wiring from defined TLP variables to TLP devices. Each TLP variable is assigned a TLP device.



In the TLP browser, you select three parameters to define point locations:



- Point Type (T) - The point type opens a list of logical numbers and parameters belonging to that Point Type.

- Logical Number (L) - In the configuration screens, the Logical Number is generally referred to as Point Number or Number.
- Parameter (P) - These are usually called by the same term as the Tag on the configuration screen.

The display field at the bottom left of the TLP browser displays the numeric point location of the TLP point and its data type as well as the point's read/write attribute. You also set the read/write attribute of the TLP variable for the DS800 Workbench.

The **ROCLINK 800** data types are matched to the corresponding DS800 data type:

Correspondence of Data Types

| ROCLINK 800 | DS800 |
|--------------------|--------------------|
| FL | REAL |
| UINT8 | DINT |
| UINT16 | DINT |
| INT16 | DINT |
| UINT32 | DINT |
| ACx | STRING(<i>x</i>) |
| TIME | STRING(22) |
| TLP | tlpAddress |
| HOURMI | DINT |
| BIN | DINT |
| Softpoint | Softpoint |

In the dictionary, the Comment field indicates the TLP numeric point definition and location

DS800 automatically generates the correct wiring for TLP variables entered in the dictionary. When a TLP variable is deleted from the dictionary, its assigned device is also deleted. Also, when a TLP device is deleted, its wired TLP variable is also deleted. When importing TLP variables, the automatic wiring is not executed.

When selecting a Softpoint Parameter point type with the 0 parameter, the TLP variable has the Softpoint structure type. For a Softpoint structure, you may need to disable the write permission for its individual fields; by default, Softpoint structure fields have the write permissions.

To disable the write permission for a structure field, in the I/O wiring view, select the corresponding field, then in the I/O Parameter dialog, set the parameter's field to FALSE.

To define TLP variables in the dictionary

You access the TLP browser from the dictionary's variables grid while in the line editing mode.

1. Select a row in the variables grid, then from the Edit menu, choose **Add Row**.
2. In the grid dialog, click  .
3. In the TLP browser, to define the point location, select the point type, logical number, and parameter, then indicate the read/write permissions for the device assigned to the TLP variable and click **OK**.

The TLP variable's attributes are displayed in the grid dialog.

4. To enter the TLP variable in the variable grid, click **OK**.

When you save changes in the dictionary, DS800 automatically performs the wiring from the TLP variable to a TLP device. The assigned TLP device's directly represented variable naming is displayed in the Wiring field of the variable grid.

To disable the write permission for a Softpoint structure field

1. In the dictionary's Wiring field, identify the TLP device index to which the Softpoint structure is wired.

The TLP device index is indicated in the directly represented variable naming used in the Wiring grid of the dictionary's variable grid. For example, the %QU19.0 means that the TLP device index is 19.

2. In the I/O Wiring view, locate and expand the TLP device having the corresponding index.

The TLP device index is the number appearing at the left in a device's name. In the following example, the device index is 19:

19:DS800:TLP:TLP_SOFT_POINT_WRITE(* *)

3. Expand the Parameters element, then double-click any field.
4. In the I/O Parameters dialog, locate the field for which to disable the write permission, then set its Value to FALSE.
5. Click **OK**.

The selected SoftPoint structure field's write permission is disabled.

Initial Values

Initial values can only apply to variables. If no initial value is entered in the variables grid, a value of 0 (or FALSE) is used by default.

The initial values are:

| Variable | Default | Possible Values |
|----------|---------|---|
| BOOL | FALSE | TRUE or FALSE |
| SINT | 0 | any other short integer value |
| USINT | 0 | any other unsigned short integer value |
| BYTE | 0 | any byte value |
| INT | 0 | any other integer value |
| UINT | 0 | any other unsigned integer value |
| WORD | 0 | any other word value |
| DINT | 0 | any other double integer value |
| UDINT | 0 | any other unsigned double integer value |
| DWORD | 0 | any other double word value |
| LINT | 0 | any other long integer value |
| ULINT | 0 | any other unsigned long integer value |
| LWORD | 0 | any other long word value |
| REAL | 0.0 | any other float value (not double). Scientific format 1.2E+10 can be entered |
| LREAL | 0.0 | any other float value. Scientific format 1.2E+10 can be entered |
| TIME | t#0s | any other timer value using the following syntax: t#WhXmYsZms or t#Z 0 <= W: number of hours 0 <= X: number of minutes 0 <= Y: number of seconds 0 <= Z: number of milliseconds Note: The h, m, s, and ms fields are optional. If t#100 is entered, it corresponds to t#100ms. |

| | | |
|--------------------------|--------------|--|
| DATE | d#1970-01-01 | any other date value ranging from 1970-01-01 to 2038-01-18 using the following syntax: d#yyyy-mm-dd |
| STRING | empty | any set of characters contained within single quotes, for example, 'hello' |
| Array initialization | 0 or FALSE | you need to initialize each element of an array* |
| Structure initialization | 0 or FALSE | you need to initialize each field in a structure* |

* When initializing the values of elements for arrays or structures, the total number of characters, including commas automatically inserted to separate the initial values defined for each element, cannot exceed 482. In the Dictionary window, the Initial Value field at the root of the array or structure displays this cumulation.

| All variables | | | | | |
|-----------------|-------|------|---|---|------------|
| Name | Alias | Type | 0 | Init. value | Dimensi... |
| Temperature | | DINT | | 1,2,3,2(4),5,10,75,45,175,25,86,32,5000 | [1..25] |
| Temperature[1] | | DINT | | 1 | |
| Temperature[2] | | DINT | | 2 | |
| Temperature[3] | | DINT | | 3 | |
| Temperature[4] | | DINT | | 4 | |
| Temperature[5] | | DINT | | 4 | |
| Temperature[6] | | DINT | | 5 | |
| Temperature[7] | | DINT | | 10 | |
| Temperature[8] | | DINT | | 75 | |
| Temperature[9] | | DINT | | 45 | |
| Temperature[10] | | DINT | | 175 | |
| Temperature[11] | | DINT | | 25 | |
| Temperature[12] | | DINT | | 86 | |
| Temperature[13] | | DINT | | 32 | |
| Temperature[14] | | DINT | | 5000 | |

To initialize the elements of an array

You initialize an array one element at a time.

1. Set the dictionary to grid mode.
2. Expand the array by clicking the '+' sign.
3. Double-click the array element's Init Value column.
4. Enter a value corresponding to the element type.

To initialize the fields of a structure

The first line, with the structure's name, displays the list of each field's values. You initialize a structure one field at a time.

1. Set the dictionary into the grid mode.
2. Expand the structure by clicking on the '+' sign.
3. Double-click on the structure field, in the 'Initial Value' column.
4. Enter the value that corresponds to the field's type.

The first line, with the structure's name, displays the list of field's values. The parenthesis display a list of values that correspond to the array's elements.

Validation

Validation is performed at all levels of input and use of the grid.

When an error is detected, a message box with an error description appears.

Cell-level Validation

The system processes cell-level validation:

- IEC 61131 compliance checks of:
 - Variable, Array and Structure names
 - Dimensions
 - Initial values
 - Text length (for example, Comments, Alias)
- Variable, Array, and Structure names cannot be the same as Reserved Keywords
- Validity and range checks of Addresses

Row-level Validation

The system validates the rows (records) at grid-level. When editing a row, the system checks:

- Retain variables cannot be Input/Output
- If Type is not of type String, the "(" column must be empty Direction checks:
 - Internal: Wiring must be empty
 - Input: Wiring must begin with %I
 - Output: Wiring must begin with %Q

- Attribute checks:
 - Can only be "Read-only" for Inputs
 - Can only be "Write" or "Free" for Outputs
- Inputs cannot have an initial value
- Retain variables cannot have an initial value

Database-level Validation

The system validates the database. When saving, the system checks:

- IEC 61131 functions have one (and only one) Output parameter, named as the function
- Function and function block parameter names are not duplicated
- Parameters are ordered (Input then Outputs)
- Variable names are not duplicated within a resource
- Local variable names:
 - are not duplicated within a POU
 - do not have the same name as a global variable
- within a Structure, a field is not repeated
- the Maximum number of function and function block parameters respects target capabilities

I/O Wiring View

I/O wiring enables you to define links between the variables defined in a project and the channels of the devices existing on the target system. Wiring is performed at the resource level, therefore, I/O wiring is only available when a resource is selected in either the link architecture or hardware architecture views and when a target has been attached to the current configuration.

After creating variables in the Dictionary, you perform I/O wiring in the I/O wiring tool by adding I/O devices, setting device parameters and I/O filters, then wiring the channels of the devices to variables in the grid. You can also define the mapping of logical channels to physical channels.

To open the I/O wiring tool from the link or hardware architecture view

1. Select a resource.
2. Do one of the following:
 - From the Project menu, choose **I/O wiring**.
 - On the Window Buttons toolbar, click .

To open the I/O wiring tool from the link architecture view

- Within a resource window, open the parameters component, then the I/O wiring component.

Appearance

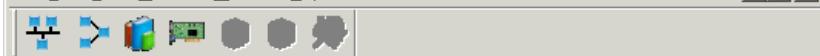
The I/O Wiring view is displayed in the workspace. The menus and toolbar now reflect I/O Wiring options only.

The left of the I/O Wiring Workspace is a hierarchical 'Tree View' of defined I/O devices. The right side of the workspace displays a grid-like table of the free (unwired) variables of the current resource. These unwired variables are listed in alphabetical order. A Splitter is available to change the proportion of the width of the Tree and grid windows.

Title bar



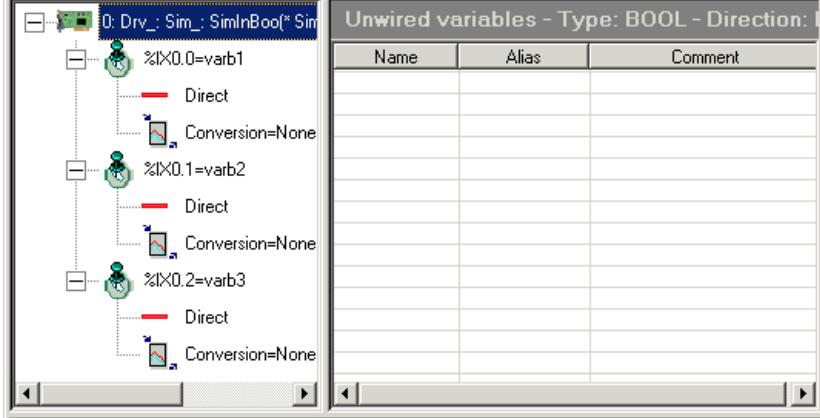
Menu bar



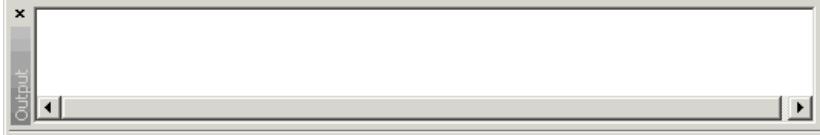
Toolbars



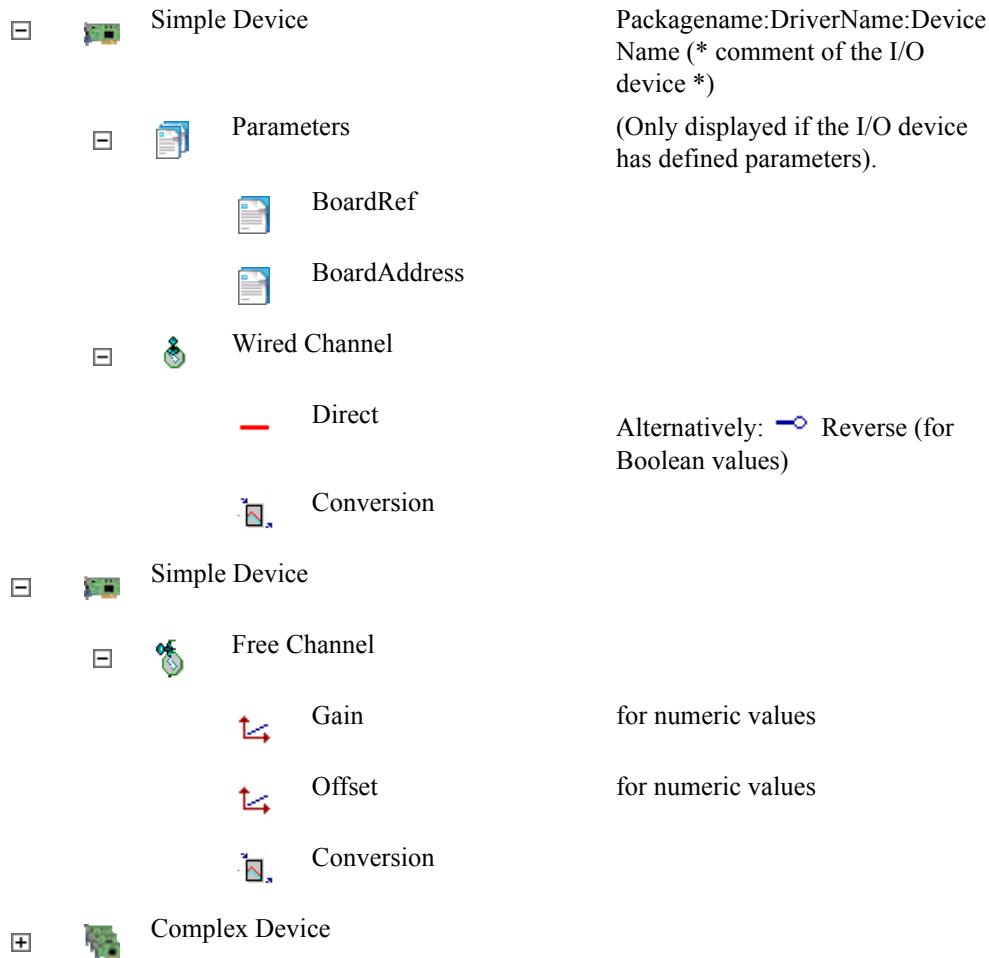
Workspace



Output window



I/O Wiring Tree View



When a device has been added, you can use Direct Variable Representation (%IX1.1) to access IO values. This syntax is shown in the Tree. You can also wire variables that you have already declared in the Dictionary to the device channels, and use these Variable names in your programs to access channel values. The diagram above shows examples of certain simple devices in the I/O Wiring Tree view.

Parameters

- Double-click on any parameter in the tree to open a dialog box that allows you to modify its value.

Direct/ Reverse

For a boolean IO channel, you can switch between the original value (direct) or its negation (reverse). Simply double click on 'Direct' or 'Reverse' to swap from one choice to another.

Gain/Offset

For a numerical channel, you can apply a gain and an offset to a channel value.

For inputs, the original value (coming from the input device) is multiplied by the gain, and the offset value is added. This gives the value used by the programs of the resource.

For outputs, the value of the variable resulting from the execution of the program is multiplied by the gain and the offset value added, before updating the output device.

- Double-click 'Gain' or 'Offset' in the tree to open a dialog box that allows you to modify the values.

Note: Gain is composed of a multiplier factor and a divider factor.

The conversion formula applied is as follows:

$$\text{NewValue} = (\text{Value} * \text{MultFactor}) / \text{DivFactor} + \text{Offset}$$

For details on specific implementations, contact your supplier.

Conversions

Conversions can be applied to any kind of channels. The list of available conversions depends on the target implementation. Please contact your supplier for more information on conversions they provide.

Simply double click on 'Conversion' in the tree to open a dialog box that allows you to select the desired conversion for the channel.

I/O Wiring Grid View

The Grid view displays a read-only list of the available (non-wired) variables of the resource that match the type and direction of the device selected in the Tree View.

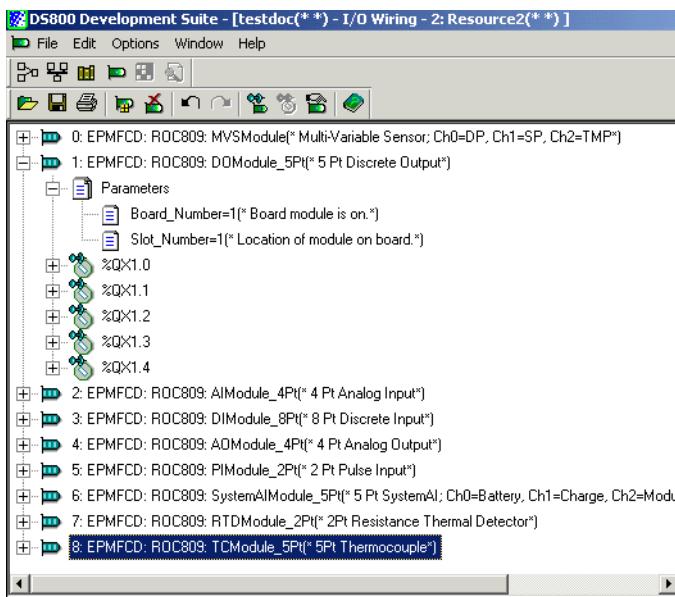
| Unwired variables | | | |
|-------------------|-------|-------------------------|--|
| Name | Alias | Comment | |
| StartProduction | PSTA | User order to start | |
| StopProduction | PSTO | User order to stop | |
| HeatOK | HOK | No error on temperature | |
| PressureOK | POK | No error in pressure | |
| SpeedOK | SOK | No error in speed | |

Working with the I/O Wiring Tool

When defining the I/O Wiring the first time, the Tree and Grid views are empty. After an I/O device is added, the Grid view lists all the variables of the current resource that correspond to the device type and direction. Example: all Boolean inputs for an I/O device: BOOL - Input.

The I/O devices correspond to I/O modules in the RAS device unit:

- Analog Input
- Analog Output
- Discrete Input
- Discrete Output
- MVS Input
- Pulse Input
- RTD Input
- System Analog Input
- Thermocouple Input



TLP Devices (Automatic Wiring)

When you define TLP variables in the dictionary, these are automatically wired to TLP devices. You may need to modify the write permissions for the fields of Softpoint structures. While modifying the permissions for these fields, avoid interfering with the automatic wiring.

Note: The DS800 TLP I/O device drivers require ROC800-Series firmware 2.10 or greater.

The following devices are available for automatic wiring:

TLP_DINT_READ
TLP_DINT_WRITE
TLP_REAL_READ
TLP_REAL_WRITE
TLP_STRING_READ
TLP_STRING_WRITE
TLP_SOFT_POINT_READ
TLP_SOFT_POINT_WRITE
TLP_REGISTER_READ
TLP_REGISTER_WRITE

Analog Input - 4 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the AI module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set three ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

Analog Output - 4 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the AO module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set 3 ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

Discrete Input - 8 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the DI module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set 3 ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

Discrete Output - 5 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the DO module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set 3 ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

Multi-Variable Sensor Input - 6 Point

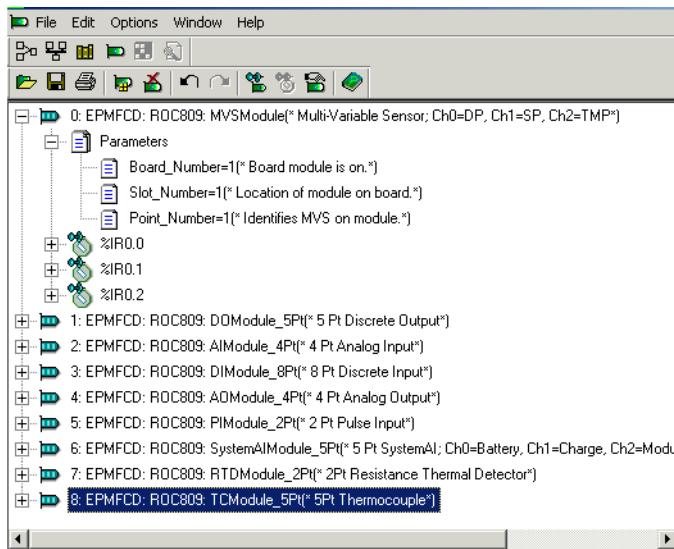
This Wiring is set in the I/O Wiring View. The three channels refer to the three points received from the MVS Sensor:

- Differential Pressure (DP)
- Static Pressure (SP)
- Temperature

For more information on these points, refer to the ROCLINK 800 Configuration Software User Manual.

The parameters are Board number, Slot number, and Point number. Board number is always set to 1. The Slot number refers to the slot in the RAS device housing in which the MVS module resides. The Point number refers to which of the 6 MVS sensor points is being used. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name (tag) should identify the I/O module. The direction can be set three ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it cannot be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).



Pulse Input - 2 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number refers to which RAS device PI Module is being wired (as more than 1 PI module may be used), and the Slot number refers to the slot in the RAS device housing in which the PI module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each I/O module must be defined as a variable in the Dictionary View Variables Grid. The Name should be A (Analog), D (Discrete), P (Pulse), MVS, RTD, SA (System Analog), or TC, the direction should be I (Input), O (Output), or Internal (for System AI). The Attribute should be R (read), W (write), or Free.

RTD Input - 2 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the RTD module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set 3 ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

System Analog Input - 5 Point

This Wiring is set in the I/O Wiring View. The 5 points (channels) refer to inputs for:

- +/- BATT (supply from Battery to power supply module is low)
- +/- CHG (Supply to solar Charge is low)
- Module V (Voltage monitor from power supply module to I/O and Comm modules)
- One point that is not used
- Board Temperature (monitor of thermistor on CPU)

For more information on these points, refer to the ROCLINK 800 Configuration Software User Manual.

The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number is fixed (non-configurable) as it has no physical location in the RAS device housing. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set three ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it cannot be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

Thermocouple Input - 5 Point

This Wiring is set in the I/O Wiring View. The parameters are Board number and Slot number. Board number is always set to 1, and the Slot number refers to the slot in the RAS device housing in which the T/C module resides. The I/O module wiring in DS800 software is NOT self-identifying, as it is in ROCLINK 800 software.

Before wiring, each variable must be defined in the Dictionary View Variables Grid. Select the appropriate variable from the unwired variables list. The Name should identify the I/O module. The direction can be set 3 ways. If defined as I (Input) then the variable will have to be wired to an Input module, if defined as O (Output) then the variable will have to be wired to an Output module, or if defined as Internal then it can not be wired to a ROC I/O module. The Attribute should be R (read for Inputs), W (write for Outputs), or Free (for Inputs or Outputs that will be used as Inputs in other locations in the algorithm).

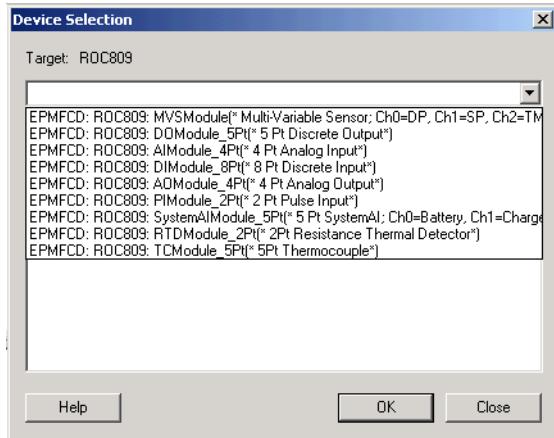
Adding I/O Devices

You can add simple and complex devices to the I/O wiring tree. Available devices for a target are displayed in the device selection list. When adding complex devices, the number of channels, i.e., device size, of individual simple devices making up a complex device varies depending on the definition of the complex device in the target.

To add an I/O device

1. From the Edit menu, choose **Add I/O Device** or click  on the I/O Wiring toolbar.

The Device Selection dialog box appears:



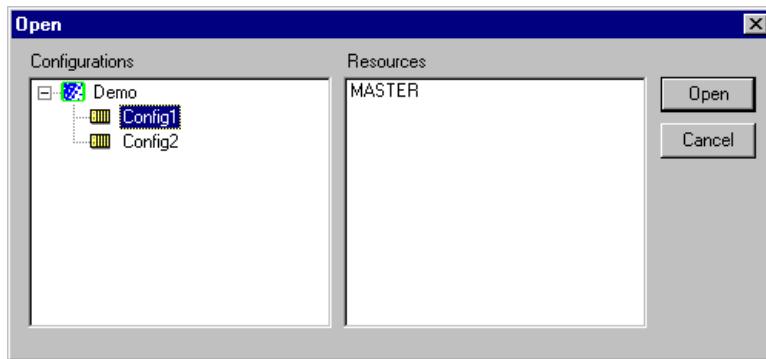
2. Choose the device from the pull-down menu.
3. Change the device index and number of channels (if required and available).
4. Click **OK**.

Opening Devices

You can open existing devices defined for any resource of a project.

To open an existing device

1. From the File menu, choose **Open Device** or click  on the I/O Wiring toolbar.
2. In the Open window, browse to select the resource holding the device, then click **Open**.



The devices defined for the selected resource are displayed in the I/O Wiring View.

Deleting Devices and Conversions

You can delete devices and conversions from the I/O wiring view. You cannot delete Parameter, Gain, or Offset elements. You remove a current conversion by replacing it with "None". You can also disconnect variables attached to selected channels.

When deleting devices, all variables are unwired from the device (as with Free I/O device channels).

To delete a device or conversion

You can delete devices or conversions using the main menu or the I/O Wiring toolbar.

- ▶ From the Edit menu, choose **Delete Device** or click  on the I/O Wiring toolbar.

Setting the Real or Virtual Attribute

This command sets the Real/Virtual attribute for the currently selected device.

To toggle the Real/Virtual attribute:

1. Select the device in the Tree View.
2. From the Edit menu, choose **Real / Virtual I/O Device** or click  on the I/O Wiring toolbar.

The Tree View icon for a virtual device is .

In Real Mode, I/O variables are directly linked to the corresponding I/O devices. Input or Output operations in the programs correspond directly to the input or output conditions of the actual I/O device fields. In virtual mode, I/O variables are processed as internal variables. They can be read or updated by the Debugger so that the user can simulate the I/O processing, but no actual connection is made.

Wiring Channels

You wire variables to channels by selecting a channel in the Tree, then double-clicking or pressing <Return>on a variable in the grid. If the channel is already wired, the existing variable is unwired and replaced by the one in the grid.

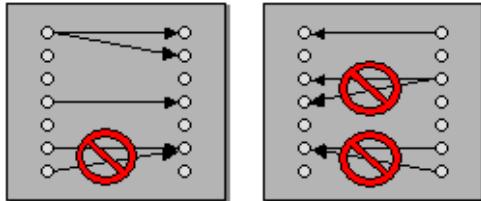
After a connection, the variable is removed from the grid and the next channel is selected; only variables available for wiring appear in the grid.

Note: If no channel is selected, nothing happens.

Mapping Channels

You can define the mapping of logical channels to physical channels. When mapping channels, only one link can send to or receive from a logical channel. For an input device, you can map a physical input to one or more logical inputs. Whereas, you cannot map more than one physical input to a logical input. For an output device, you can only link one logical output to one physical output:

| Input Device | | Output Device | |
|-------------------|------------------|--------------------|-------------------|
| Physical Input | Logical Input | Physical Output | Logical Output |



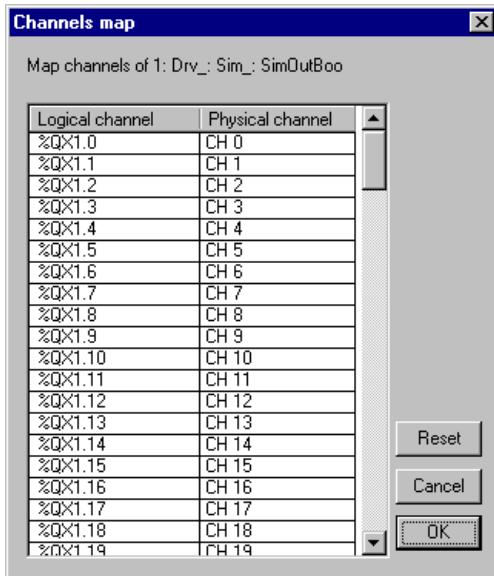
When performing online changes, you can modify channel mappings.

To map logical and physical channels for a device

1. In the I/O wiring tool, select the device.

2. From the Edit menu, choose **Map Channels** or click  on the I/O Wiring toolbar.

The Map Channels editor displays the current mapping of channels for the device:



3. For each logical channel to map, locate and double-click its corresponding physical channel, then from the drop-down list assign the new physical channel by double-clicking it.
4. Click **OK**.

Freeing Channels

You can unwire one or all variables for a selected device.

To free one channel

1. Select a wired channel in the tree view.
2. Do one of the following:
 - From the Edit menu, choose **Free I/O device channel**.
 - From the I/O wiring toolbar, click  on the I/O Wiring toolbar.
 - Press **Delete**.

To free all channels

1. Select a device in the tree view.
2. Do one of the following:
 - From the Edit menu, choose **Free all I/O device channels**.
 - From the I/O wiring toolbar, click .

Run-time System Events

You can log run-time system events on the Windows platform using the Events Logger and view these events using the Events Viewer.

You access the Events Logger and Events Viewer from the Workbench. You can also start the logger and viewer from command lines.

Logging Events

The Events Logger receives events from DS800 targets. You can view these events using the Events Viewer. Events are stored in a log file, in Unicode format, located in the Events Logger folder of the current project's directory. A new log file is automatically created each day at 00:00:00 hours.

The name of the log file is `Events_YYYYMMDD.txt` where *YYYY* is the year, *MM* is the month, and *DD* is the day on which the file is created.

You can open the log file in text format using a text editor.

When starting the Events Viewer from the Workbench while an application is running, the Events Logger automatically points towards the application's project and the logger is started. You can also choose to start the Events Logger from a command line.

To start the Events Logger from a command line

You can set the Events Logger to start for a given Workbench project from a command line using the following syntax:

```
EventsLogger -P"full_directory_path"
```

The executable file for the Events Logger is installed in the following location:

Program Files\Emerson\DS800\bin\EventsLogger.exe

When manually starting the Events Logger, you may need to provide the location of the Workbench project. The Events Logger needs to be started in it's location directory. For example:

```
C:> cd "Program Files\Emerson\DS800\Bin"  
C:> EventsLogger -P"Program Files\Emerson\Projects\DS800\  
Prj\MyProject"
```

You can also start the Events Viewer from a command line.

To open a log file

You can view the log of events as a text file by opening the log in a text editor such as Notepad. The default location for the log file is in the Events Logger folder of the current project's directory.

- ▶ Locate and double-click the .txt file.

The file opens in the associated text editor.

Viewing Events

The Events Viewer displays run-time system events logged with the Events Logger.

The Events Viewer displays the contents of the log file, created daily by the Events Logger. In the viewer, events appear as they occur. You can sort events according to the categories at the top of the viewer window:

- Date and time when the event took place
- Level, the level of the event. Possible values include Error, Warning, and Info.
- Module, the module sending the event
- Sub-module, the sub-module sending the event
- Error, the code number of the error
- Description, a textual description of the event

- Value, a number relating to target development values
- Configuration, the name of the configuration running on the target that sent the event. When the event is related to a resource, the resource name is added to the configuration name, for example, Config1.Res1.

You can choose to view events for a day other than the current day. You can also view events for a day in a different month and year as long as the log file for the specified date is available. Furthermore, you can sort the contents of the viewer according to individual columns in ascending or descending order by clicking a column heading a first time for ascending order and a second time for descending order.

When viewing events, you can access more detailed information for specific messages by pressing F1.

Note: The Events Logger only logs target errors; Simulator errors are not displayed in the Events Viewer.

To access the Events Viewer

When starting the Events Viewer while running an application, the Events Logger automatically points towards the application's project and is started.

- In the Workbench, from the Tools menu, choose **Events Viewer**.

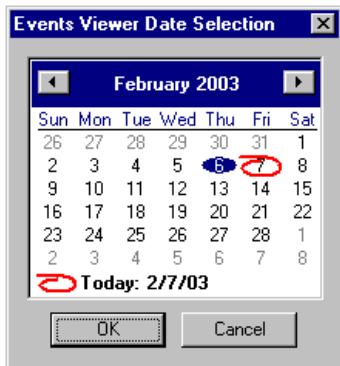
To sort events in the Events Viewer

- At the top of the window, click a category heading. To inverse sorting order, click the category heading a second time.

To view events for another day

1. At the top of the window, click the date.

The Events Viewer Date Selection window appears:



2. To view the events for another day, click the day on the calendar.
3. To view the dates for another month, do one of the following:
 - Click or to scroll through the previous or following months.
 - Click on the month at the top of the calendar, then choose one from the list.
4. To change the year, click the year at the top of the calendar, then choose one from the list.
5. To return to viewing events for the current day, click below the calendar.

To start the Events Viewer from a command line

You can set the Events Viewer to start for a given Workbench project from a command line using the following syntax:

```
EventsViewer -P"full_directory_path"
```

The executable file for the Events Viewer is installed in the following location:

Program Files\Emerson\DS800\bin\EventsViewer.exe

The Events Viewer needs to be started in it's location directory. For example:

```
C:> cd "Program Files\Emerson\DS800\Bin"  
C:> EventsViewer -P"Program Files\Emerson\Projects\DS800\  
Prj\MyProject"
```

You can also start the Events Logger from a command line.

Language Editors

The Workbench holds several language editors, having some Common Editor Features, for use with the many supported languages.

SFC Editor

FC Editor

Multi-language Editor

Composite IEC 61499 Editor

Common Editor Features

Each Editor in the Workbench has a similar and consistent interface using standard Windows layout and functionality (for example, menus, toolbars).

The Dictionary, listing variables that can be used in the current POU, or used to declare new variables, can be opened from any Editor. Building POU Code and starting Test Mode can also be performed from all the Editors.

Printing from an Editor launches the Document Generator with elements specific to that Editor.

Appearance

title bar

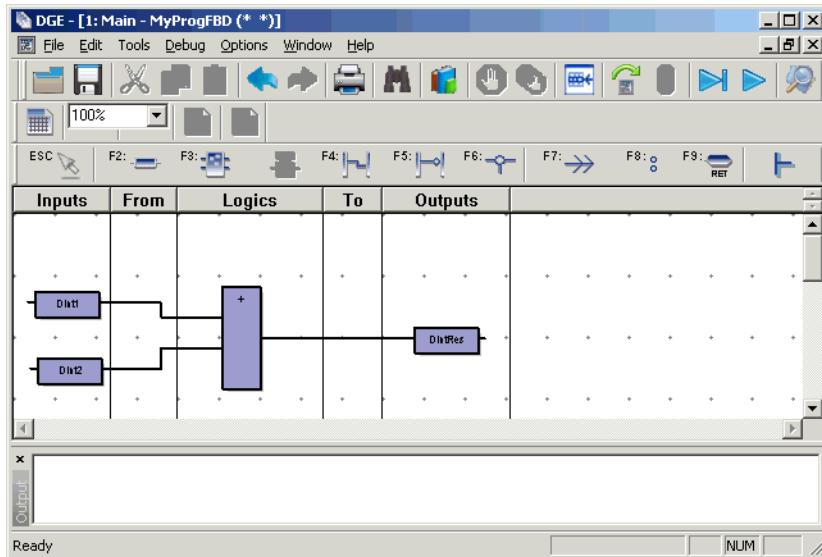
menu bar

toolbars

workspace

output
window

status bar



Title Bar

For help locating the title bar, see the Appearance diagram. The title bar displays the application name and filename of the active Program.

Control Icon

At the left end of the title bar is the Control Icon, which is used to access the Control Menu (see following section). Double-clicking on the Control Icon closes the Editor.

Control Menu

Clicking on the Control icon opens the Control Menu. The Control Menu is used to position the Main Window or to exit.

Window Buttons

The standard window buttons appear at the right end of the title bar. Use these to resize or close the Window.

Menu Bar

The Menu Bar contains the Editor's menus. For help locating the menu bar, see the Appearance diagram. Each menu lists a "family" of selections, each selection performs a specific action.

Note: Menus that are not currently available are temporarily removed from the menu bar. Menu Items not available are displayed in gray.

Using the Menus

1. Open a menu by clicking on it, or by pressing <Alt> plus the letter that is underlined in the menu's title. For example, to open the File Menu, you press <Alt> + <F> (shown in this User's Guide as ALT+F).
2. Choose a menu selection by clicking on it, by pressing its underlined letter, or by using the cursor keys to highlight it and then pressing <Enter>. Menu selections that appear in grey are not currently available.

Control Icon

When a Program is open, the menu bar has a Control Icon on the left.

Control Menu

Clicking on the Control Icon opens the Control Menu. The Control Menu is used to position the Window or to alternate between them.

Window Buttons

The standard window buttons appear at the right end of the menu bar.

Toolbars

The language editors holds toolbars performing various functions.

Displaying the toolbars

To show or hide a toolbar

1. From the Options Menu, choose **Layout**.

The Layout Dialog Box appears.

2. Check / uncheck the names of the toolbars to show / hide.

Moving toolbars

The toolbars can be placed anywhere on the screen.

To move a toolbar

1. Point the cursor at the toolbar's title bar or main panel.

Note: Do not point at the control icon or one of the window's buttons.

2. Press and hold the left mouse-button.

3. Drag the toolbar by moving the mouse.

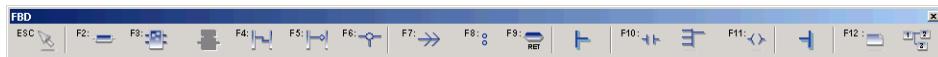


4. Release the mouse-button.

Docking toolbars

- Dock a toolbar to a side of the Workspace by positioning it at the Workspace's edge, this toggles between a toolbar's floating and docked states.

The toolbar shown above appears as follows in its floating state:



Standard Toolbar



Opens a POU



Saves the current POU



Cuts the selection and places it on the clipboard



Copies the selection and places it on the clipboard



Pastes the contents of the clipboard



Undoes the last operation



Redoes the last operation



Accesses the document generator



Finds and replaces items



Accesses the Dictionary view



Sets or removes a breakpoint



Removes breakpoints



Inserts identifiers



Builds the current POU



Stops a build



Switches the application to debug mode



Switches an application to simulation mode



Accesses the cross references browser

Options Toolbar



Displays the grid



Adjusts the zoom



Increases the X to Y Ratio (LD Only) Cells are displayed wider



Decreases the X to Y Ratio (LD Only) Cells are displayed narrower

Debug Toolbar

The Debug toolbar is accessible when you run a POU in either debug or simulation mode.



Starts all stopped resources



Starts a stopped resource



Stops all running resources



Stops a running resource



Switches the application to Real-time mode



Switches the application to cycle-to-cycle mode



Executes one cycle



Steps to the next **line of code or rung**



Steps into the next **line of code or rung**



Locates the current step



Sets the cycle timing



Sets or removes a breakpoint. For LD programs only.



Removes breakpoints. For LD programs only.



Show/Hides output values. For FBD programs only.



Debugs a function block



Displays the spy variable list



Stops the debug/simulation mode



Refreshes the status of resources



Clears the output window

SFC Breakpoints Toolbar



Sets a breakpoint on step activation



Sets a breakpoint on step deactivation



Sets a breakpoint on transition



Removes a breakpoint



Removes all breakpoints



Clears a transition

SFC Tools



Select



Insert Initial Step



Add a Step



Add a Transition



Add an OR Divergence



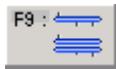
F6 : Add an OR Convergence



F7 : Add an AND Divergence



F8 : Add an AND Convergence



F9 : New Branch



F10 : Add a Link



F11 : Jump



F12 : Renumber



Add Action Block



Move Action Block Up



Move Action Block Down



Delete Action Block

Flow Chart Tools



Select



Insert Action



Insert I/O Specific Action



Insert Test



Insert Flow



Insert Connector



IF-THEN-ELSE



DO-WHILE



WHILE-DO



Insert Sub-Program



Renumber Flow Chart



Insert Comment

ST Tools

This toolbar is displayed when editing an ST POU, an Action, or a test of an FC or SFC POU written in ST. Clicking on one button of this toolbar inserts the corresponding word, at the caret position, in the text of the current POU.

| | |
|--|----------------------|
| | Assignment |
| | Boolean True |
| | Boolean False |
| | Boolean AND operator |
| | Boolean OR operator |
| | Boolean XOR operator |
| | RETURN statement |
| | IF Statement |
| | THEN Statement |
| | ELSIF Statement |
| | ELSE Statement |

F10 END_IF

END_IF Statement

F11 CASE

CASE Structure

F12 END CASE

END_CASE Structure

IL Tools

This toolbar is displayed when editing an IL POU or an action or a test of an FC or SFC POU written in IL. Clicking on a button of this toolbar inserts the corresponding word, at the caret position, in the text of the current POU.

TRUE

FALSE

F2 LD

F3 ST

AND

OR

XOR

ADD

SUB

MUL

DIV

F4 LT

F5 LE



LD Tools

This toolbar is displayed when editing an LD POU or an Action or a test of an FC or SFC POU written in LD.

- F2: Insert Contact before
- F3: Insert Contact after
- F4: Insert parallel Contact
- F5: Insert Coil
- F6: Insert Block before
- F7: Insert Block after
- F8: Insert parallel Block

- F9:**  Insert a Jump
- F10:**  Insert RETURN
- Change Coil/Contact Type** (pressing the <spacebar> has the same effect)

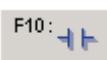
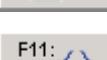
- Insert link**

- Align coils**


FBD Tools

The FBD tools bar is displayed when editing a POU written in the FBD language.

- ESC:**  Selects items
- F2:**  Adds a variable
- F3:**  Adds a function block
- F4:**  Draws a link
- F5:**  Draws a link with negation
- F6:**  Adds a corner
- F7:**  Inserts a jump to label

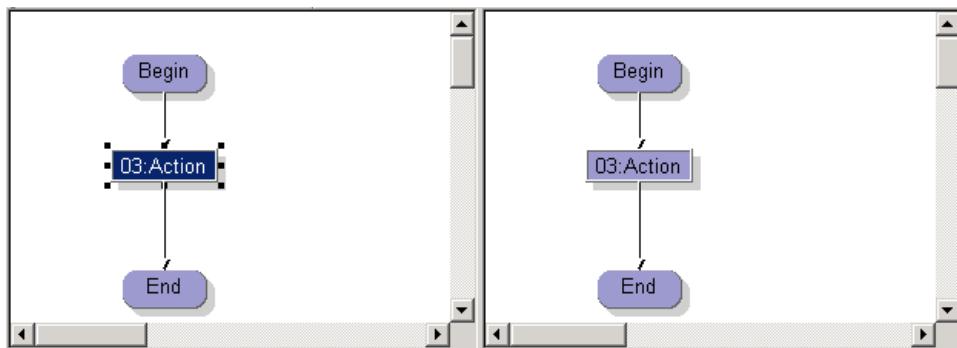
- | | |
|--|---|
|       | <p>F8: Inserts a label</p> <p>F9: Inserts a return</p> <p>F10: Adds a left power bar</p> <p>F11: Adds a contact</p> <p>F12 : Adds an LD vertical connection</p> <p>Change Coil/Contact Type (pressing the <spacebar> has the same effect)</p> <p>Adds a right power bar</p> <p>Adds a comment</p> <p>Shows or hides the execution order</p> |
|--|---|

Workspace

When you open a POU, it appears in a window. This windows appear within the Editor's Workspace.

For the FBD and LD language editors, you can also change the foreground and background colors.

The Workspace of the FC (Flow Chart) and SFC Editors can be sub-divided into two simultaneous views:



Each view can be zoomed independently.

To split the workspace

1. From the Window menu, choose **Split**.
2. Drag and drop the vertical division to the required position.

Grid

The editing grid shows matrix cells. An editor option allows the user to show or hide the grid during development. The grid is very useful for placing new elements.

To toggle (display / hide) the grid



- From the Layout sub-menu of the Options Menu, choose **Grid** or click  on the Options toolbar.

Note: The grid visibility does not affect its use to position elements, simply whether or not it can be seen.

X-Y Ratio

The x-y ratio determines the relative width spacing of the grid compared to the height of each grid 'cell'. This is a display property only, it has no effect on the definition or execution of the Program.

To change the x-y ratio



- From the Options Menu, choose **Layout** OR use the buttons ( , ) on the Options toolbar.

Note: The X-Y ratio features are only available when editing LD.

Contextual Menus

The Contextual Menus are displayed by clicking the right mouse-button in the Editor Workspace.

The commands on the Contextual Menu are generally available in the Edit Menu.

Example



Output Window

To view the output window

- From the Window Menu, choose **Show Output Window**.

The output window appears, docked to the status bar:



Note: The output window is moved like a toolbar. It is automatically displayed when Building and Debugging a Program. Compilation errors are displayed in the output window.

To clear the output window

- From the Window Menu, choose **Clear Output Window**.

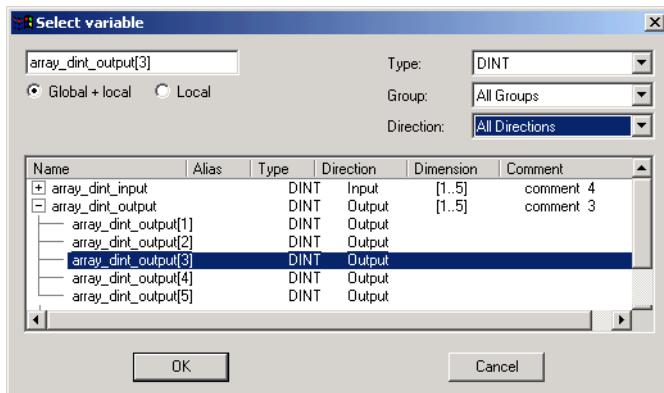
Status Bar

The Status Bar appears at the bottom of the Main Window. Information about commands, operations and POU's is given on the Status Bar.

Inserting Identifiers

You can insert identifiers, i.e., variables, previously declared in the Dictionary. You can also create new variables and enter constant values into a POU as well as access the parameters of functions or function blocks. When creating a new variable, you need to assign a unique name (not corresponding to an existing variable) as well as specify its type and scope: global or local to the POU. These variables are added to the project database with default values for their other attributes (Internal, Free). For new variables of the STRING type, a string of 80 characters is automatically defined.

You insert identifiers using the Select Variable dialog. You can list all types of variables or individual standard IEC 61131 types as well as defined words, arrays, and structures. You can also list variable groups and variable directions. When editing functions or function blocks, the parameters option appears in this list. When typing identifier names, the selector automatically searches for the first item in the list matching the entered criteria.



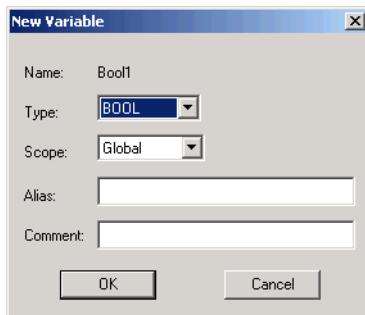
Note: Arrays must be declared in the Dictionary View before using them in Functional Block Diagrams (FBD).

To insert an identifier in a POU

- From the Edit menu, choose **Insert Identifier** or click  from the Standard toolbar.

The Select Variable dialog box is displayed.

2. To reduce the number of variables appearing in the list, select a type, variable group, and direction of the identifiers to list. To list the parameters for functions and function blocks, select the Parameters option.
3. Do one of the following:
 - To use a previously declared variable, select a variable from the list or type the name of the variable in the field at the top left.
 - To create a new variable, in the top left field, type a unique name and click **OK**, then in the New Variable dialog box, specify the type and scope for the new variable (optionally an alias and comment). To specify the local scope, select the name of the currently edited POU.

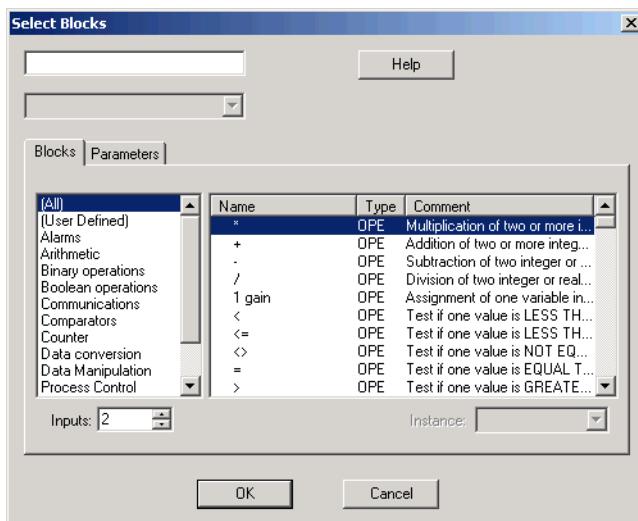


- To enter a constant value, type the value in the field at the top left.
4. Click **OK**.

The identifier is inserted in the currently edited POU at the current position.

Inserting Blocks

You insert blocks, i.e. operators, functions, and function blocks into programs from the Select Blocks window. The items displayed in the list depend on the program type. For SFC, FC, ST, LD, FBD, and IL programs, the available items are operators (OPE), standard functions (SFU), standard function blocks (SFB), user IEC 61131 Functions (IFU), user IEC 61131 Function Blocks (IFB) and all "C" Functions (CFU) and Function Blocks (CFB) supported by the target attached to the current resource.



- The block identifier field (top left) indicates the selected operator, function, or function block. When an instance is selected, the instance name is displayed.
- The Blocks list enables you to display all or various types of operators, functions, and function blocks.
- Inputs are only available for operators such as +, *, and AND to define the number of input connections for the block.

Inputs:

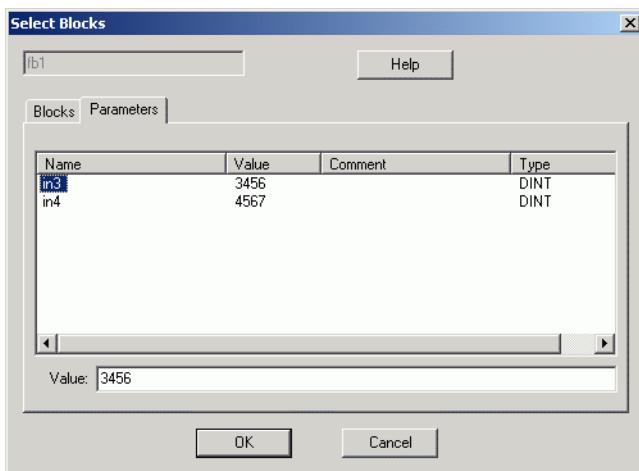
- For FBD 61131 programs, the Instance field is only available when the currently selected Block is a declared instance. The Instance field enables you to select the Instance name to

insert into the POU. When the field is left blank, an automatic instance is created for the function block.



The Help button displays the description of the Block or Function or the associated help if it exists (C Function or Function Block).

The Parameters tab is significant only for some "C" Functions and Function Blocks. It shows the Parameters that are not shown when inserting the block in the program editor. These Parameters are called "Hidden Parameters". They correspond to Input Parameters of the Block to which you can give a constant value. The Parameters tab allows you to enter a value for these Parameters.



Select the parameter name in the list, and enter its value in the "Value" edit box, press Enter to assign the value.

Printing POUs

You can choose to print a standard list of elements for a POU from the Document Generator. For information about the Document Generator, see page 337.

To print the current POU



- From the File menu, choose **Print** or click on the Standard toolbar.

Opening the Dictionary

From a language editor, you can open the Dictionary filtered for the current POU.

To open the Dictionary



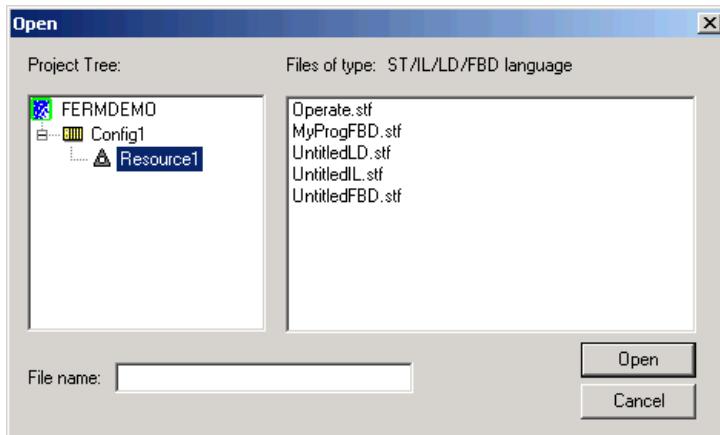
- From the File menu, choose **Dictionary** or click on the Standard toolbar.

Opening Another POU

From a language editor, you can open another POU written with the language supported by the current editor from any resource.

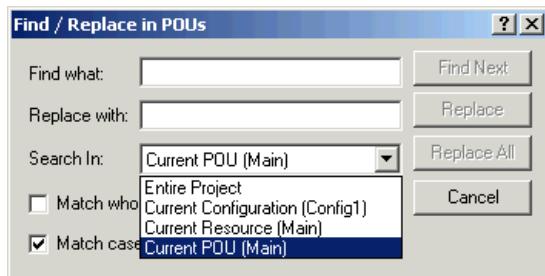
To open another POU from a language editor

1. From the file menu, choose Open or click  on the Standard toolbar.
2. In the Open dialog box, from the project tree, select the resource holding the POU to open, then the file from the list of available files.



Finding and Replacing in POU

You can find and replace text throughout all POU. You can specify to search an entire project, a configuration, a resource, or a POU.



Searches include level 2 code of SFC and FC POU as well as action block names of steps. The Find / Replace in POU utility is not case-sensitive, for instance, FIND is the same as FinD.

To find a step or transition name in an SFC chart or an action or test name in an FC chart, use the Goto command in the Edit menu from the respective editor.

To find an item (characters, word, or phrase)

Searches are performed from top to bottom and from left to right.

1. From the Edit menu, choose **Find / Replace in POU** <Ctrl+F> or click  on the toolbar.

Note: While in the Dictionary view, the toolbar element accesses the Dictionary grid Find/Replace utility and the shortcut key for the Find/Replace In POU menu item is Ctrl+Shift+F.

2. Enter the item to search for. To perform a case sensitive search, check **Match Case**.
3. To find the next occurrence of the item, click **Find Next**.

4. To replace found items, in the Replace field, enter the text to replace, then do one of the following:
 - To replace the found occurrence, click **Replace**.
 - To replace all occurrences of an item, click **Replace All**.

SFC Editor



The SFC (**S**equential **F**unction **C**hart) **E**ditor is launched automatically when an SFC program is opened from the Workbench. The SFC language is used to describe operations of a sequential process. It uses a simple graphic representation for the different steps of a process, and conditions that enable the change of active steps. An SFC Program is entered by using the graphical SFC editor.

SFC is the core of the IEC 61131-3 standard. The other languages (except Flow Chart) usually describe the actions within the steps and the logical conditions for the transitions. The SFC editor allows the user to enter complete SFC programs. It combines graphic and text editing capabilities, thus allowing the entry of both the SFC chart, and the corresponding actions and conditions.

The SFC editor is automatically opened when an SFC program is edited.

Note: Before creating new programs, you need to close the Dictionary.

To subsequently open another program from the SFC Editor

- From the File Menu, choose **Open (CTRL+O)** or click  , on the Standard toolbar.

Appearance

Title Bar

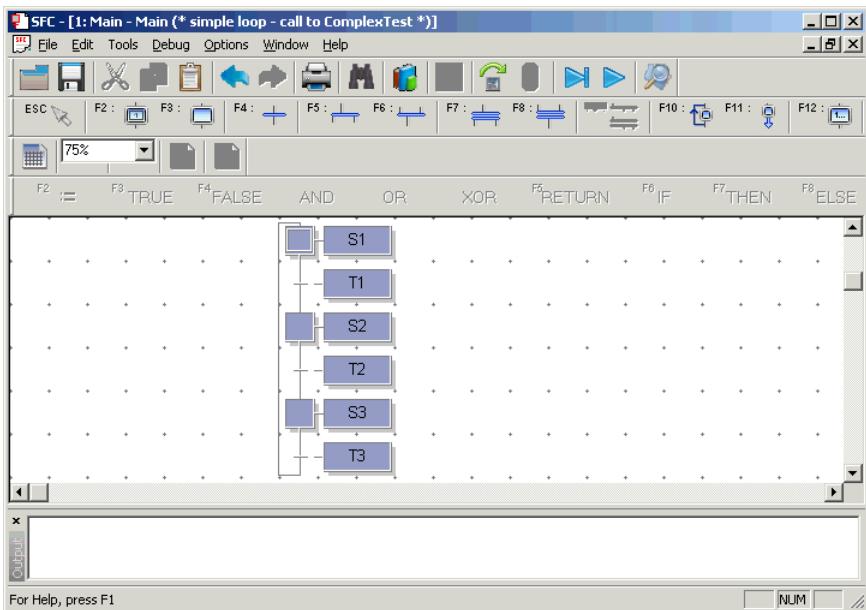
Menu Bar

Toolbar

Workspace

Output Window

Status Bar



Menu Bar

Some options are available as keyboard commands.

| | | | |
|------|------------------------|--------|---|
| File | Open | Ctrl+O | opens an existing POU |
| | Close | Alt+F4 | closes the POU |
| | Save | Ctrl+S | saves the current POU |
| | Build Program | Alt+F3 | builds the code for the current POU |
| | Stop Build Program | | stops the build in progress for the current POU |
| | Dictionary | Ctrl+D | opens the dictionary filtered for the current POU |
| | Description | Ctrl+K | accesses the program description |
| | Print | Ctrl+P | prints the current POU |
| | Exit | Ctrl+Q | leaves the language editor |
| Edit | Cut | Ctrl+X | removes the selected item and places it on clipboard |
| | Copy | Ctrl+C | takes a copy of the selected item and places it on the clipboard |
| | Paste | Ctrl+V | inserts the contents of the clipboard into the selected item |
| | Delete | DEL | removes the selected item |
| | Undo | Ctrl+Z | cancels the last action |
| | Redo | Ctrl+Y | restores the last cancelled action |
| | Find / Replace in POU | Ctrl+F | finds and replaces text in a project, a configuration, a resource, or a POU |
| | Go to | Ctrl+G | jumps to the indicated step or transition number |
| | Rename Step/Transition | Ctrl+R | renames the element |
| | Renumber | | renumbers all elements in the chart in sequential order |
| | Add Action Block | | add an action block |

| | | | |
|---------------------|---------------------------------|------------|---|
| Edit (Continued) | Delete Action Block | | deletes an action block |
| | Edit Level 2 | Enter | opens the level 2 programming for an element |
| | Edit Level 2 in Separate Window | Ctrl+Enter | opens the level 2 programming for an element in a separate window |
| Tools | Insert/Set Identifier | Ctrl+I | accesses the Select Variable dialog box where you can insert a variable |
| | Insert New Rung | Ctrl+R | inserts a rung |
| | Browser | Ctrl+B | accesses the Cross References browser listing and localizing all instances of global variables and I/Os declared in a project |
| Debug | Debug | Alt+F6 | switches the application to debug mode |
| | Simulation | Alt+F7 | switches the application to simulation mode |
| | Debug FB | F11 | opens a selected function block in the language editor with its instantiation values |
| Options | Set Level 2 Language | | sets the programming language used for level 2 programming. For programs, possible languages are ST, IL, and LD. For function blocks, possible languages are ST and LD. |
| | Customize | Ctrl+U | accesses the customization properties for Workbench views and editors |
| | Target/Code Settings | | accesses the compilation options for the POU |

| | | | |
|--------|---------------------|--------|---|
| Window | Cascade | | sets the different views of the project to appear in a cascading manner |
| | Tile | | sets the different views of the project to appear in a tiled manner |
| | Split | | splits the workspace into two simultaneous views |
| | Show Output Window | Ctrl+4 | displays the output window below the workspace |
| | Clear Output Window | | clears the contents of the output window |
| Help | Contents | F1 | accesses the online help |
| | Search Help On... | | not currently supported |
| | About | | displays product and version information |
| | Support Info | | not currently supported |

Working with the Editor

The SFC language is used to represent sequential processes. The SFC programming is usually separated into two different levels:

- Level 1 shows the graphic chart, the reference names of the steps and the transitions, and the comments.
- Level 2 is the ST, LD or IL programming of the actions within the steps, or the conditions attached to the transitions. Actions or conditions may refer to functions written in other languages (FBD, LD, ST or IL). The level 2 programming of a step includes action blocks programmed in ST, LD or IL. The level 2 programming of a transition describes a Boolean condition entered in ST, LD or IL.

Individual elements are automatically linked if the SFC editor considers them to be in a valid position.

From the editor, you can:

- Build the current program code to check your program and prepare the code for building the resource code.
- Print your program.
- Launch the Dictionary.

You can also enter a description to document your Program

- From the File menu, choose **Description**.

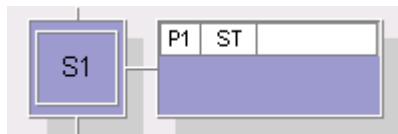
SFC Elements

To draw an SFC chart, you simply introduce the significant components of the chart. The SFC editor automatically draws most of the single lines joining two elements (horizontally or vertically). Lines (or links) can be drawn manually.

To place an SFC component on the chart, the user has to select the type of the component in the editor toolbar and then click in the edition workspace at the desired position. If the mouse button is kept depressed, a "ghost" of the element is shown in order not to place it blindly. When the symbol is placed, links can be created automatically depending on the element position regarding the existing elements. The editor may not accept the placement of the element. For example, you cannot place an element over an existing element.

Initial Step

Every SFC program must have an **Initial Step**. Initial steps are double bordered. For information about initial steps, see page 396.

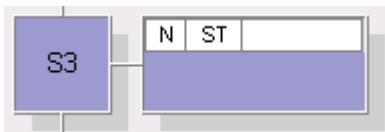


To place an Initial Step

1. On the SFC toolbar, click .
2. Click in the workspace at the desired position.

Step

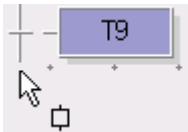
Steps are given sequentially numbered default names, e.g. S1, S3, S5... For information about steps, see page 396.



To place a Step

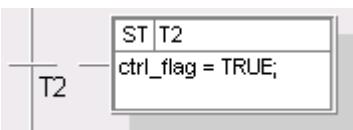
1. On the SFC toolbar, click
2. Click in the Workspace at the desired position.

Note: To link a step to an existing transition, place the mouse cursor on the grid cell above or below the transition.



Transition

Transitions are given sequentially numbered default names, e.g. T2, T4, T6... For information about transitions, see page 397.

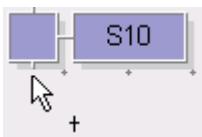


To place a Transition



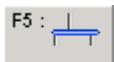
1. On the SFC toolbar, click .
2. Click in the workspace at the desired position.

If you want to link the transition to an existing step, click the mouse with the cursor on the grid cell above or below the step.

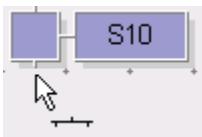


Divergence/Convergence

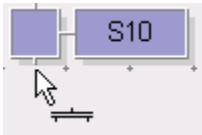
To place a **divergence**, click on the button on the SFC Toolbar then click in the chart workspace at the desired position.



- To link an OR divergence () to an existing step, place the mouse cursor on the grid cell below the step.

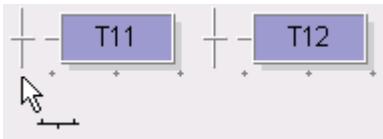


- To link an AND divergence () to an existing transition, click grid cell below the transition:



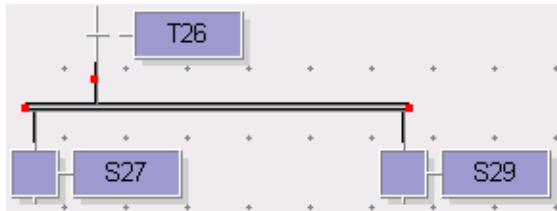
- To place a **convergence** and attach it to previous elements, click the left-most branch. OR

convergences () are attached to the preceding transitions, AND convergences () are attached to preceding steps.



Creating New Branches

Inserting a new branch creates an alternative routing for connections.



To insert a new branch on a divergence or convergence

1. Select a divergence / convergence.



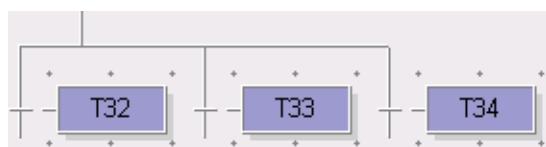
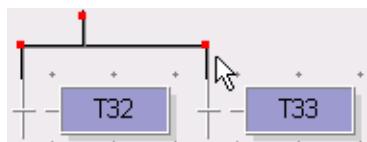
2. On the SFC toolbar, click .

Note: Moving the upper handles, on the left or right of a divergence or convergence, automatically causes a new branch to be created.

To create a branch next to existing branches, select the divergence (OR divergence for transitions, AND divergence for steps), then press F9 or click the new branch icon and add an element (transition or step).

Example

Right-click or press F9 to add a branch Add element on new branch

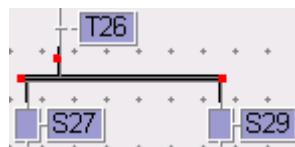
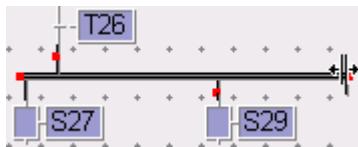


Deleting Branches

Moving non-connected branches back onto the nearest connected branch deletes 'extra' branches.

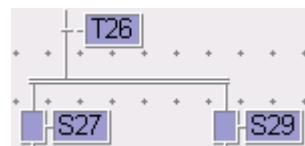
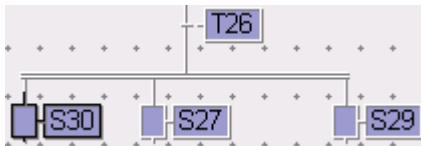
Select the divergence, place the cursor on the upper-right handle (red square), then drag the branch onto the branch with S29:

Select branch end and move towards
existing element Branch is removed from
divergence



Deleting an element removes the branch directly above it.

Select and delete element Element and branch are removed

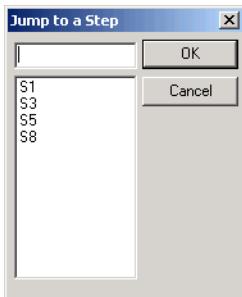


Link

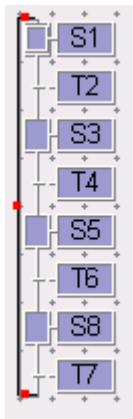
Drawing a **link** is a drag and drop operation linking one element to another. Links always move from a step to a transition or from a transition to a step. Links can be moved using drag and drop operations on the handles (red squares), displayed when the link is selected. For information about oriented links, see page 398.

To insert a link

Inserting a link with a single click on the link origin, or dropping the link in an empty area of the workspace, displays the Jump to a Step dialog. The "Jump to a Step" dialog is only displayed when the link origin is a transition.



- Choose the required step name then click **OK**.



Jump

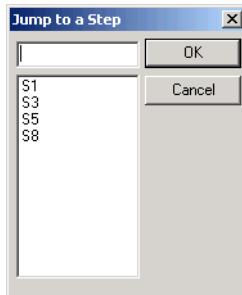
You can insert jumps between transitions and steps. For information about jumps to steps, see page 398.

To insert a jump

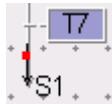
1. On the SFC toolbar, click  .

The cursor changes to a 'Jump' cursor.

2. Click on the workspace, immediately below the transition to jump 'from'.
3. In the Jump to a Step dialog, select the required step name then click **OK**.



The step name is indicated next to the jump symbol.

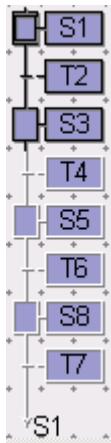


Managing Elements

SFC elements can be cut, copied, and pasted within a Sequential Function Chart or, if more than one is open, between different charts. When an element is moved, removed or added, the chart is automatically refreshed, elements are placed according to the grid and links are redrawn.

Select

To select an item simply click on it with the left mouse-button. Multiple selections are made by clicking a blank area of the workspace then drag and drop until the required items are highlighted. Alternatively, multiple items are selected by holding down (**CTRL**) or (**SHIFT**) then clicking elements to add to the selection.



Rename

You can rename steps and transitions.

To rename elements

1. Select the element to rename.
2. From the Edit menu, choose **Rename Step (Transition)** OR right-click on the element to display the contextual menu.

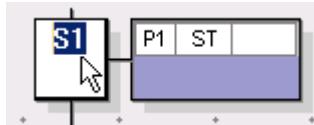
The Change Name dialog box appears:



3. Edit the name then click **OK**.

OR

1. Select the element to rename.
2. Click on the name to edit.
3. Edit the name directly in the program element:



4. When finished, click elsewhere in the workspace.

Move

To move elements

1. Select the element(s) to move.
2. Drag the 'ghost' to a valid location.
3. Drop the elements as required.

Cut

Use the cut command to remove selected elements and move them to the clipboard, replacing the clipboard's current contents.

To cut elements

- Select the element(s) to cut, then from the Edit menu, choose **Cut (CTRL+X)** or click  , on the Standard toolbar.

Copy

Use the copy command to copy selected elements and place them on the clipboard, replacing the clipboard's current contents.

To copy elements

- Select the element(s) to copy, then from the Edit menu, choose **Copy (CTRL+C)** or click  , on the Standard toolbar.

Paste

Use the **Paste** command to place the contents of the clipboard at the insertion point. Pasted elements are automatically assigned sequentially numbered names.

To paste elements

1. From the Edit menu, choose **Paste (CTRL+V)**.
2. Position the 'ghost'.
3. Click to paste at the new location

OR



1. On the Standard toolbar, click .
2. Position the 'ghost'.
3. Click to paste at the new location

Delete

To delete elements

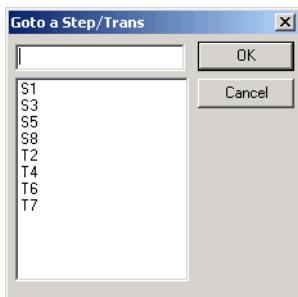
- ▶ Select the element(s), then from the Edit menu, choose **Delete** <Delete>.

Goto

The step / transition is selected in the level 1.

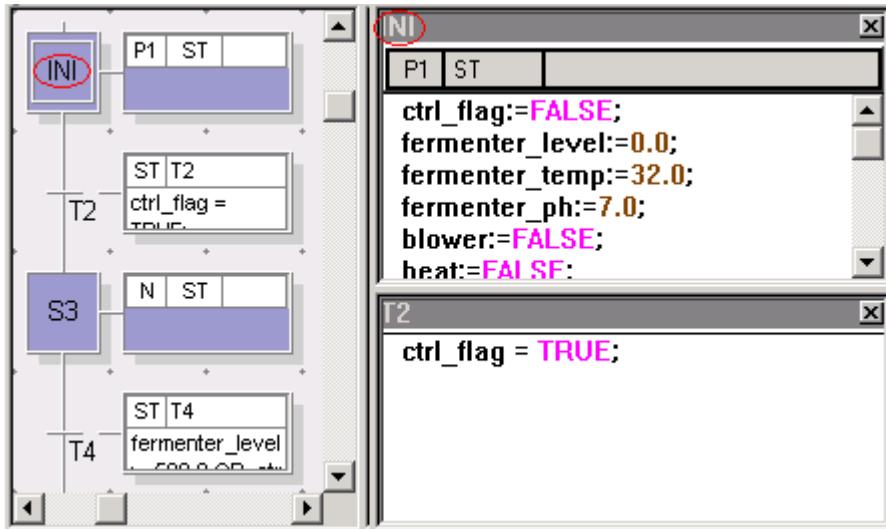
To go to a step or transition in the current SFC program

1. From the Edit menu, choose **Goto**.
2. In the Goto Step/Transition dialog, select the element from the list then click **OK**.



Level 2

The **Level 2** window displays the coding for steps and transitions. For steps, the window displays the defined action blocks. For transitions, the window displays the defined conditions. You can also display a second level 2 window for another step or transition. When first coding steps, you need to add action blocks.



To edit the Level 2

The Edit Level 2 option is available from the main menu and the contextual menu accessed by right-clicking a step or transition.

1. Select the step or transition.
2. From the Edit menu, choose **Edit Level 2**.
3. To display a second level 2 window for another step or transition, do one of the following:
 - Right-click an element (without selecting it), then choose **Edit Level 2 in separate window** from the contextual menu
 - Drag to select an element, then press Ctrl+Enter

Coding Action Blocks for Steps

You attach **action blocks** to steps by adding them in the level 2 window, then defining their name, comment (optional), type, and qualifier. Comments are displayed after the action block name in the level 2 window for a step, for example, InitAction (* initialize all *). When changing the type of an action block, the code zone must be empty. For details on actions within steps, see page 404.

You can specify the ST, IL, or LD language for use as default for the level 2 programming of steps.

The available action block types are the following:

- Boo
- IL
- LD
- SFC child
- ST

Boo action blocks require the selection of a Boolean variable from the variable selector. These action blocks take the name of the selected variable. SFC child action blocks require assigning the name of the SFC child to the action block. You cannot program Boo or SFC-child action blocks.

The available qualifiers for all action block types are None Store Action (N), Set (S), and Reset (R). The qualifiers for IL, LD, and ST action block types also include the Pulse on Deactivation Action (P0) and Pulse On Activation Action (P1).

To add action blocks to steps

You can add action blocks using the main menu, the SFC toolbar, or a contextual menu accessed by right-clicking in the level 2 window.

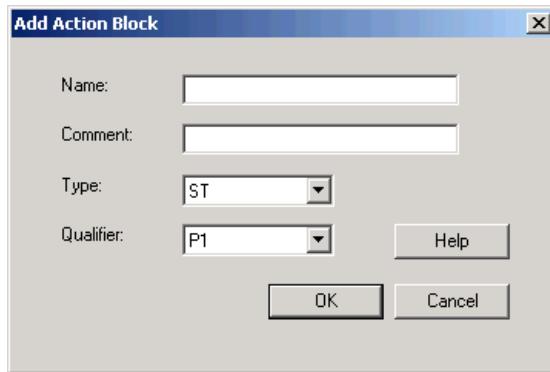
1. In the SFC chart, select the step for which to add an action block, then from the Edit menu choose **Edit Level 2**.

2. Click in the level 2 window, then from the Edit menu, choose **Add Action Block** or click



, from the SFC toolbar.

3. In the Add Action Block dialog, enter a name and comment (optional), then select a type and qualifier for the action block. For the Boo type, the selected variable's name is automatically entered in the Name field. For the SFC child type, enter the name of the SFC child in the Name field.



4. Click **OK**.
5. In the editor window, enter the code for the action block. You cannot program Boo and SFC child action blocks.

To specify the default programming language for steps

- From the Options menu, choose Set Default Level 2 Language, then Step, then the desired programming language.

Coding Conditions for Transitions

You attach conditions to transitions by programming these in the level 2 window. Only one condition can be attached to a transition. When defining conditions, you indicate a name, a comment (optional), and the programming language (type). The available programming languages for transitions are LD and ST. When changing the programming language of a transition, the code zone must be empty. For details on conditions attached to transitions, see page 410.

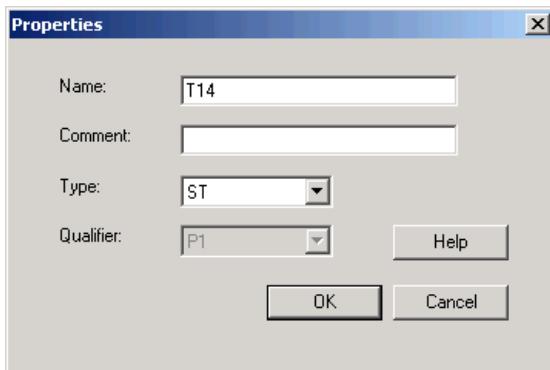
You can specify the ST or LD language for use as default for the level 2 programming of transitions.

To attach conditions to transitions

1. In the SFC chart, select the transition for which to attach a condition, then from the Edit menu choose **Edit Level 2**.

The level 2 window is displayed with the transition's name and set for programming in the ST language.

2. To change the name or programming language for the transition, double-click the level 2 window title bar, then in the Properties dialog, make the necessary changes and click **OK**.



To specify the default programming language for transitions

- From the Options menu, choose Set Default Level 2 Language, then Transition, then the desired programming language.

Moving Action Blocks Up or Down

You can change the order of the action blocks in a step. The displayed order is used during execution.

To move an action block up

1. Select the step in the SFC chart, then from the Edit menu, choose **Edit Level 2**.
2. In the level 2 window, click the action block to move up, then do one of the following:
 - Right-click the action block, then from the contextual menu, choose **Move Up**.
 - On the SFC toolbar, click  .

To move an action block down

1. Select the step in the SFC chart, then from the Edit menu, choose **Edit Level 2**.
2. In the level 2 window, click the action block to move up, then do one of the following:
 - Right-click the action block, then from the contextual menu, choose **Move Down**.
 - On the SFC toolbar, click  .

Deleting an Action Block

You delete action blocks from a step from within the level 2 window.

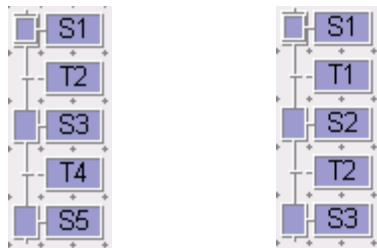
To delete an action block

1. Select the step in the SFC chart, then from the Edit menu, choose **Edit Level 2**.
2. In the level 2 window, click the action block to delete, then do one of the following:
 - From the Edit menu, choose **Delete Action Block**.
 - From the SFC toolbar, click  .

Renumbering Charts

Renumbering of SFC elements takes place from top to bottom, then from left to right. Renumbering is only applied to steps and transitions having the standard default names (Sx and Tx).

Before Renumbering After Renumbering



To renumber a chart

- From the Edit menu, choose **Renumber** or click  , on the SFC toolbar.

FC Editor



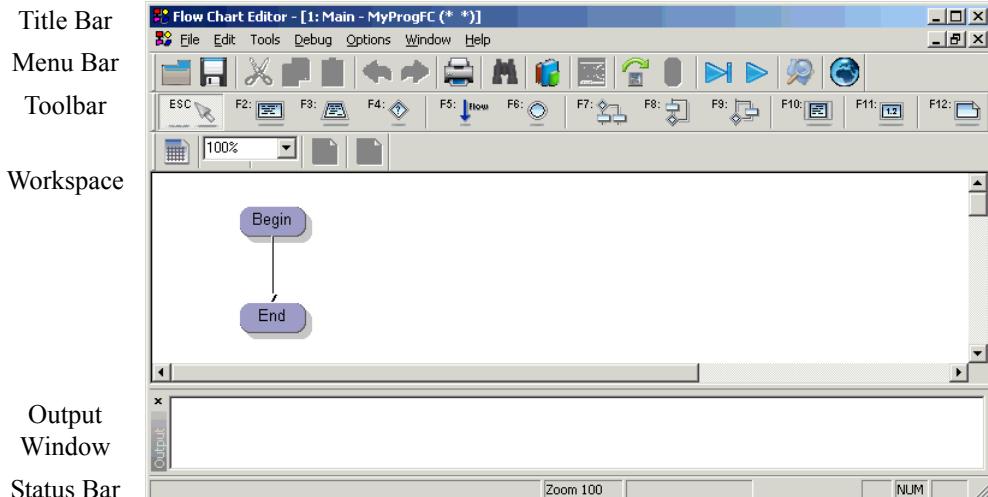
The **FC (Flow Chart) editor** is launched automatically when an FC program is edited from the Workbench.

Note: Before creating new programs, you need to close the Dictionary.

To subsequently open another program, from the FC editor

- From the File menu, choose **Open (CTRL+O)** or click  on the Standard toolbar.

Appearance



Menu Bar

Some options are available as keyboard commands.

| | | | |
|------|------------------------|--------|---|
| File | Open | Ctrl+O | opens an existing POU |
| | Close | Alt+F4 | closes the POU |
| | Save | Ctrl+S | saves the current POU |
| | Build Program | Alt+F3 | builds the code for the current POU |
| | Stop Build Program | | stops the build in progress for the current POU |
| | Dictionary | Ctrl+D | opens the dictionary filtered for the current POU |
| | Description | Ctrl+K | accesses the program description |
| | Print | Ctrl+P | prints the current POU |
| | Exit | Ctrl+Q | leaves the language editor |
| | Undo | Ctrl+Z | cancels the last action |
| Edit | Redo | Ctrl+Y | restores the last cancelled action |
| | Cut | Ctrl+X | removes the selected item and places it on clipboard |
| | Copy | Ctrl+C | takes a copy of the selected item and places it on the clipboard |
| | Paste | Ctrl+V | inserts the contents of the clipboard into the selected item |
| | Delete | DEL | removes the selected item |
| | Find / Replace in POUs | Ctrl+F | finds and replaces text in a project, a configuration, a resource, or a POU |
| | Go to | Ctrl+G | jumps to the indicated element number |
| | Renumber | | renumbers all elements in the chart in sequential order |
| | Edit Level 2 | Enter | opens the level 2 programming for an element |

| | | | | |
|---------|----------------------|---------------------------------|------------|--|
| | Edit | Edit Level 2 in Separate Window | Ctrl+Enter | opens the level 2 programming for an element in a separate window |
| | (Continued) | | | |
| | | Insert/Set Identifier | Ctrl+I | accesses the Select Variable dialog box where you can insert a variable in the current POU |
| | | Insert New Rung | Ctrl+R | inserts a rung |
| | | Change Yes/No Direction | | changes the direction of an IF-THEN-ELSE structure |
| Tools | Browser | | Ctrl+B | accesses the Cross References browser listing and localizing all instances of global variables and I/Os declared in a project |
| Debug | Debug | | Alt+F6 | switches the application to debug mode |
| | Simulation | | Alt+F7 | switches the application to simulation mode |
| | Debug FB | | F11 | opens a selected function block in the language editor with its instantiation values |
| Options | Set Level 2 Language | | | sets the programming language used for level 2 programming. Possible languages are LD, ST, and IL. |
| | Layout | | | accesses the Layout editor where you specify options such as the toolbars to display, the magnification of the workspace area, and other level 2 options |
| | Customize | | Ctrl+U | accesses the customization properties for Workbench views and editors |
| | Target/Code Settings | | | accesses the compilation options for the POU |

| | | | |
|--------|---------------------|--------|---|
| Window | Cascade | | sets the different views of the project to appear in a cascading manner |
| | Tile | | sets the different views of the project to appear in a tiled manner |
| | Split | | splits the workspace into two simultaneous views |
| | Show Output Window | Ctrl+4 | displays the output window below the workspace |
| | Clear Output Window | | clears the contents of the output window |
| Help | Contents | F1 | accesses the online help |
| | Search Help On... | | not currently supported |
| | About | | displays product and version information |
| | Support Info | | not currently supported |

Working with Flow Charts

Flow Chart programs are created in a resource in the link architecture view of the Workbench. After having opened the program, you can create / insert new elements in the Level 1 (diagram) or modify / move existing elements.

Every Flow Chart must have a BEGIN and an END, these are automatically inserted when a new Flow Chart is created from the link architecture view. These elements can be moved, but not deleted.

Level 2 programming of each action and test using ST, LD or IL syntax is also performed within the Flow Chart editor.

To save the current flow chart

- From the File menu, choose **Save (CTRL+S)** or click  , on the Standard toolbar.

From the editor, you can:

- Build the current program code to check your program and prepare the code for building the resource code.
- Print your program.
- Launch the Dictionary.

To include a description documenting your program

- From the File menu, choose **Description**.

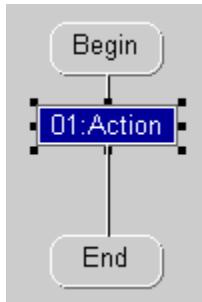
Flow Chart Elements

Most common operations are performed with the mouse: insertion, selection and drag and drop of elements. Moving an element also moves all the elements directly linked below. Elements are individually re-sizable.

Action



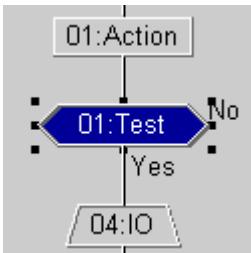
Insert FC **action** creates a new action each time the mouse button is pressed. Actions are automatically linked by the Flow Chart editor. An action number is automatically generated, sequentially for each new action. For details about FC actions, see page 420.



Test



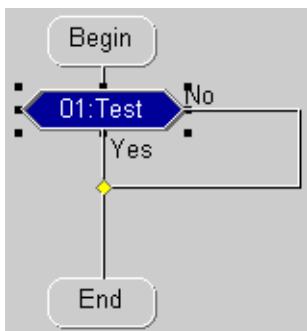
Insert a test to branch between sections of the program that are executed conditionally. Double-clicking the Yes or No text, swaps their position. For details about FC conditions, see page 420.



IF-THEN-ELSE

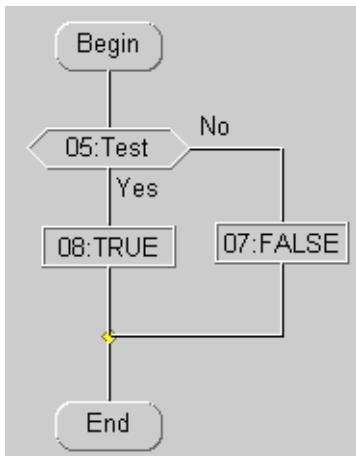


This generates a standard **IF-THEN-ELSE** structure in the Flow Chart. Examples of Flow Chart complex structures are available on page 425.



Actions can be added on both Branches before the Connection.

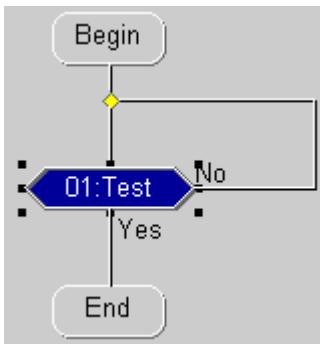
Example



DO-WHILE



This generates a standard **DO-WHILE** structure in the Flow Chart. Examples of Flow Chart complex structures are available on page 425.

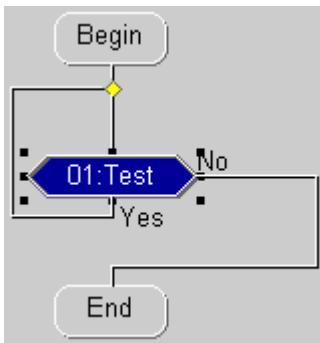


Note: The difference between this structure and the WHILE-DO is the location of the action(s) to repeat.

WHILE-DO



This generates a standard **WHILE-DO** structure in the Flow Chart. The difference between this structure and the DO-WHILE is the location of the action(s) to repeat. Examples of Flow Chart complex structures are available on page 425.



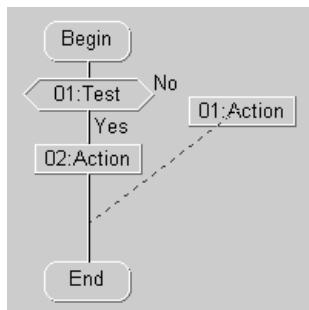
Flow



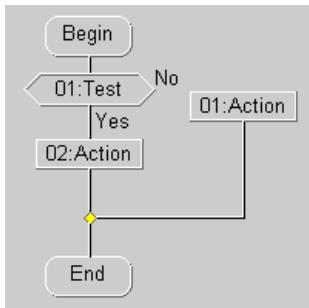
A **Flow** indicates a link between two elements. For details about FC flow links, see page 419.

To insert a flow

1. On the Flow Chart Tools toolbar, click .
2. Click on the elements to flow from.
3. Drag the link to a point on another link or a non-connected element.



4. Drop the Flow link.

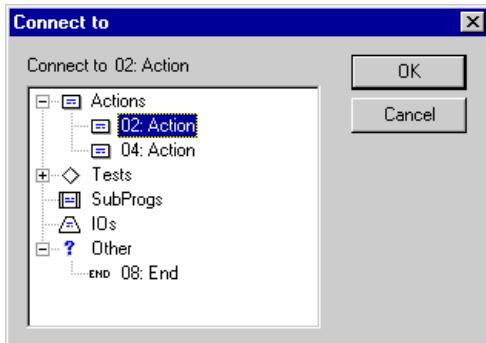


Connector



A **Connector** is used to link to an element, without specifically 'drawing' the link. For details about FC connectors, see page 424.

The Connect To Dialog Box is automatically displayed:

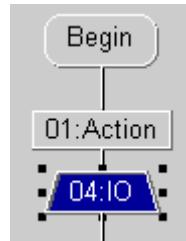


1. Expand (or collapse) sections in the tree by clicking on the **[+]** (or **[□]**) Buttons.
2. Select an element then click **OK**.

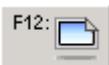
I/O Specific



An **I/O Specific action** is one that contains hardware dependent code, they must be re-written for different I/Os. For details about FC I/O specific actions, see page 423.



Comment



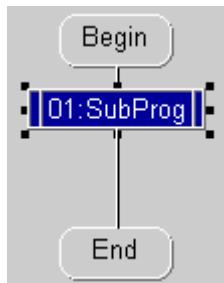
F12: Comments are free format text inserted anywhere in the Flow Chart for documentation purposes only. For details about FC comments, see page 424.



Sub-Program



F10: When inserting a Sub-program symbol, a dialog box is displayed to select the Sub-program from the list within the current program. For details about FC sub-programs, see page 422.



Note: Double-clicking on a **Sub-program** opens the selection dialog box to change the sub-program reference

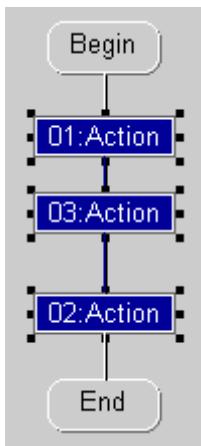
Managing Elements

Flow Chart elements can be cut, copied and pasted within a Flow Chart or, if more than one Flow Chart is open, between different Flow Charts. When an element is moved, removed or added, the chart is automatically refreshed, elements are placed according to the grid and links are redrawn.

Select



To **Select** an item, simply click on it with the left mouse-button. Multiple selections are made by clicking a blank area of the workspace then drag and drop until the required items are highlighted. The shift key, combined with a mouse click, selects multiple, distant, elements.



Cut

Use the cut command to remove selected elements and move them to the clipboard, replacing the clipboard's current contents.

To cut elements

1. Select the element(s) to cut.
2. From the Edit menu, choose **Cut (CTRL+X)**.

OR

1. Select the element(s) to cut.



2. On the Standard toolbar, click .

Copy

Use the **copy command** to copy selected elements and place them on the clipboard, replacing the clipboard's current contents.

To copy elements

1. Select the element(s) to copy.
2. From the Edit menu, choose **Copy (CTRL+C)**.

OR

1. Select the element(s) to copy.



2. On the Standard toolbar, click .

Paste

Use the **Paste** command to place the contents of the clipboard at the insertion point. Any existing selected items are automatically unselected.

To paste elements



1. From the Edit menu, choose **Paste (CTRL+V)** or click , on the Standard toolbar.
2. Position the 'ghost'.
3. Click to paste at the new location

Delete

To delete elements

1. Select the element(s).
2. From the Edit menu, choose **Delete** OR press <Delete>.

Move

All elements linked directly below a 'moved' element are automatically moved and their flow links re-drawn.

To move elements

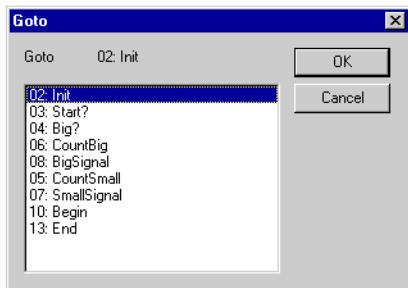
1. Select the element(s) to move.
2. Drag the 'ghost' to a valid location.
3. Drop the elements as required.

GoTo

To go to a symbol in the current FC program

1. From the Edit menu, choose **Goto**.

The Goto dialog box appears:



2. Select the element from the list then click **OK**.

The Action / Test Level 1 is selected.

Renumber

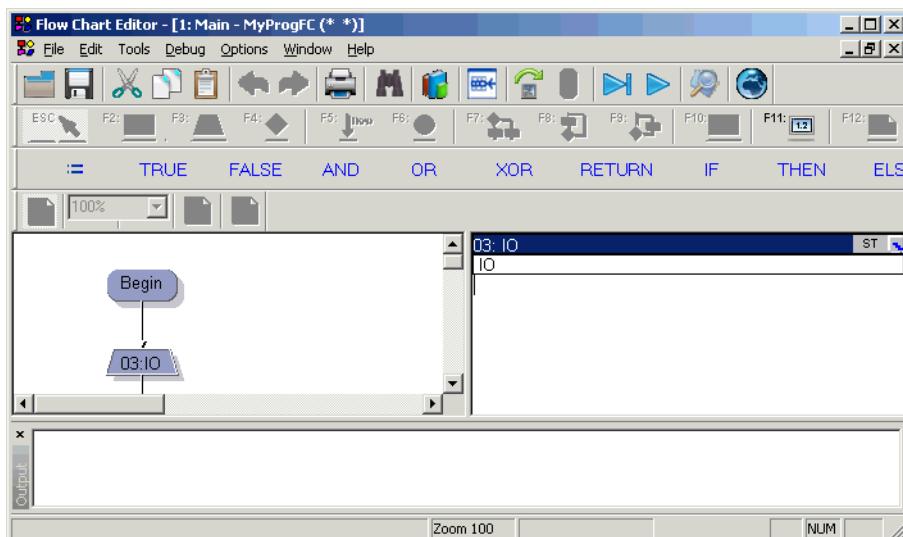


Two elements cannot have the same logical number within one Flow Chart. In this case, a renumber facility is provided to automatically generate sequential numbers. The order in which the chart is renumbered is based on each element's position, from top to bottom, then from left to right.

Level 2

To view the Level 2 window of an FC Element (action or test)

1. Select an FC element.
2. Do one of the following:
 - From the Edit menu, choose **Edit Level 2**.
 - Double-click an FC element.



Note: The FC Level 2 is also shown within the corresponding element representation in the Level 1 Workspace.

To view the Level 2 of another element, follow the instructions above, in which case the new Level 2 will replace the one displayed, or open the new Level 2 in a separate window.

To open the Level 2 in a separate window

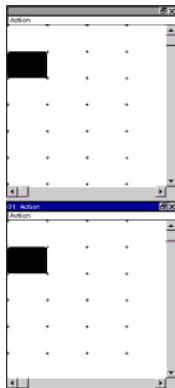
1. Select an FC element.
2. Do one of the following:
 - From the Edit menu, choose **Edit Level 2 in separate window**.
 - Press <**Ctrl+Return**>.

You can close level 2 windows by clicking on the close icon on the right of their title bar.

Level 2 Window

When the Edit Level 2 command is used:

- If no Level 2 window exists, a level 2 window is opened.
- If one level 2 window is already open, it is replaced by the level 2 of the current element.
(The Level 2 of the FC window is sub-divisible).
- If there are two level 2 windows, the level 2 window that had the focus is replaced by the level 2 of the currently selected element.



A maximum of two separate windows (elements) can be opened for simultaneous editing.

Edit the Level 2

ST is the default language of the level 2. You can change the language to LD or IL with the list displayed on the right of the title bar of the level 2 window.

You can set the default language for the level 2 programming from the menu by choosing Options, then Set Level 2 Language, then the desired default language.

- In a test only one condition can be written; In the case of editing in LD, there is only one coil without any variable attached. The coil value corresponds to the value of the test.
- In a test, no pulse is permitted, i.e. neither positive, nor negative contacts can be used.

Related Topics

Multi-language Editor

Multi-language Editor

 The **Multi-language editor** has editing functions for graphical and textual languages. These editing functions are automatically launched when an FBD, ST, IL, or LD program is opened from the Workbench.

The editor only allows new elements to be inserted if the current position is valid. Use the mouse or cursor keys to move the current position around within the Workspace.

From the editor, you can perform several tasks:

- Build the current Program code (to check your Program and prepare the code for building the Resource code)
- Print programs
- Launch the Dictionary

Note: Before creating new programs, you need to close the Dictionary.

When printing programs, the fonts used in the diagram are the same as for the editor. The FBD and LD diagrams are scaled to fit the width of the printed page format (portrait or landscape). To adjust the font for printed diagrams, you need to modify the font used for the editor.

To subsequently open another program, from the Multi-language editor

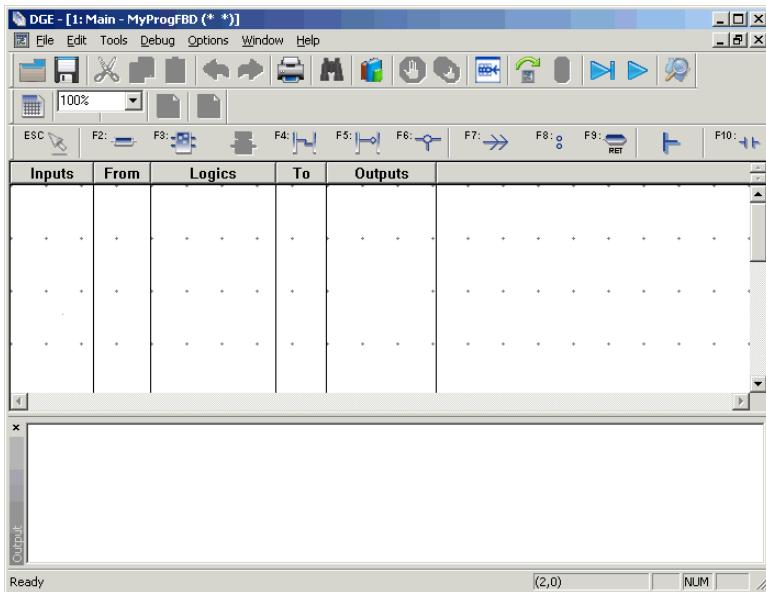
- From the File menu, choose **Open (CTRL+O)**.

To add a description to a program

- From the File menu, choose **Description**.

Appearance

Title Bar



Menu Bar

Toolbars

Workspace
with or
without
Guidelines

Output
Window

Status Bar

Note: The Language toolbar contains tools for LD, ST, IL, or FBD.

Workspace

You can arrange the workspace of the FBD editor to show guideline areas. These areas divide the workspace into logical sections: Inputs, From, Logics, To, and Outputs. Elements move independently of the area guidelines. You can choose to hide individual areas and resize the areas. You can also choose to restore the default area sizes.

When moving the cursor across the FBD or LD editor, the cursor's coordinates are displayed in the status bar. These coordinates refer to grid areas. For instance, the top-leftmost grid area is coordinate (0,0) and the grid area to its immediate right is coordinate (1,0). The grid coordinates remain the same whether the zoom or cell width changes.

To manage the guidelines

You access the Areas layout options window from the menu or by right-clicking an area titlebar.

1. To show or hide the guideline areas, from the Tools menu, choose **Show/Hide Areas**, then in the areas layout window, check the areas to display.
2. To resize an area, drag the boundary on the left or right side of the heading until the area is the width you want.
3. To return the area guidelines to their initial widths, click **Restore Default Area Sizes**.

Menu Bar

The options available differ depending on the POU's programming language. Some options are available as keyboard commands.

| | | | |
|------|-----------------------|--------|---|
| File | Open | Ctrl+O | opens an existing POU |
| | Close | Alt+F4 | closes the POU |
| | Save | Ctrl+S | saves the current POU |
| | Build Program | Alt+F3 | builds the code for the current POU |
| | Stop Build Program | | stops to build in progress for the current POU |
| | Dictionary | Ctrl+D | opens the dictionary filtered for the current POU |
| | Description | Ctrl+K | accesses the program description |
| | Print | Ctrl+P | prints the current POU |
| | Exit | Ctrl+Q | leaves the language editor |
| Edit | Undo | Ctrl+Z | cancels the last action |
| | Redo | Ctrl+Y | restores the last cancelled action |
| | Cut | Ctrl+X | removes the selected item and places it on clipboard |
| | Copy | Ctrl+C | takes a copy of the selected item and places it on the clipboard |
| | Paste | Ctrl+V | inserts the contents of the clipboard into the selected item |
| | Special Paste | | places the contents of the clipboard in a specified position |
| | Delete | DEL | removes the selected item |
| | Select All | Ctrl+A | selects all items in the active view |
| | Find / Replace in POU | Ctrl+F | finds and replaces text in a project, a configuration, a resource, or a POU |
| | Find Matching Name | Alt+F2 | finds and selects matching variable names in the current POU |

| | | | | |
|-------|---------------------------|--------------------------|--------|---|
| | Edit (Continued) | Find Matching Coil | Alt+F5 | finds and selects matching variable names for coils in the current POU |
| | | Go to Line | Ctrl+G | jumps to the indicated line number |
| | | Insert/Set Identifier | Ctrl+I | accesses the Select Variable dialog box where you can insert a variable in the current POU |
| | | Insert/Set Block | Ctrl+R | accesses the list of all available functions and function blocks to insert in the current POU |
| | | Insert Rung | Ctrl+R | inserts a rung |
| | | Change Coil/Contact Type | Space | changes the selected coil or contact type |
| | | Insert Comment | | inserts a comment above a rung in LD diagrams |
| Tools | Browser | | Ctrl+B | accesses the Cross References browser listing and localizing all instances of global variables and I/Os declared in a project |
| | Show/Hide Execution Order | | Ctrl+W | shows or hides the execution order of FBD diagrams |
| | Show/Hide Areas | | | accesses the areas layout window where you check the areas to display in the FBD editor workspace |
| | Show/Hide Output Values | | | shows or hides the output values of blocks (operators, functions, and function blocks) in the FBD and LD editors, while in debug or simulation mode |
| Debug | Debug | | Alt+F6 | switches the application to debug mode |
| | Simulation | | Alt+F7 | switches the application to simulation mode |
| | Spy Selection | | | adds a selected variable to the Spy List while in Debug mode |

| | | | |
|----------------------|----------------------|---------|--|
| Debug (Continued) | Debug FB | F11 | opens a selected function block in the language editor with its instantiation values |
| | Toggle Breakpoint | F10 | sets or removes a breakpoint for step-by-step mode |
| | Breakpoints | | removes a breakpoint for step-by-step mode |
| | Real Time | | switches the application to real-time mode |
| | Cycle to Cycle | | switches the application to cycle-to-cycle mode |
| | Execute One Cycle | Alt+F10 | executes one cycle |
| | Step | Alt+F8 | executes the current line then steps to the next line |
| | Step Into | Alt+F9 | executes the current line then steps into the next line |
| | Show Current Step | | shows the current step |
| | Layout | | accesses the Layout editor where you specify which toolbars to display and the magnification of the workspace area |
| Options | Customize | Ctrl+U | accesses the customization properties for Workbench views and editors as well as working preferences |
| | Tab Setting | | sets the number of spaces for the Tab character |
| | Show Coils/Contacts | | sets the display of the name, alias, or name and alias for coils and contacts |
| | Target/Code Settings | | accesses the compilation options for the POU |
| | Auto Input | | assigns a variable name or block when inserting elements |

| | | | |
|-------------|---------------------------------|--|--|
| Options | Manual Input | | |
| (Continued) | | | |
| | Numerical Display | assigns a variable name or block at any time | |
| | Show I/O Variable Comments | sets the numerical display of values in the FBD editor, displays comments for I/O variables, entered in the dictionary | |
| | Hide I/O variable Comments | in the FBD editor, hides comments for I/O variables | |
| | Show Internal Variable Comments | displays comments for I/O variables, entered in the dictionary | |
| | Hide Internal Variable Comments | hides comments for I/O variables | |
| Window | Cascade | sets the different views of the project to appear in a cascading manner | |
| | Tile | sets the different views of the project to appear in a tiled manner | |
| | Show Spy List | accesses the Spy List window where you specify variables whose values are displayed while in test mode | |
| | Show Output Window | Ctrl+4 | displays the output window below the workspace |
| | Clear Output Window | | clears the contents of the output window |
| | Show Call Stack | | displays the call stack window |
| Help | Contents | F1 | accesses the online help |
| | Search Help On... | | not currently supported |
| | About | | displays product and version information |
| | Support Info | | not currently supported |

Multi-Language Elements

The language used for the Program currently edited determines the elements that can be inserted. This is reflected in the menu commands and toolbar buttons.

- ST/IL Elements
- LD Elements
- FBD Elements

Note: Arrays must be declared in the Dictionary View before inserting them in Functional Block Diagrams (FBD).

ST/IL Elements

The main keywords of the ST or IL language are available in the Language toolbar. When entering ST or IL syntax, basic coding is black while other items are displayed using color:

- Keywords are pink
- Numbers are brown
- Comments are green

Inserting a variable name can be done directly by typing it or by using the Insert Identifier command from the Edit menu. To insert block instances or to get help on a block, use the Insert Block command from the Edit menu.

LD Elements

When editing an LD POU, you can place elements by using the keyboard. Keyboard shortcuts are indicated on the LD toolbar. Alternatively, use the mouse to select the element to insert from the toolbar. The element is inserted at the current position in the diagram.

The current position is the cell that is marked in black.

To attach a variable to a coil or a contact, double-click on it or press <Return> when it is selected. The Select Variable dialog box is displayed. You can also use the Insert identifier

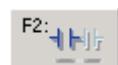
button  on the Standard toolbar.

To attach a block type to a block, double-click on it or press <Return> when it is selected.

The Select Block dialog box is displayed. You can also use the Insert Identifier button  on the Standard toolbar.

If you want to enter a variable name or block type when you place the element, check "Auto input" in the Option menu. If you want to do it at a later time, uncheck this option.

Contact on the Left



The contact is inserted to the left of the current position (highlighted in black).

Note: Pressing F2 on the keyboard has the same effect.

Contact on the Right



The contact is inserted to the right of the current position (highlighted in black).

Note: Pressing F3 on the keyboard has the same effect.

Parallel Contact



Inserts a contact, parallel to the current selection.

Note: Pressing F4 on the keyboard has the same effect.

Coil



Inserts a coil on the current rung.

Note: Pressing F5 on the keyboard has the same effect.

Block on the Left



The block is inserted to the left of the current position (highlighted in black).

Note: Pressing F6 on the keyboard has the same effect.

Block on the Right



The block is inserted to the right of the current position (highlighted in black).

Note: Pressing F7 on the keyboard has the same effect.

Parallel Block



Inserts a block, parallel to the current selection.

Note: Pressing F8 on the keyboard has the same effect.

Jump



Inserts a jump to a label.

Note: Pressing F9 on the keyboard has the same effect.

Label

In Ladder, the label identifies a rung.

To enter a label identifying a rung

1. Press Enter or double-click the header-cell of the rung
2. In the dialog box, enter a name for the label.
3. Press OK to confirm.

Return



Inserts a return symbol.

Note: Pressing Shift+F9 on the keyboard has the same effect.

Change Coil/Contact Type



The available types of coils and contacts are listed in the Language Reference.

For LD elements in FBD diagrams, you can also change the type of contact or coil.

To change the type of a coil or a contact

1. Select the coil or contact.
2. Do one of the following:
 - From the Edit menu, choose Change coil/contact type.
 - On the LD toolbar, click .
 - Press the <space bar>.

Insert New Rung

To insert a rung between two existing rungs

- From the Edit menu, choose **Insert New Rung**.

The new rung is inserted above the rung that contains at least one selected element. The rung is composed of one contact and one coil.

When you press any button on the LD toolbar at the end of the diagram, a new rung is created.

Other Operations

To insert a link

1. Select the desired part of the rung by clicking on it.

2. On the right hand side of the LD toolbar, click .

The new link is inserted to the left of the position highlighted in black.

To align coils on all rungs



"justifies" the coils on each rung so that coils are aligned vertically on the right.

FBD Elements

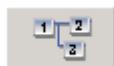
When programming in FBD, choose the element to be inserted from the FBD toolbar and place it in the Program Workspace.

Place all elements (blocks and variables), then link them by using links. An element is automatically linked to another element if it is placed next to it such that their connectors meet. When wiring intersects or diverges, the junction is indicated by a dot.

Before using arrays in FBD, these must be declared in the Dictionary View. Ladder elements are also available for use in FBD programs.

To show the order of execution of an FBD program

You can show the order of execution in the form of numerical tags for the following elements in an FBD program: coils, returns, jumps, instances of function blocks (declared or not), and variables where a value is assigned in the program. When the order cannot be determined, the tags display question marks (?). You can perform this task from the menu bar, the toolbar, or keyboard shortcut (Ctrl+W).

- ▶ From the Tools menu, choose **Show/Hide Execution Order** or click  on the Standard toolbar.

To assign a name to a variable or block graphic symbol

- ▶ Select the graphic symbol, then on the Standard toolbar, click  or select the graphic symbol, then double-click it.

To assign variable names or block types when placing an element

- ▶ In the Options menu, choose **Auto Input**.

To assign variable names or block types at any time

- ▶ In the Options menu, choose **Manual Input**.

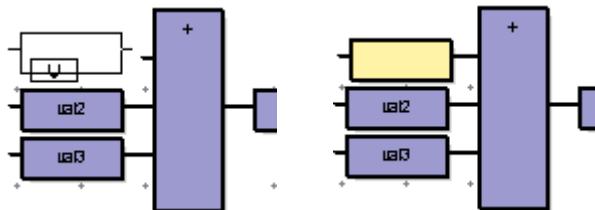
Variable



Accesses the variable selector enabling the insertion of a variable or constant into the workspace.

To connect a new symbol to an existing one (another variable, a block input, or a block output), keep the mouse button depressed (the cursor becomes a "ghost" symbol) and drag the element until its connecting line on the left (or right) overlaps an existing connecting point. When the mouse is released, the new symbol is automatically created and linked.

Drag to place the element: Release the mouse button. The new variable is automatically connected:



For input and output variables, you can choose to display comments entered in the dictionary directly below the variable by choosing **Show I/O Variable Comments** from the Options menu. You hide comments by choosing **Hide I/O Variable Comments**. When moving the cursor over a selected variable, its data type and hidden comment is displayed as a tooltip.



When entering variable blocks, you can choose to have the Workbench automatically prompt you to enter a constant or select a variable from the Select Variable dialog by choosing **Auto Input** from the Options menu. You can also choose to enter variable names manually by choosing **Manual Input**.

You can resize variable blocks.

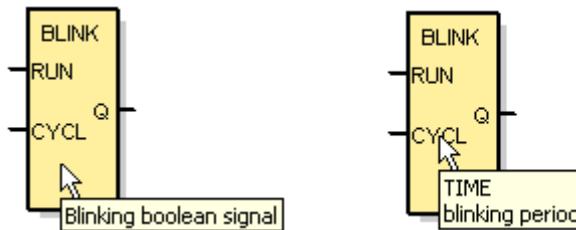
Function Block



Accesses the block selector enabling the insertion of a block into the workspace. Blocks can be function blocks ("RAS device" or IEC 61131), functions ("RAS device" or IEC 61131) or operators.

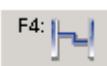
Inputs and outputs can be connected to variables, contacts or coils, or other block inputs or outputs. Formal parameter short names are displayed inside the block.

When moving the cursor over a selected function block or instance of a function block, its comment is displayed as a tooltip. Furthermore, when moving over a parameter, its data type and comment is displayed as a tooltip.

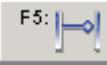


You can resize function blocks.

Link



Connection links are drawn between elements in the diagram.



Negation connection links are equivalent to placing a NOT block on a direct link.

Both direct links and negated links are always drawn from an output to an input point (in the direction of the data flow).

Corner



User-defined points may be inserted in the diagram that determine the routing of links. First, place a corner, then add links to and from this point. If no corner is placed, the editor uses a default routing algorithm.

Jump

F7: 

Inserts a jump in the workspace.

A dialog box containing a list of labels to jump to is displayed. Alternatively, by entering a new name in the edit box, then clicking OK, a Jump is created to a new Label (the corresponding Label symbol must then be placed in the diagram).

A Jump symbol must be linked to a Boolean point. When this Boolean (left) connection is TRUE, the execution of the diagram Jumps directly to the target Label.

Note: Backward jumps may lead to a blocking of the PLC loop in some cases.

Label

F8: 

Inserts a label in the workspace.

The Jump Label dialog box is displayed to enter and create a Label name. Alternatively, if a Jump symbol was previously inserted, and a new Label name was entered in the edit box, the Label name specified when creating that Jump can be chosen from the list.

Labels can be placed anywhere in an FBD diagram. They are used as a target for Jump instructions, to change the execution order of the diagram. Labels are not connected to other elements.

Note: It is highly recommended to place Labels on the left of the diagram in order to increase diagram readability.

For more details about labels, see page 431.

Return



Inserts a return symbol in the workspace.

If the connection line (to the left of the Return symbol) has the Boolean state TRUE, the Program ends - no further part of the diagram is executed.

No connection can be put on the right of a RETURN symbol.

For more details about return statements, see page 431.

LD Elements

The LD elements available for use in Function Block Diagrams are the following:

Left Power Bar



Contacts of the Ladder Diagram language must be connected, on the left, to a left power bar which represents the initial "TRUE" state. The editor also allows the connection of any Boolean symbol to a left power bar.

You can resize the height of a left power bar.

Contacts



A contact can be connected, on the left, to a left power bar or another contact. A contact can be connected, on the right, to any other Boolean point; another contact, a coil, a Boolean input of a block...

By default, a direct contact is inserted. To change the contact type, select the contact and press the <spacebar>. Repeatedly pressing the <spacebar> cycles between all contact types.

LD Vertical "OR" Connection



LD vertical connection accepts several connections on the left and several connections on the right. Each connection on the right is equal to the OR combination of the connections on the left.

You can resize the height of an OR Connection.

Coils



A coil represents an Action. It must be connected on the left to a Boolean symbol, such as a contact or the Boolean output of a block. By default, a coil is inserted with a small Right Power Bar. If a link is inserted, from the right of a coil to a Right Power Bar, this small Bar is removed.

Before linking to a Right Power Bar:



After linking to a Right Power Bar:



By default, a direct coil is inserted. To change the coil type, select the coil and press the <spacebar>. Repeatedly pressing the <spacebar> cycles between the all coil types.

Right Power Bar



Coils may be connected, on the right, to a right power bar. This is optional. If a coil is not connected on the right, a small right power bar is included. For details about multiple connections, see page 437. For details about basic LD contacts and coils, see page 439.

You can resize the height of a right power bar.

Comment



Press this button, on the Language toolbar, then click in the Workspace to insert a comment. **Comments** are free format text inserted anywhere in the FBD POU, for documentation purposes only.

After entering text, click elsewhere in the Workspace to 'validate' the comment.

You can also resize comments.

Managing Elements

Programming elements can be cut, copied and pasted within a program or, if more than one is open, between different programs. When an element is moved, removed or added, the chart is automatically refreshed, elements are placed according to the grid and links are redrawn.

Select



Selections can contain text, graphics or both.

To make a selection

- Click the cursor on an element to make / change a selection.

To select multiple elements in LD or ST or IL

- Drag the cursor to highlight multiple elements in the workspace.

OR

- Hold down **SHIFT** then use the cursor keys to extend the current selection.

To select multiple elements in FBD

- Click in a blank area of the workspace then drag to enclose the required elements.

OR

- Hold down **CTRL** or **SHIFT**, then use the mouse, to add to the current selection.

Note: In the FBD editor, **ESC** removes the current selection. If the editor is in 'element insertion' mode, **ESC** returns to 'select' mode.

Resize

The dimensions of individual programming elements can be changed. Resizing elements in the Multi-language editor is only valid for FBD POU's.

To resize an element

1. Select the element to resize.
2. Click and hold the cursor over on the edge of the selected element.
3. Drag the edge to the desired position.

The cursor changes during a resize.

4. Release the mouse button to complete the operation.

Note: Elements cannot be resized so that they overlap other elements, you may need to move elements prior to resizing.

Undo/Redo

The Multi-language editor provides a multi-level Undo / Redo facility (*limited to only one action for ST and IL*).

To Undo (Redo) the previous action

- From the Edit menu, choose **Undo (Redo)** or click  or 

274

Move

Moving elements in the Multi-language editor is only valid for FBD IEC 61131 POU. You can drag individual elements to a new position in the workspace without first selecting them. However, to drag multiple elements, you need to select each element.

To move elements

- To move a single element, click the element and while holding down the mouse drag the element to its new position, then release the mouse.
- To move multiple elements, select all elements and while holding down the mouse drag the elements to their new position, then release the mouse.

Cut

Use the Cut command to remove selected Elements and move them to the clipboard, replacing the clipboard's current contents.

To cut elements

1. Select the element(s) to cut.
2. From the Edit menu, choose **Cut (CTRL+X)**.

OR

1. Select the element(s) to cut.



2. On the Standard toolbar, click .

Copy

Use the **Copy** command to copy selected elements and place them on the clipboard, replacing the clipboard's current contents.

To copy elements

1. Select the element(s) to copy.
2. From the Edit menu, choose **Copy (CTRL+C)**.

OR

1. Select the element(s) to copy.



2. On the Standard toolbar, click .

Paste

Use the **Paste** command to place the contents of the clipboard at the insertion point.

- For ST and IL, if text is selected before a paste, it is replaced by the contents of the clipboard.
- For LD, the elements on the clipboard are pasted in parallel with selected elements. To paste before or after a selection, use the Paste Special command. The Paste command may fail when placing a coil in parallel with a contact or a contact in parallel with a coil
- For FBD, using the ghost (keeping the mouse button depressed) enables moving pasted elements to the desired position.

To paste elements

- From the Edit menu, choose **Paste (CTRL+V)** or click , on the Standard toolbar.



Paste Special

This command is only valid for LD POUs. The Paste Special command places the contents of the clipboard in a specified position.

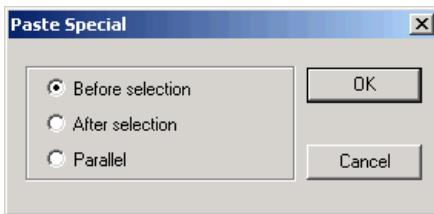
Notes:

- The standard Paste command has the same effect as a Parallel Paste Special command.
- The Paste command may fail because: *a coil can not be put in parallel with a contact, or a contact can not be put in parallel with a coil.*

To paste elements

1. From the Edit menu, choose Paste Special.

A dialog box appears, to choose the paste location:



2. Choose the desired paste location.
3. Click OK.

Delete

To delete elements

1. Select the element(s).
2. From the Edit menu, choose Delete.

Select All

All the elements in the current Program are simultaneously selected with the Select All command.

To Select All elements

- From the Edit menu, choose **Select All (CTRL+A)**

Find Matching Name

For LD only. Find Matching Name finds and selects matched variable names within the current POU. You can also find matching names for function blocks or rung labels.

To find a matching variable name

- Select a variable with the name to match.
- To select the next element with the same variable name as the current selection, press **<ALT+F2>**.

Find Matching Coil

For LD only. Find Matching Coil finds and selects matched variable names within the current POU. This feature is mainly used while in debug mode, to quickly find rungs forcing suspicious variables.

To find a matching coil

- Select a variable name to match.
- To select the next coil with the same variable name as the current selection, press **<ALT+F5>**.

Go to Line

The Go To Line command is only valid when editing ST and IL POUs. In the Multi-language editor, you access it from the File menu by choosing Go To Line.

You enter a line number, for the current POU, indicating the line to which to move the cursor.

Display/Hide Comments

You specify displaying or hiding variable comments at the language editor level. However, you can also choose to display or hide individual variable comments. For instance, you can choose to display all internal variable comments, then hide the comment for a specific internal variable. You can also choose to hide all I/O variable comments, then display the comment for a specific I/O variable. The display/hide setting for individual comments overrides the display/hide setting at the editor level (for either I/O variable or internal variable comments).

To display/hide comments defined for I/O variables

You display or hide all comments defined for I/O variables at the editor level from the main menu.

- ▶ To display comments for I/O variables, from the Options menu, choose **Show I/O Variable Comments**.
- ▶ To hide comments for I/O variables, from the Options menu, choose **Hide I/O Variable Comments**.

To display/hide comments defined for internal variables

You display or hide all comments defined for internal variables at the editor level from the main menu.

- ▶ To display comments for internal variables, from the Options menu, choose **Show Internal Variable Comments**.
- ▶ To hide comments for internal variables, from the Options menu, choose **Hide Internal Variable Comments**.

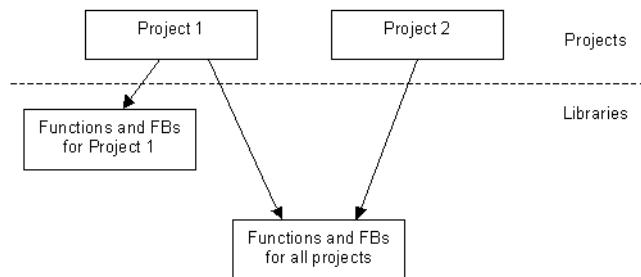
To display/hide individual comments defined for variables

You display or hide individual comments defined for I/O variables and internal variables at the comment level from a contextual menu. You can also reset individual variable comments to use the display/hide setting defined for a given type of variable comments.

- ▶ To display the variable comment, select the comment, then right-click and choose **Show Comment**.
- ▶ To hide the variable comment, select the comment, then right-click and choose **Hide Comment**.
- ▶ To set the variable comment to use the display/hide setting specified for the variable type, select the comment, then right-click and choose **Reset Default**.

Libraries

Libraries are special projects made up of configurations and resources in which you define functions and function blocks for reuse throughout **DS800 Development Suite** projects. Libraries also enable you to modularize projects and to isolate functions and function blocks so that these can be validated separately.



Functions and function blocks can be written using the IEC 61131 languages (FBD, LD, ST, or IL). Libraries can also contain POU's, global variable definitions, and any other item used for testing functions and function blocks.

Before using libraries, you need to create them.

Creating Libraries

You create libraries much the same as you create projects. You base a library on a template then develop its elements, i.e., configurations, resources, programs, functions, and function blocks. Libraries are stored in the same location as projects and are also MS-Access database (.MDB) files:

```
<root directory>/prj/<library name>/PRJlibrary.MDB
```

Three templates are available for use with libraries:

- LibSingleROC800 and LibSingleFB107, containing one resource in one configuration
- LibMultipleROC800, containing two resources in two different configurations linked by an Ethernet network

Furthermore, the target type of a library resource affects the usability of functions and function blocks throughout projects using the library. Functions and function blocks can only be used in resources referring to the same target type except when they use the SIMULATOR target type. When library resources use the SIMULATOR target type, all of their functions and function blocks can be used in any project resource regardless of its target type. Below are examples of possible combinations of resource target types:

| Target in library | Target in project | Usage |
|-------------------|-------------------|-------|
| SIMULATOR | ROC809 | OK |
| ROC809 | ROC809 | OK |

Library functions and function blocks must have unique names. When they have the same names as those defined in a project in which they are used, only those from the project will be recognized. Furthermore, you do not need to compile functions and function blocks in the library before using them in projects. They are compiled in the calling project space, in order to take care of the compiling options defined for the project.

To create a library

1. From the File menu, choose **New Project\Library**.
2. Enter a name for the library.
3. Select a template.
4. Click **OK**.

Using Libraries in a Project

Projects can use functions and function blocks from one or more libraries. You need to create libraries before using them. Furthermore, you need to define a project's dependencies, i.e., the set of libraries the project will use, before using a library's defined elements. A project can depend on more than one library.

You can also add dependencies to third-party library projects. However, to enable their use, you need to license third-party library projects. Otherwise, their dependency appears invalid.

Library functions and function blocks can refer to some global defined words or data types defined in the library. In such a case, these defined words and data types from the library can also be used in the project.

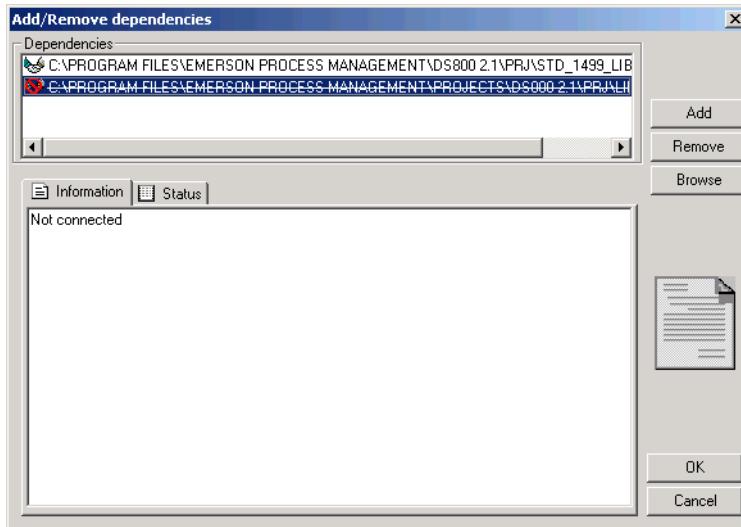
A library cannot use functions and function blocks from another library. In other words, you cannot define external dependencies for a library. However, a function or function block from a library can call other functions or function blocks from the same library. Furthermore, functions or function blocks from libraries can call 'C' written functions and function blocks defined for the corresponding target.

All functions and function blocks within a project, including those coming from libraries, must have unique names. When more than one uses the same name, the following conditions apply:

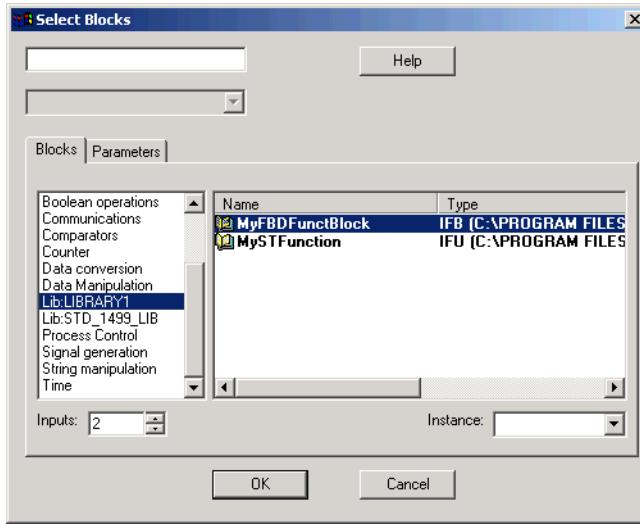
- If the functions or function blocks come from different libraries, warnings are generated at compilation and only the first definition is recognized
- If one function or function block is defined in the project and the other from a library, only the one defined in the project is recognized. The other is ignored.

Furthermore, when the same name is used for several types or several defined words having different definitions in a project and attached libraries, an error is generated at compilation time. However, when a data type or defined word is defined several times with the same contents or definition, a warning is reported but the project can be compiled.

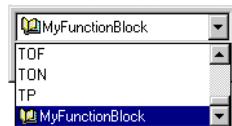
You specify a project's dependencies in the Add/Remove Dependencies window. This window is divided into the Dependencies list and the Information and Status areas. The Dependencies list displays the full pathnames of all libraries used by the project. The Information area shows the description of the library currently selected in the Dependencies list. The Status area shows the status of the library.



Once a project's dependencies are set up, you can access the functions and function blocks contained in the specified libraries from the Select Blocks dialog box, available in the ST, LD, and FBD editors. In this dialog box, library items appear with the IFB or IFU types and the source library's path. The IFB type indicates an IEC Function Block and the IFU type indicates an IEC Function.



In the dictionary, when declaring an instance of a function block from a library, the pathname of the library is also displayed together with the function block's type:



When a project has dependencies, an icon indicating the status of its dependencies appears at the bottom right-hand corner of both the hardware and link architecture views:



The status of the project's dependencies is OK



The project dependencies refer to an invalid library. This can happen if a library has been removed, renamed or moved.

When a project has a dependency on an invalid library, to retain all associations between the project and library upon renaming or moving, you need to reestablish the library path. Upon deleting a library, all associations are broken.

To define a project's dependencies

You can access the Add/Remove Dependencies window from the menu or the main toolbar. You can only define the dependencies for the currently opened project.

1. From the Tools menu, choose **Add/Remove Dependencies**.

The Add/Remove Dependencies window appears.

2. To add a new library to the list of dependencies:

- a) Click **Add**.
 - b) In the file browser, locate the library's PRJlibrary.MDB file.
 - c) Click **Open**.

3. To remove a library from the list of dependencies:

- a) From the Dependencies list, select the library to remove.
 - b) Click **Remove**.

To reestablish an invalid library path

You reestablish an invalid library path for a renamed or moved library from the Add/Remove Dependencies window. Reestablishing a library path restores all associations between a project and library.

1. From the Tools menu, choose **Add/Remove Dependencies**.

The Add/Remove Dependencies window appears.

2. From the Dependencies list, select the invalid library to reestablish.
3. Click **Browse**, then locate and select the library.

To license a third-party library project

You can choose to license third-party library projects while adding them as dependencies or at any other time. You initiate licensing for these library projects in the Add/Remove Dependencies window, then complete the process in the License Manager.

1. Make sure the third-party library project is copied onto your disk, then from the Tools menu, choose **Add/Remove Dependencies**.
2. In the Add/Remove Dependencies window, add the third-party library project to the list of dependencies:

- a) Click **Add**.
- b) In the file browser, locate the library's PRJlibrary.MDB file.
- c) Click **Open**.

A message stating that the library is not licensed is displayed.

- d) Click **OK**.

The License Manager is displayed

3. Do one of the following:

- To license the third-party library project at this time, click **Send** in the License Manager, then include all required information and send the email.
- To license the third-party library project at a later time, click **Cancel** in the License Manager. You can launch the licensing process at any time by selecting the unlicensed third-party library project, then clicking **Browse** to locate, select, and open the library's PRJlibrary.MDB file.

The original setup code and user codes as well as a Registration Key 1 and Registration Key 2 will be returned via e-mail.

4. Upon reception, make sure the setup and user codes are the same as those in the License Manager window, then copy and paste the registration keys in their respective fields.

The third-party library project is enabled for use.

Debug

When developing an application, you can choose to debug, i.e., detect and remove errors, from a project using one of two methods:

- Simulation mode. In this case, inputs and outputs are not managed by the target virtual machine. The rest (i.e. Binding exchanges and execution of the POU of each resource) is executed by the standard Windows platform. Each resource will be executed by one virtual machine on the PC running the Workbench.
- Online mode. In this case, each resource is executed by one **DS800 Development Suite** virtual machine on the real platform. A download operation is required, to download the code of each resource to the corresponding platform. For details on downloads, see page 293.

Note: To enable the debugging of a Project, you must first build it using the Build Project command. For details on building projects and resources, see page 347.

Before simulating a resource, the code for the simulation has to be generated for each resource. By default, this option is checked in a resource's compilation options. For details on compilation options, see page 55.

When switching an application to Debug mode, the Workbench verifies the coherency between the current resource definitions and the resources' compiled code. The Workbench also verifies the coherency between all versions of the resource code. You can access version information for a resource.

While in debug mode, the security state of unlocked resources and resources having no access control switches to read-only mode. The security state of unlocked POU and POU having no access control also switches to read-only mode. Locked resources and locked POU remain locked.

To test a resource "online", its TIC code must be produced and downloaded to the target system, otherwise, the corresponding virtual machine may have been generated with the "C" code of the application.

Status Information

When running a project in Debug mode, status information for resources from the target is updated at a regular interval, indicated by the debug refresh rate. The status information is displayed in resource title bars as:

- resource icons
- text information

You can also choose to refresh the status information for resources at any time.

The debug refresh rate applies to all resource data including variables and status information while in debug mode; its default value is 300 milliseconds. You set the refresh rate in the DefaultRefreshTime property in the [REFRESH] section of the Dta.ini file located in the Workbench's Bin folder. You can also set refresh rates for individual resources by adding the following entry for a resource in the same section: RefreshTime(X)=YYY, where X indicates the resource number and YYY indicates the refresh time in milliseconds.

Note: For a project in normal editing mode, refreshing the status of resources will not refresh the status of resources previously opened by other users for single-resource editing. To refresh the displayed status for these resources, you need to reopen the Workbench project.

Resource Icons

The resource icons appear in the left-most corner of the title bar:

| Icon | Description |
|------|---|
| | The resource belongs to the current project and is running on the configuration. |
| | The resource belongs to the current project and is either running on the configuration but with a different version or it is not running on the configuration but the code of the resource is available on the configuration. |



| Icon | Description |
|-------------|---|
| | The resource belongs to the current project and is not running on the configuration and no code is available on the configuration. |
| | The resource does not belong to the current project but is running on the configuration. In this case, a new empty resource appears with this icon in the link architecture and hardware architecture views of the project. |

Resource icons also display the security state of a resource.

Text Information

The state of the resource appears next to the resource icon before the resource's name, in the title bar.

| Resource State | Description |
|-----------------------|---|
| RUN | The resource is running (real-time mode). You can switch the resource to cycle-to-cycle mode. |
| STOP | The resource is in cycle-to-cycle mode. Possible operations are: - switch the resource to real-time mode - execute one cycle - go to the next step (only when step-by-step mode is instantiated) |
| BREAK | The resource is in break point mode (SFC POU). Possible operations are: - switch the resource to real-time mode - execute one cycle - go to the next step (only when step-by-step mode is instantiated) |
| ERROR | The resource is in error. Possible operations are: - switch the resource to real-time mode - execute one cycle - go to the next step (only when step-by-step mode is instantiated) |

| Resource State | Description |
|----------------|---|
| STEPPING | <p>The resource is in step-by-step mode.</p> <p>Possible operations are:</p> <ul style="list-style-type: none"> - switch the resource to cycle-to-cycle mode returning the resource to the start of its cycle without executing the remaining code - execute one cycle - execute the current step and go to the next one - locate the current step - switch the resource to real-time mode |
| STEPPING_ERROR | <p>The resource is in stepping error mode. This state is caused when an invalid operation occurs such as a division by 0 or a bound check error.</p> <p>Possible operations are:</p> <ul style="list-style-type: none"> - locate the current step to debug it - switch the resource to cycle-to-cycle mode returning the resource to the start of its cycle without executing the remaining code |

The BREAK, STOP, and ERROR states are possible while in cycle-to-cycle mode. I/Os and bindings are done, but POU's are not executed. A complete cycle is executed when you execute one cycle. The STEPPING and STEPPING_ERROR states are possible while in step-by-step debugging mode.

The existence of code on the configuration is indicated with the following text:

| | |
|--------|--|
| CODE | The resource is not running but the code exists (disk or PROM) on the configuration. |
| NOCODE | The code does not exist on the configuration. |

On a running resource, when the version of code in the project differs from the code on the corresponding virtual machine of the configuration, a message is displayed.

To refresh the status of resources

- ▶ From the Debug menu, choose **Refresh Status**.

The status information displayed in the title bars of all resources is updated.

Download

When simulating a project, you do not need to perform a download operation. You perform a download operation for each resource having code to send to a target.

The Download window shows the list of resources making up a project. In this list, resources are displayed next to the name of the configuration to which they are attached. When the resource code contains debug information generated for ST, IL, or LD programs, the word "debug" appears in comments.

Note: Each time you perform a download operation, the Workbench verifies the coherency between the current resource definitions and the resources' code to download. The Workbench also verifies the coherency between all versions of the resource code. You can access version information for a resource.

Conditions necessary to download a resource:

1. The code (corresponding to the resource available on the hardware configuration) must first be generated by building the project or resource. By default, TIC code is generated for a DS800 virtual machine.
2. Verify that the Power Switch parameter in the DS800 screen of ROCLINK 800 is set to on.
3. The computer where the Workbench is installed must be connected to the configuration through a network supported by the Debugger. The standard network used by the Workbench is Ethernet. If downloading a project with a single configuration, then you may also choose to use the ISaRSI network.

In the Download window, select the resources to download by clicking on their name in the list (click again to unselect). You can click **Toggle** to select or unselect a selected resource. You can also choose to select all or unselect all resources.

Download options:

- Start after download, indicates that the virtual machine executes the resource code upon reception

- Save after download, indicates that the virtual machine saves the resource code on the configuration platform upon reception. The code can be saved to a disk, if the platform has one, or another storage method, depending on the platform and the implementation of the virtual machine.

To download the code of project resources

At anytime during a download operation, you can abort the operation by clicking **Abort**.

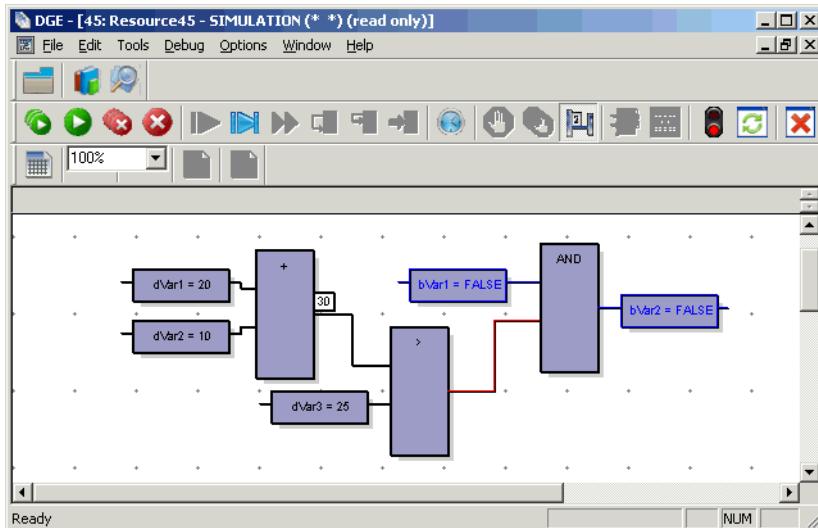
- From the Debug menu, choose **Download** or click  on the Standard Toolbar.

Debug/Simulate

You can test a project using one of two modes:

- Debug, where you test resources online. The debugger establishes the connections with remote configurations. Execution errors and warnings can appear in the Output Window.
- Simulation, where you simulate the running of the project. The Configuration manager and a virtual machine for each resource is launched. The Simulate I/Os Panel is displayed.

While in debug or simulation mode, for FBD and LD programs and function blocks, you can choose to graphically monitor the block (operator, function, or function block) output values. You can temporarily resize variables to enable viewing their output values. Resized variables return to their original size upon quitting debug or simulation mode.



- Output values of boolean type are displayed using color. The output value color continues to the next input. The default colors are red when True and blue when False. You can customize the colors used for the boolean items.
- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are displayed

as a numeric or textual value in a label directly above the output. When the output is a structure type, the displayed value is the selected member.

When the output value is unavailable, the ??? text is displayed in the output label.

To enable the graphic display of output values, you need to generate the symbols monitoring information for individual FB and LD programs and function blocks before building. You can choose to show or hide the output display values.

To start Debug mode

You can start Debug mode from the link architecture view, hardware architecture view, or from a language editor.



- ▶ From the Debug menu, choose **Debug Target** or click , on the Standard toolbar.

To start Simulation mode

You can start Simulation mode from the link architecture view, hardware architecture view, or from a language editor.



- ▶ From the Debug menu, choose **Simulation** or click , on the Standard toolbar.

To show or hide output values

You can choose to display the values of non-boolean outputs in the FBD and LD editors.



- ▶ From the Tools menu, choose **Show/Hide Output Values** or click , on the Debug toolbar.

To stop Debug or Simulation mode

- ▶ From the Debug menu, choose **Stop Debug/Simulation**.

Or



- ▶ On the Debug toolbar, click .

Start / Stop a Resource

The "Start" command on the Debug menu enables you to start a resource which has been stopped. It launches the DS800 virtual machine on the RAS device system and the resource is executed. The resource code must be available on the RAS device system.

To stop the execution of a resource and kill the corresponding virtual machine

- ▶ From the Debug menu, choose **Stop**.

Or



- ▶ On the Debug toolbar, click .

To stop all resources of the project



- ▶ On the Debug toolbar, click .

Resource Execution Mode

You can execute a resource in one of three possible execution modes:

- Real-time
- Cycle-to-cycle
- Step-by-step

When defining a resource's Settings properties, you can set it to automatically start in real-time or cycle-to-cycle execution mode prior to code generation. By default, resources start in real-time mode.

Real-time Mode

Real-time mode is the run time normal execution mode where target cycles are triggered by the programmed cycle timing. While in real-time mode, you can switch the resource to cycle-to-cycle mode. When debug information is generated for POU's in a resource, the resource automatically switches to step-by-step mode when the application encounters a breakpoint.

A resource where real-time mode is activated is in the RUN state.

To activate real-time mode

- ▶ On the Debug toolbar, click  .

Cycle-to-cycle Mode

In cycle to cycle mode, the virtual machine loads the resource code but does not execute it until you execute one cycle or activate real-time mode. When debug information is generated for POU's in a resource, the resource automatically switches to step-by-step mode when the application encounters a breakpoint. You can also switch to step-by-step mode by stepping.

A resource where cycle-to-cycle mode is activated can be in one of three states: STOP, BREAK, and ERROR.

To activate cycle-to-cycle mode



- ▶ On the Debug toolbar, click .

To execute one cycle



Cycle-to-cycle mode must be activated before executing individual cycles.

- ▶ On the Debug toolbar, click .

Step-by-step Mode

You can instantiate step-by-step mode for ST, IL, and LD POU's. For ST and IL POU's, you set breakpoints to specific lines of code. For LD POU's, you set breakpoints to rungs. When you run an application in Debug mode, the application stops when it encounters a breakpoint. At this time, depending on the state of the resource, you can choose to perform various operations:

- Step to the next line of code or rung
- Step into the next line of code or rung
- Execute one cycle
- Switch to cycle-to-cycle mode
- Switch to real-time mode

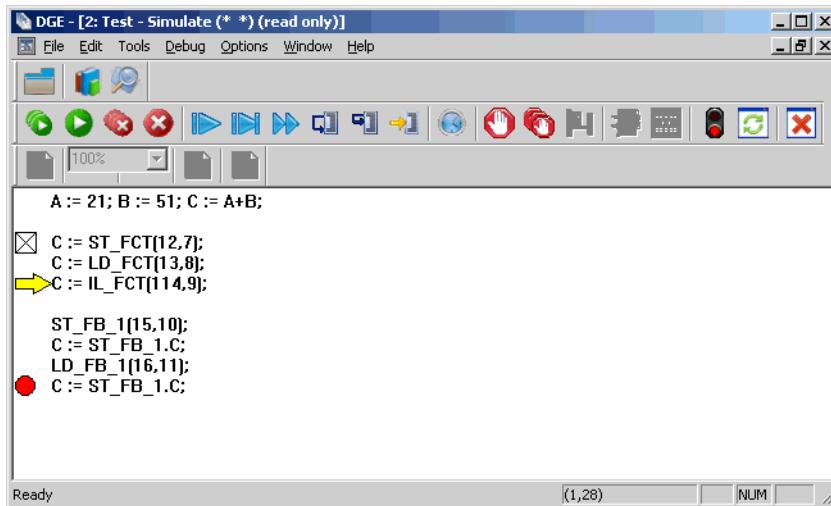
A resource where step-by-step mode is activated is in the STEPPING state. When the resource encounters a stepping error, the resource is in the STEPPING_ERROR state. When stepping within a resource reaches the end of its cycle, the resource automatically switches to cycle-to-cycle mode in the STOP state.

Note: You can only set breakpoints for resources producing TIC code; you cannot set breakpoints for resources producing C source code. Furthermore, you cannot set or remove SFC breakpoints while a resource is in the STEPPING state.

Before setting up step-by-step mode for ST, IL, and LD POU s, you need to specify the generation of debug information for the resource and the individual POU s.

When switching an application to debug mode, to use step-by-step mode, defined breakpoints are sent to the target. When you stop debug mode, you can choose to remove the breakpoints from the target. Breakpoints remaining on a running target may interfere with its cycle.

In the language editor, while in step-by-step mode, defined breakpoints that have been successfully sent to the target appear as red circles to the left of the line of code or rung; breakpoints that are disabled on the target appear as . The current line is indicated with a yellow arrow at its left. When stepping passes beyond the last line or rung of a POU, the arrow points downward (). A Call Stack window shows stepping information such as the name of the POU from which a Step Into command jumped from upon execution.



Setting Breakpoints

You set breakpoints for ST, IL, and LD POU s in the POU editor. Before setting breakpoints in a POU, you need to specify the generation of debug information for the resource and the individual POU s. For details on generating debug information, see page 56.

Note: You can only set breakpoints for resources producing TIC code; you cannot set breakpoints for resources producing C source code.

To set a breakpoint in an ST, IL, or LD POU

1. Click in the line of code or rung on which to set the breakpoint.



2. On the toolbar, click .

A breakpoint appears to the left of the line of code or rung.

Removing Breakpoints

You can remove breakpoints set for ST, IL, and LD POU s for step-by-step mode.

To remove breakpoints

- To remove a breakpoint while in its POU, click in the line of code or rung with the

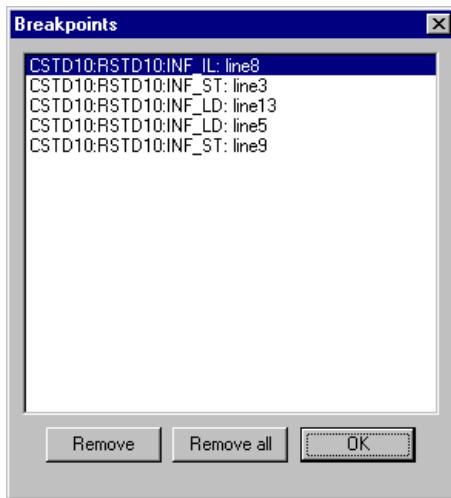


breakpoint, then click

The breakpoint is removed from the line of code or rung.



- To remove a breakpoint from any POU in a resource, click
- on the toolbar, then select the breakpoint from the list and click **Remove**. You can also remove all breakpoints set in all POU s of the resource by clicking **Remove All**.



The individual or multiple breakpoints are removed from their POU's.

Stepping in POU's

While a resource is in the STEPPING state, you can step in a POU (ST, IL, and LD for which you generated debug information) once its execution is interrupted by encountering a breakpoint. You can execute one of two types of steps:

- Step, executes the current line of code or rung then steps to the next line or rung
- Step into, executes the current line of code or rung then steps into the next line of code or rung. When the next line includes a call to a function, stepping continues in the called function then returns to the next line of code or rung in the POU.

When a resource holds POU's for which debug information is generated, stepping is also available while the resource is in either the STOP, BREAK, or ERROR state. However, in these states, stepping jumps to the first line or rung of the first POU for which debug information is generated.

When stepping in POU's, you can locate the current step from within any POU.

To step to the next line of code or rung

1. Select the POU to step in.



2. From the Debug toolbar, click .

The POU executes the current line of code or rung then steps to the next one.

To step into the next line of code or rung

1. Select the POU to step in.



2. From the Debug toolbar, click .

The POU executes the current line of code or rung then steps into the next one and stepping continues in any called function before returning to the next line of the POU.

To locate the current step in a resource



- From the Debug toolbar, click .

Set Cycle Time

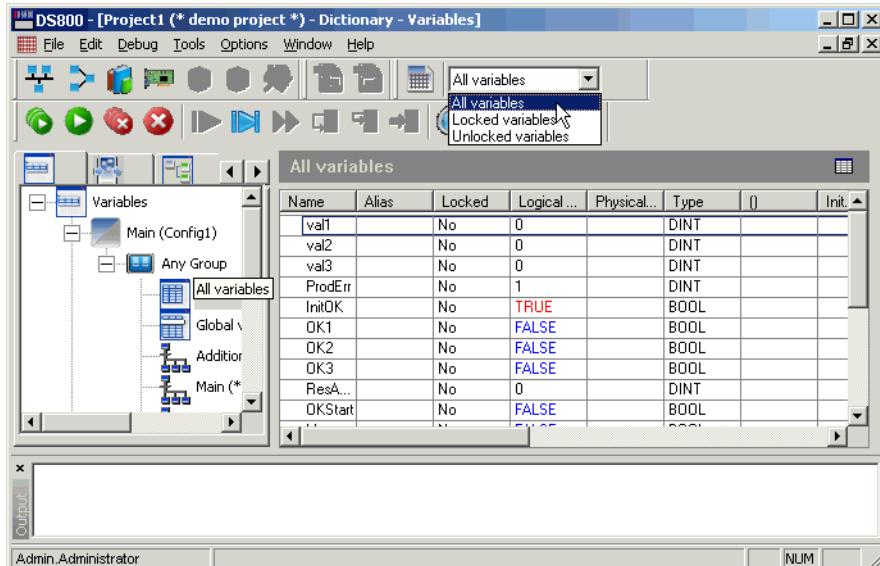
While in debug mode, you can change the cycle time of a resource. You can also set the cycle time before building the code for the resource in the run time settings for the resource. To view the current value of the resource cycle time, from the Debug menu, choose **Diagnosis**.

To change the Cycle Time of the resource

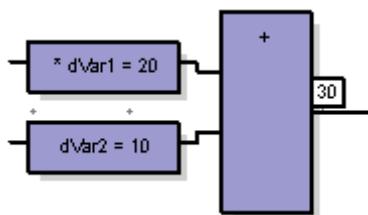
1. Select the resource.
2. From the Debug Menu, choose **Change Cycle Timing**.

Write / Lock / Unlock

While in debug or simulation mode, you can view the values and lock status of variables from within the dictionary view, LD editor, and FBD editor. In the dictionary view, the Locked column indicates whether a variable is locked. You can also choose to display all variables, locked variables, or unlocked variables.



In the LD and FBD editors, the * symbol displayed at the left of a variable name indicates a locked variable.



You lock and unlock variables, and force the values of variables from the dictionary view, LD editor, and FBD editor. You can also unlock variables from the Diagnosis window.

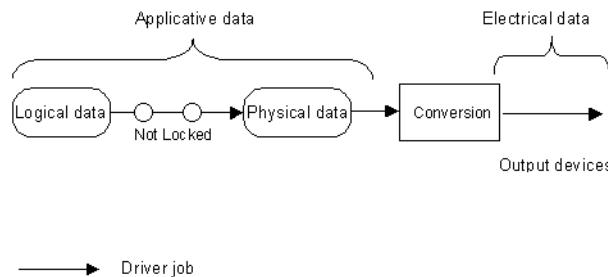
For simple-type members of a complex variable such as a structure or array, locking or unlocking any member affects the entire complex variable. For structure-type variables, you can display one simple-type field. The write mechanism automatically displays the first field of the structure.

For function blocks, you need to instantiate these before locking their parameters.

For locked variables, the values displayed in the Logical Value and Physical Value columns of the dictionary view differ depending on their direction:

| Locked Variable | Logical Value | Physical Value |
|------------------------|---------------------------------|---|
| Direction | | |
| Input | Locked | Updated by the field value |
| Output | Updated by the running TIC code | Locked |
| Internal | Locked | Updated by the consumer binding when one exists or else updated by the running TIC code |

The following diagram shows the lock/unlock process.



To lock variables

Locking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked directly. For array and structure elements, locking a single element causes all other elements to be locked. For function block parameters, locking a parameter affects only that parameter.

- From the dictionary view, double-click the variable's corresponding cell in the Locked column, then in the dialog, click **Lock**.

- From the LD or FBD editors, double-click the variable, then in the dialog, click **Lock**.

To unlock variables

- From the dictionary view, double-click the variable's corresponding cell in the Locked column, then in the dialog, click **Unlock**.
- In the LD or FBD editors, double-click the variable, then in the dialog, click **Unlock**.

You can also unlock variables from the Diagnosis window.

To force the values of variables

- From the dictionary view, double-click the variable's corresponding cell in the Logical column, then in the dialog, enter a value and click **Write**.
- From the LD and FBD editors, double-click the variable, then in the dialog, enter a value and click **Write**.

Diagnosis

You can access diagnostic information for individual resources while running an application in simulation mode. This information is divided into five categories:

- Timing
- System Variables
- Locked Variables
- Breakpoints
- Version Information

For details on run-time settings for resources, see page 59.

To access diagnostic information for a resource

1. While the application runs in simulation mode, select a resource for which to obtain diagnostic information.
2. From the Debug menu, choose **Diagnosis**.

The Diagnosis window displays the diagnostic information for the resource.

Timing

Timing information holds the current values of specific system variables for a selected resource. The timing information is:

- Programmed cycle time, the defined cycle time for the resource
- Current cycle time, the time of the last executed cycle
- Maximum cycle time, the longest period of time used for a cycle, since the resource was started
- Overflow, the number of cycles having exceeded the programmed cycle time

- State, the current state of the resource. Possible states are RUN (real-time mode), STOP, BREAK, ERROR, STEPPING, and STEPPING_ERROR.
- Code, the indication of whether the code has been saved on the target system

For details on setting the cycle time of a resource, see page 303.

System Variables

System variables hold the current values of all system variables for the resource. You can read from or write to system variables. These variables are defined in the **dsys0def.h** file. The system variables are:

| Variable Name | Type | Read/Write | Description |
|----------------------------------|--------|------------|---|
| <code>__SYSVA_RESNAME</code> | STRING | Read | Resource name (max length=255) |
| <code>__SYSVA_SCANCNT</code> | DINT | Read | Input scan counter |
| <code>__SYSVA_CYCLECNT</code> | DINT | Read | Cycle counter |
| <code>__SYSVA_KVBPERR</code> | BOOL | Read/Write | Kernel variable binding producing error (production error). Not supported in DS800. |
| <code>__SYSVA_KVBCERR</code> | BOOL | Read/Write | Kernel variable binding consuming error (consumption error) |
| <code>__SYSVA_TCYCYCTIME</code> | TIME | Read/Write | Programmed cycle time |
| <code>__SYSVA_TCYCURRENT</code> | TIME | Read | Current cycle time |
| <code>__SYSVA_TCYMAXIMUM</code> | TIME | Read | Maximum cycle time since last start |
| <code>__SYSVA_TCYOVERFLOW</code> | DINT | Read | Number of cycle overflows |

| Variable Name | Type | Read/Write | Description |
|----------------------|-------------|-------------------|---|
| __SYSVA_RESMODE | SINT | Read | Resource execution mode. Possible modes are: -1: Fatal error 0: No resource available 1: Stored resource available NOT USED (CMG) 2: Ready to run 3: Run in real time 4: Run in cycle by cycle 5: Run with SFC breakpoint encountered 7: Stopped in stepping mode |
| __SYSVA_CCEXEC | BOOL | Write | Execute one cycle when application is in cycle to cycle mode |
| __SYSVA_WNGCMPTNM | STRING | Read | Warning component name |
| __SYSVA_WNGCMD | SINT | Read/Write | Warning command. Set it to 1 to get next warning |
| __SYSVA_WNGARG | DINT | Read | Warning Argument |
| __SYSVA_WNGNUM | DINT | Read | Warning Number |

Warning: For the __SYSVA_CCEXEC system variable, its use in an ST program is not significant because resources run in cycle-to-cycle mode. Therefore, programs are not executed.

Locked Variables

Locked variables are input, output, and internal variables that have been locked. When deleting locked variables through an online change, these deleted locked variables remain displayed but are preceded by the __DEL__ prefix. To remove these variables from the list, you need to unlock them.

You can unlock variables in the Diagnostic window. You can also unlock variables from the from the dictionary view, LD editor, and FBD editor.

When viewing locked unwired IO channels, these are displayed with their directly represented variable naming.

To unlock variables

- To unlock a single variable, select its name in the list then click **Unlock**.
- To unlock all variables, click **Unlock All**.

Breakpoints

You can view a list of all breakpoints defined for ST, IL, and LD POU's of a resource, for use with the step-by-step mode.



```
IL_prg: line 10
IL_prg: line 21
LD_prg: line 6
ST_prg: line 8
```

Version Information

You can view version information including the compilation version number, the compilation date, and the CRC (Cyclic Redundancy Checking) of the data the resource works on for three sources of resource code:

- the compiled code for the resource in the Workbench project
- the code for the resource running on the target
- the code for the resource stored on the target

| Version | Date | CRC |
|----------|--------------------------|----------|
| Compiled | 5 14:33:39 21-Feb-2003 | F23EAFAC |
| Running | 4 14:31:35 21-Feb-2003 | 166EE91A |
| Stored | 4 14:31:35 21-Feb-2003 | 166EE91A |

SFC Breakpoints

While in Debug mode, you can place SFC breakpoints on SFC steps or transitions. When a breakpoint is encountered, the resource is set to the BREAK state. This mode is equivalent to the cycle-to-cycle mode. Then to overpass the breakpoint, you can choose either to execute one cycle or switch real-time mode. When a resource is in the BREAK state and step-by-step debugging is activated for ST, IL, or LD POU's within the resource, you can also step to the first line of the first POU of the resource for which debug information is generated.

Note: You can only set breakpoints for resources producing TIC code; you cannot set breakpoints for resources producing C source code. Furthermore, you cannot set or remove SFC breakpoints while a resource is in the STEPPING state.

Four types of SFC breakpoints are available:



Breakpoint on Step Activation



Breakpoint on Step Deactivation



Breakpoint on Transition



Transition Clearing Forcing

To set a breakpoint command on a step or transition

You can set breakpoint commands from the Breakpoints toolbar or from the contextual menu.

- ▶ Right-click on the step or transition, then from the contextual menu choose the desired breakpoint command.

Once the breakpoint is reached, you can execute one cycle or switch real-time mode to continue the execution.

To remove breakpoints from steps

You can remove breakpoints from the Breakpoints toolbar or from the contextual menu.

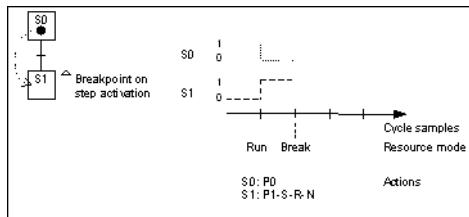
1. To remove a single breakpoint, right-click on the step, then from the contextual menu choose **Remove Breakpoint**.
2. To remove all breakpoints, right-click on a step, then from the contextual menu choose **Remove All Breakpoints**.

Breakpoint on Step Activation

When the step goes from the inactive (no token) to the active (token) state, then breakpoint mode is set for the next cycle. The current cycle goes on executing normally. In particular around the step where the breakpoint is placed, before breakpoint mode is really set:

- All P0 actions, linked to all previous steps that become inactive, are executed.
- All P1 – S – R – N actions, linked to the step that becomes active, are executed.

The following illustrates cycle execution when a breakpoint on step activation is encountered.



To set a breakpoint on step activation

You can set breakpoint commands from the Breakpoints toolbar or from the contextual menu.

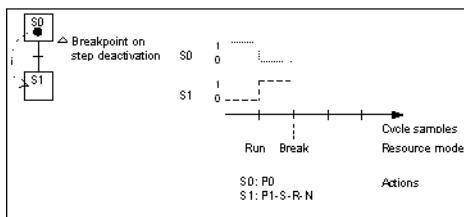
- Select the step, then from the toolbar, click .

Breakpoint on Step Deactivation

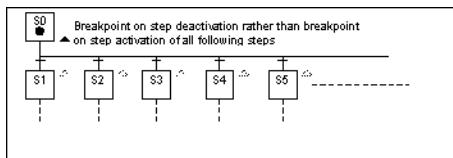
When the step goes from the active (token) to the inactive (no token) state, then breakpoint mode is set for the next cycle. Current cycle goes on executing normally. In particular around the step where the breakpoint is placed, before breakpoint mode is really set:

- All P0 actions, linked to the step that becomes inactive, are executed.
- All P1 – S – R – N actions, linked to all successor steps that become active, are executed.

The following illustrates cycle execution when a breakpoint on step de-activation is encountered.



The behaviors of setting a breakpoint on step activation is the same as step de-activation. These are both available to avoid setting multiple breakpoints as shown below.



Note: On a given step, you cannot set both a breakpoint on step activation and a breakpoint on step de-activation.

To set a breakpoint on step deactivation

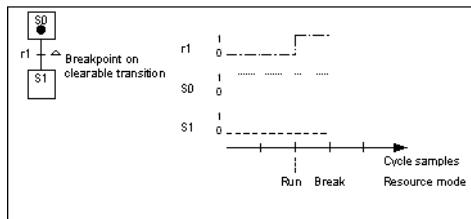
You can set breakpoint commands from the Breakpoints toolbar or from the contextual menu.

- Select the step, then from the toolbar, click .

Breakpoint on Transition

When a transition becomes clearable (transition is valid i.e. all previous steps are active, and its receptivity is true) then breakpoint mode is set for the next cycle. The current cycle goes on executing normally except that the transition is not cleared and therefore related tokens are not moved.

The following illustrates cycles execution when a breakpoint on transition is encountered.



To set a breakpoint on a transition

You can set breakpoint commands from the Breakpoints toolbar or from the contextual menu.

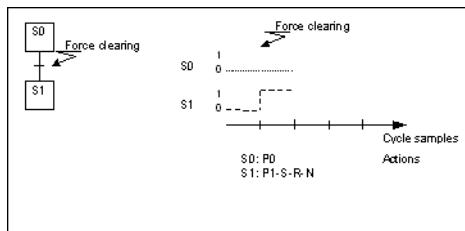
- ▶ Select the transition, then from the toolbar, click .

Transition Clearing Forcing

This debug command allows to force the clearing of a transition whether the latter is valid or not (i.e all previous steps are active or not). Tokens are moved and actions are executed as for a usual transition clearing.

More precisely, tokens of all predecessor steps are removed, if any. Tokens of all successor steps are created. All P0 actions linked to all predecessor steps are executed (even if no token was placed). All P1 – S – R – N actions linked to all successor steps are executed.

The following illustrates cycles execution when clearing of a transition is forced.



Warning: Clearing a transition may lead to abnormal behavior of your chart since it may create several tokens.

To clear a transition

You can clear transitions from the Breakpoints toolbar or from the contextual menu.

- ▶ Select the transition, then from the toolbar, click .

Spying Variables

While in Debug mode, you can choose to spy on selected variables, i.e., view the changes of values for these variables. You spy on variables by adding them to a spy list.

To access the Spy List window

You can access the Spy List window from either the link architecture, hardware architecture, or dictionary views as well as the language editors.

- ▶ From the Window menu, choose **Show Spy List**.

Adding Variables to the Spy List

You can add variables to the spy list from the Spy List window, from the dictionary view, and from ST, LD, FBD, or IL programs.

To add a variable from the Spy List window

1. Within the Spy List window, in the Name column double-click ...
2. From the list of available resources, select the resource holding the variable to spy on.
3. Using the keyboard arrows or the mouse, move to the Name cell, then press **Enter**.

The list of variables available for the resource appears (you may need to resize the Name column to display complete names).

4. Using the keyboard arrows, move within the list of variables to the desired variable, then press **Enter**.

To add a variable from the dictionary view

- ▶ In the dictionary view, select then drag a variable from the dictionary grid to the Spy List window.

To add a variable from ST, LD, FBD, or IL programs

In the language editors, you can add variables to the spy list using the menus, toolbars, or contextual menus.

1. Start the project in Debug mode.

2. Double-click the program.

The editor is launched displaying the program in read-only mode.

3. Select a variable to spy on.

4. From the editor's toolbar, click .

Selecting Variables in the Spy List

You can select one or more variables in the spy list.

To select variables in the spy list

1. To select a single variable, click at the beginning of the line holding the variable.
2. To select more than one line contiguous lines, select the lines holding the variables while holding down the **Shift** key.
3. To select more than one line non-contiguous lines, select the lines holding the variables while holding down the **Ctrl** key.

Removing Variables from the Spy List

You remove variables from the Spy List window

To remove a variable from the spy list

1. In the Spy List window, select the variable by clicking on the very beginning of the line.
2. Press **Delete**.

Rearranging the Spy List

You can change the position of a variable within the spy list.

To change the position of a variable in the spy list

- In the Spy List window, select the variable, then drag and drop it to its new position.

Saving a Spy List

You can save a spy list created for your projects. These lists are saved with the .SPY extension.

To save a spy list

1. In the Spy List window, right-click in the grid.
2. From the contextual menu, choose **Save Spy list**.
3. In the dialog box, enter a name for the file and choose a location then click **Save**.

Warning: You need to save your list each time you make changes.

Opening an Existing Spy List

You can choose to open a previously created spy list.

To open a previously created spy list

1. Within the Spy List window, right-click in the grid.
2. From the contextual menu, choose **Load Spy List**.

Forcing the Value of a Spy List Variable

You can force, i.e., change, the value of a variable in the spy list.

To force the value of a spy list variable

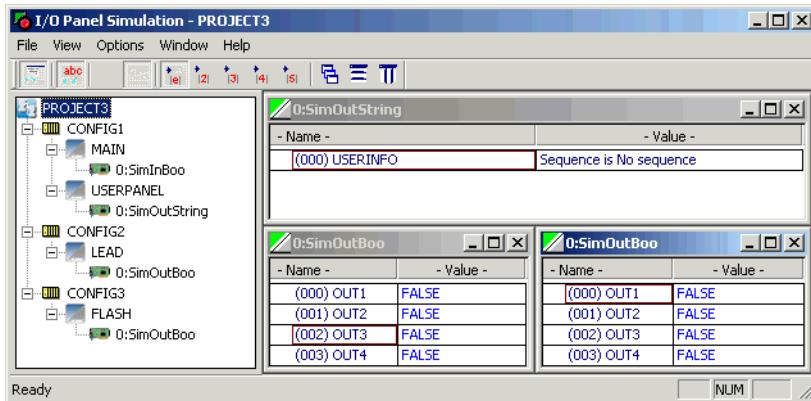
1. Double-click the Value cell of the variable.
2. Enter a new value.

Simulate a Panel of I/Os

You can simulate a panel of I/Os, i.e., display the values of inputs and outputs defined for a project, in their I/O devices. When testing a project in simulation mode, the Simulator (I/O Panel Simulation) is automatically launched. The Simulator is automatically closed when the test mode is stopped. You can perform the following tasks from the Simulator:

- Opening and closing I/O device windows
- Forcing the values of input device channels

The following example shows the Simulator displaying two I/O devices for the "Project3" project:



To display the Simulator

- ▶ While the application runs in Test mode, in the Windows task bar, click .

To open and close I/O device windows

You can choose to open individual I/O devices or all I/O devices belonging to items in a project's structure, i.e. resources, configurations, and projects.

1. From the browser, double-click an item in the structure. You can also drag and drop items into the Simulator's workspace.
2. To close I/O device windows, on the individual windows title bars, click the 'Close Window' button.

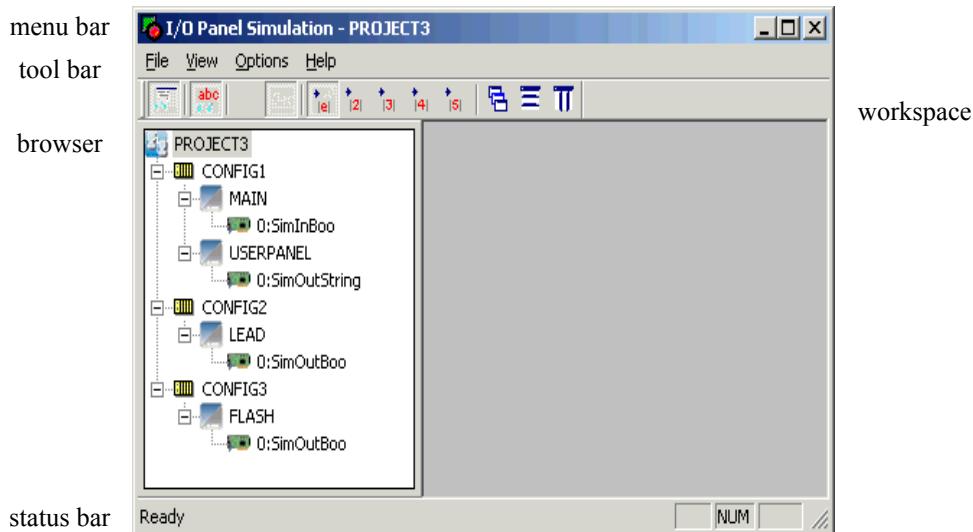
To force the value of an input device channel

You can force, i.e., change, the value of BOOL, numeric-type, and STRING input device channels. For BOOL input devices, forcing the value means changing a TRUE value to FALSE and a FALSE value to TRUE. For numeric-type (SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE) or STRING input devices, this means entering a new value. A value cell is any cell in the 'value column' of an 'I/O device' window.

1. Double-click the value cell of the required input device channel.
2. For numeric-type and STRING input devices, press **Enter**.

Appearance

The Simulator is the environment where you can simulate a panel of I/Os. Its window is divided into two parts: a browser and a workspace.



The browser, located on the left side of the window, displays the defined project items in a tree-like structure, with the project as root. The workspace, to the right of the window, enables you to display the I/O devices defined for the items selected in the browser. Each I/O Device appears in a separate window showing the resource and configuration to which the I/O Device belongs. You can customize many aspects of the Simulator including:

- Resizing and moving individual I/O device windows
- Moving and hiding the browser
- Displaying I/O device window headers

Menu Bar

Some options are available as keyboard commands.

| | | | |
|--------|------------------------------|--------|---|
| File | Open I/O Device | Ctrl+O | creates a new project |
| | Exit | Alt+F4 | leaves the Simulator |
| View | Toolbar | | shows or hides the Simulator's toolbar |
| | Status Bar | | shows or hides the Simulator's status bar |
| | Tree Bar | | shows or hides the Simulator's browser |
| Option | Auto Vertical Tile Windows | | sets the I/O device windows to automatically tile vertically |
| | Auto Horizontal Tile Windows | | sets the I/O device windows to automatically tile horizontally |
| | Auto Cascade Windows | | sets the I/O device windows to automatically display in a cascading manner |
| | Display Header | | displays a header at the top of I/O device windows |
| | Display Name | | displays the variable names associated with each channel in all I/O devices |
| | Numerical Display | | sets the numerical display of values |
| | Auto Save when Exit | | activates or deactivates the automatic saving of changes to the Simulator including the position and look of all I/O device windows. These changes are saved in the current project directory |

| | | |
|--------|--------------------|---|
| Window | Cascade | displays the I/O device windows in a cascading manner |
| | Tile | displays the I/O device windows in a tiling manner |
| Help | Contents | accesses the online help |
| | Search for Help On | not currently supported |
| | About | displays product and version information |
| | Support Info | not currently supported |

Toolbar



shows or hides the Simulator's browser



display the variable names associated with each channel in all I/O devices



displays integer values in the hexadecimal format



displays integer values in the decimal format



sets real values to be rounded off to one digit after the decimal point.
Otherwise, values appear in scientific notation (1.0E+2) format



sets real values to be rounded off to two digits after the decimal point



sets real values to be rounded off to three digits after the decimal point



sets real values to be rounded off to four digits after the decimal point



sets real values to be rounded off to five digits after the decimal point



sets the I/O device windows to appear in a cascading manner



sets the I/O device windows to appear in a horizontal tiling manner



sets the I/O device windows to appear in a vertical tiling manner

Contextual Menu

A contextual menu, accessed by right-clicking within an I/O device window, enables you to change the numeric presentation of values, the display options (I/O window header and variable name), and split the window.

Displaying I/O Device Window Headers

In the Simulator, you can choose to display a header at the top of device I/O windows in the following format:

```
<Resource number>:<Resource name> (<configuration name>)
'Direction:' <'Input' / 'Output'> '- Type:' <type-name>
1: C1_R1 (Config C1)
Direction: Input - Type: BOOL
```

To display window headers

- From the Option menu, choose **Display Header**.

Moving or Hiding the Browser

You can move, resize, or hide the Simulator's browser. To undock it, click on the 'double line' and drag the window. You can move it to the top, bottom, left, and right of the workspace or completely outside of the Simulator window.

To move or hide the browser

1. To move the browser, click its frame then drag it to the new location.
2. To hide the browser, from the View menu, choose **Treebar**.

Online Changes

You can modify a resource while it runs. This is sometimes necessary for chemical processes where any interruption may jeopardize production or safety. When performing online changes, you can choose to update a running resource at the time of download or at a later time. However, online changes should be used with care. **DS800 Development Suite** may not detect all possible conflicts generated by user-defined operations as a result of these online changes.

To perform an online change

- ▶ From the Debug menu, choose **Online Change: Download**, then choose the desired option.

Code Sequences

A sequence of code is a complete set of ST, IL, LD, FBD 61131, or FBD 61499 instructions executed in a row. In a cyclic program, a code sequence is the entire list of instructions written in the program. In an SFC or FC program, a code sequence is the level 2 programming of one step / action or transition / test.

An online change consists in replacing one or more code sequences, without stopping the PLC execution cycle. Therefore, you cannot add, delete, or rename any POU. Note that in such a case, no compiler warning is generated and the changes will be denied at download step.

Particular case of SFC

Since the control of SFC tokens is very critical, you cannot modify an SFC structure or add, renumber, or remove a step or transition.

The switch occurs between two cycles:

- In the case of a step that was already active, if the new code of the step contains non-stored boolean or SFC child actions or P1 actions, then such actions are not updated. Afterwards when the step becomes inactive, the Boolean is reset / the SFC child is killed / P0 actions are executed.

- In the case of a step that becomes inactive, if its code sequence has changed, then the new one is used (P0 actions are executed).

Code sequence for receptivity equation of a transition is changed if it is required and it will be evaluated when the transition is valid.

Particular case of IEC 61131 function blocks

You can make changes to the body of an IEC 61131 function block but cannot change its definition. That is to say you cannot change:

- The number of parameters.
- Parameters name, type, direction (input, local, output), dimension for arrays, and string size for string type.

Therefore, in case of graphic languages you cannot add/remove nested blocks ('C' block or IEC 61131 block calls) because they lead to automatic instances and therefore number of parameters modifications. For same reasons you cannot add/remove a 'pulse' variable.

Particular case of calls to Functions

You can add a call to a standard IEC 61131 function. You cannot add a call to a specific RAS device function if it is its first use.

Particular case of calls to Function Blocks

When adding calls to IEC 61131 function blocks, you need to respect the particularities of function block instances. For details on the particularities of function block instances, see page 330.

Variables

As the variable database is a critical part of the resource, it can be accessed at any time by other processes (via multitasking PLC). It is also possible to modify variable values from the Debugger. Therefore, you cannot add, rename, or remove a variable online. However, you can modify the way a variable is used in the application. You can also reserve "unused" internal or I/O variables in the first version of the resource, so that future modifications can make use of them.

Target databases contain different styles of variables each having their own limitations.

Declared Variables

Declared variables are declared using the Dictionary. You can add new variables with or without initial values and you can remove variables. You cannot:

- Add a variable with the name of a previously removed variable
- Add/Remove an I/O variable
- Change the definition of an existing variable

The definition of a variable refers to many aspects:

- Name
- Type
- Scope
- Dimension (arrays)
- String size (for string type)
- Direction (Input / Output / Internal)
- Address
- Retain attribute

Notes:

- Renaming variables has the same effect as removing and adding them, i.e., their values will be lost in the RAS device
- During code generation, the Workbench linker keeps information about removed and added variables in the PLC data memory map. Therefore, before performing complete downloads instead of online changes, you should clean the project before building it.
- When the initial value of an existing variable is changed, no warning message appears but the modification is not taken into account by the target at online change stage. If changes have been saved, the new initial value take effect at the next 'Stop' /'Start'.

Function Block Instances

Each instance of IEC 61131 or RAS device written function corresponds to data stored in **DS800 Development Suite** virtual machine real time database. You cannot add new automatic instances of IEC 61131 function blocks or of standard 'C' function blocks with or without initial values. To enable online changes, you need to work with function block instances declared in the Dictionary.

You cannot add any user RAS device Function Blocks instances.

Compiler Allocated Hidden Variables

The compiler generates "hidden" temporary variables to solve complex expressions. The compiler forces a minimum number of temporary variables to be allocated for each program, even if not used for compiling the first version of the resource. As long as a new compiling of the resource gives a number of allocated temporary variables lower than this minimum, the online change will be possible.

I/O Devices

Since the I/O system is very open, required modifications should be implemented by an integrator, using specific features of the corresponding hardware.

For simple or complex I/O devices, when supported by the driver, you can perform online changes for OEM parameters. For I/O channels, also when supported by the driver, you can perform online changes for the Gain, Offset, Direct, and Conversion parameters as well as the mapping of logical and physical channels. You cannot add, connect, or remove an I/O variable, or modify the description of an I/O device online. Operations such as modifying device parameters may be available using specific functions provided by the integrator.

Memory Requirements

In order to support the "Online Change" capability, the target PLC must have free memory space to enable the storage of:

- The modified version of the code sequences. Original code and modified code have to be stored in PLC memory.
- The addition of new data variables

Online changes will be denied if there is not enough memory space. You specify the available memory for online changes in the Advanced settings for resource properties. For details about advanced settings for resources, see page 59.

Miscellaneous Limitations

As described before, you can change code sequences and add or remove variables with some limitations. However, you cannot change the descriptions of I/O devices. Other limitations exist for various items of a project:

- Types, you cannot add, remove, or change types definitions. When required, you could define extra types. Such extra types could then be used for future changes.

- Bindings, for some changes made to bindings, no warning message appears during compilation and modifications are not taken into account by the target at online change.
- Resource properties, for some changes made to other options, no warning message appears and the modification is not taken into account by the target at on line change.

During compilation, changes that are not allowed are detected result in the generation of warning outputs. Online changes are denied. The target also does some extra checks. However this function should be used with care. **DS800 Development Suite** may not detect all possible conflicts generated by user-defined operations as a result of these online changes.

Operations

Modifying a running resource consists of the following operations:

1. Modifying the resource source code on the Workbench
2. Generating the new resource code
3. Downloading the new resource code using "Online change: download" command on the Debug menu (instead of "download")
4. Switching from the old resource code to the new one in between PLC execution cycles, using the "Online change: update" command on the Debug menu

This procedure guarantees that the RAS device Target always has a complete and reliable running resource, and enables you to control the timing of the sample operations in a very safe and efficient way. It also enables the user to modify the project when required.

Regardless to the process itself, the "Online Change" is essentially the same as a normal "stop, download and start" set of commands. The only differences are that no variable state is lost and the switching time is very short (usually 1 or 2 cycle duration). During the switch, no variable is modified, and all internal, input, or output variables keep the same value before and after the resource modification. During the switch, no action is performed, and SFC tokens are not moved.

Detailed operations:

1. Before making any change on a running application, it is highly recommended to make a copy of the current project under another name.
2. Before editing any program, you should edit the description of each POU that will be modified and indicate the current date and the nature of the modification, to ease future program maintenance. Select the POU and use the "Tools / Edit Description" command.
3. When one or more allowed changes have been made, the code of the new resource must be generated on the workbench before downloading. Use the Project / Build Resource

command  .

4. Use the "Debug / Online Change: Download" command  . In the dialog box displayed, check the options as desired:
 - Update and Save after download
 - Update after download
 - Update later

The modified code is downloaded by selecting the "Download" button. This may slightly slow down the RAS device target during transfer.

To save your change later, once it is validated, use the command "Debug / Save code on target". This command saves the code of the running resource (including changes). To update your change later, use the command "Debug / On-line change: Update".

If you did not update the change after download (above option):

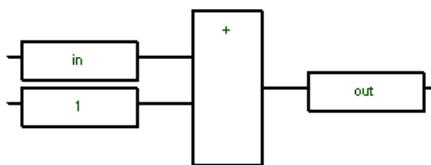
- Using the Debugger, connect the RAS device target and perform any operation which can make the resource update faster, or more safely, then run the "Debug / On-line change: update" command

A message is displayed in the Output window to indicate the success of the switch. If unsuccessful, the existing running application remains as is.

Debug Function Block Instances

You can visually debug instances of function blocks. Function blocks can be written in SFC, ST, FBD 61131, FBD 61499, or LD language. Visual debugging consists of animating the source code of the function block body with the data of a specified instance of the block.

Below is an example of a very simple function block programmed in FBD. The `LIB_FBD1` function block has the `in` input and the `out` output and a constant having a value of 1:

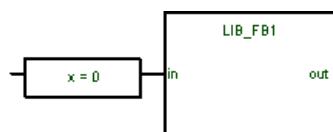


You can distinguish two types of instances of function blocks:

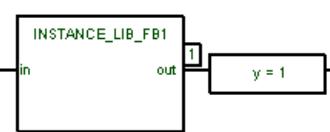
- Declared instances declared in the variable dictionary. These instances are considered as variables.
- Automatic instances created in LD or FBD diagrams. The compiler automatically assigns a unique identifier to each automatic instance. This identifier consists of the `__INST` prefix and a sequential number before the function block's name.

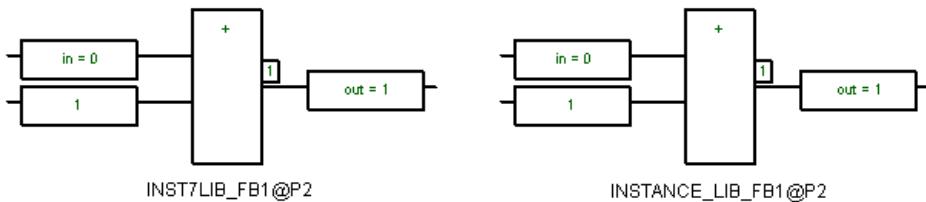
At debug time, you can select instances within a program to open and visualize their diagram. The following examples show the `LIB_FBD1` function block used as an automatic instance and a declared instance in the `P2` program. The upper diagrams show the instances in the program, whereas, the lower diagrams show the individual instances open.

Automatic Instance



Declared Instance





The automatic instance is assigned the `INST7LIB_FB1@P2` name and the declared instance retains its defined name, `INSTANCE_LIB_FB1@P2`. For automatic and declared instances, a suffix consisting of the @ symbol and scope is added to the instance name.

To debug declared instances of function blocks

You can debug variables declared instances of function blocks either from the dictionary, in the LD and FBD diagrams, and in the resource window. However, when declared instances are from a library, you can only debug these from the dictionary or from the LD and FBD diagrams.

Note: You cannot debug function block instances declared as parameters of function blocks.

- ▶ To debug a declared instance do one of the following:
 - From the dictionary view, select the block then from the Debug menu, choose **Debug FB**.
 - In the function blocks section of a resource window, locate the block then double-click it.
 - In the LD or FBD diagram, locate the block then double-click it.

To debug automatic instances of function blocks

You can only debug automatic instances of function blocks from the LD and FBD diagrams.

- ▶ Open the LD or FBD diagram where the instance is inserted then double-click it.

Clean Stored Code

If you have downloaded a resource with the "Save" option checked in the Download dialog box, the resource's code is stored on the target system. Then if the target RAS device system restarts, it will load this code and start a virtual machine to run this code.

Note: If you want to clean (i.e. remove) this code from the target and avoid restarting on it, from the Debug menu, choose **Clean Stored Code**.

Document Generator

You can build and print the complete or partial documentation for the current project from within the Document Generator.

You can access the Document Generator from the hardware architecture view, link architecture view, dictionary view, or any of the language editors. The Document Generator window has three tabs:

- Table, showing a table (or tree) representing all items that can be printed for the current project
- Options, showing a list of printing options
- Preview, displaying a preview of the project to print

To print the documentation for a project

You can choose to print from any tab of the Document Generator.



1. From the File menu, choose **Print** or click on the Standard toolbar.

The Document Generator is displayed.

2. On the Table tab, select the project items to print.
3. On the Options tab, set the desired printing options for the project documentation.
4. On the Preview tab, review the appearance of the documentation print job.
5. Click **Print**.

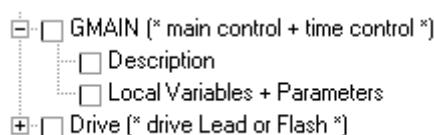
Building and formatting a project's documentation may take a few minutes. Before running other commands in the Workbench, you should wait until the printing task is completed. Building the whole documentation may require a large space on the hard disk. If the disk is full, an error message is displayed, then you need to either free up disk space by removing files or reduce the size of the print task.

Table of Items

The table of items displays all items available for printing for a project. The items preselected for printing differ depending on the location from which printing is initiated. For example, when you initiate printing from the link architecture view where the Main resource is selected, all items defined for this resource appear selected in the Document Generator. When you initiate printing from a program, only the items defined for the program appear selected in the Document Generator. You can always choose to select other items for printing.

You expand or collapse a branch of the tree, by clicking the symbol before an item.

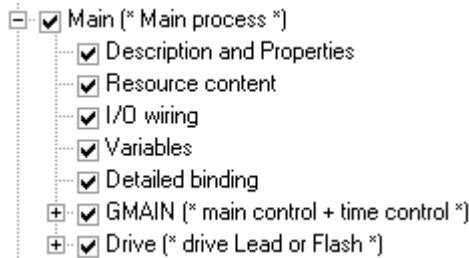
Clicking here collapses GMAIN sub-tree



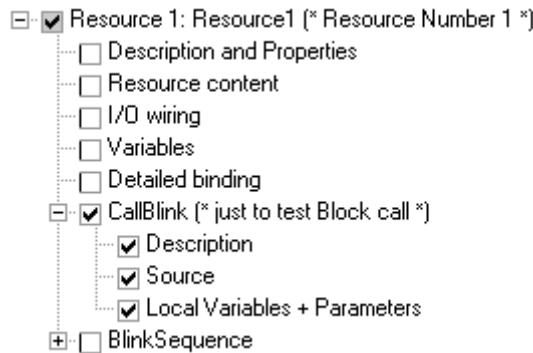
Clicking here expands Drive sub-tree



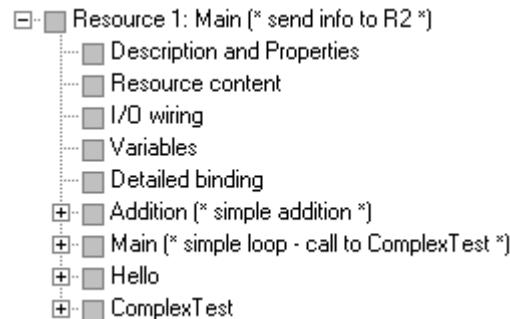
You select items for printing by checking the box at their left. You deselect items by unchecking them. Checking an item at the top of a sub-tree automatically selects all items below it for printing. In the following example, only the Main resource was checked for printing:



When some (but not all) items within a tree are selected, the check box at the top of the structure is grayed:



When items such as projects, resources, or POU's are password-protected (locked), these are unavailable for printing and appear grayed:

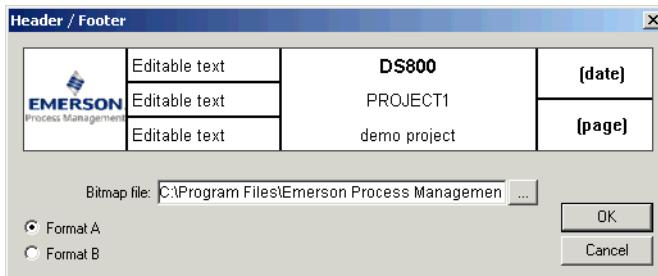


Printing Options

Project documentation uses the default printer settings specified for your computer. However, you can define many other printing options. You can choose to place each item on a new page. You can also choose to print diagrams in landscape orientation. This option sets the printing of all FBD and LD diagrams using the landscape orientation while printing all other items using the portrait orientation. FBD and LD diagrams including guideline areas are automatically scaled to fit the width of the printed pages. You can also specify printing options for the following documentation aspects:

- **Header / Footer.** You can choose to display document information including the date and page count as a header at the top of each page or as a footer at the bottom of each page. You can also choose to have no headers or footers. You can modify the contents of the displayed header or footer by clicking Edit in the Header/Footer section of the printing options.

You can choose to use one of two formats as header/footer. One format provides three fields where you can enter text. In both formats, you can change the logo by entering the path and filename of a bitmap (.bmp) file. Click "..." to browse and select your file.



When replacing the bitmap for the format B option, you need to use one consistent with the resolution of your printer. For example, the default bitmaps are consistent with a 600 dpi printer.

- **Page numbering.** You can specify the page numbering method used for the project document printing: page count (#/total number of pages), page number (#), or section number (#.#.#.#).

For page count, the page section in the header/footer displays the page number out of the total number of pages and the table of contents starts count at 1.

For page number, the page section in the header/footer displays the page number and the table of contents starting count at the Start Page value. When no value is specified, page numbering begins at 1.

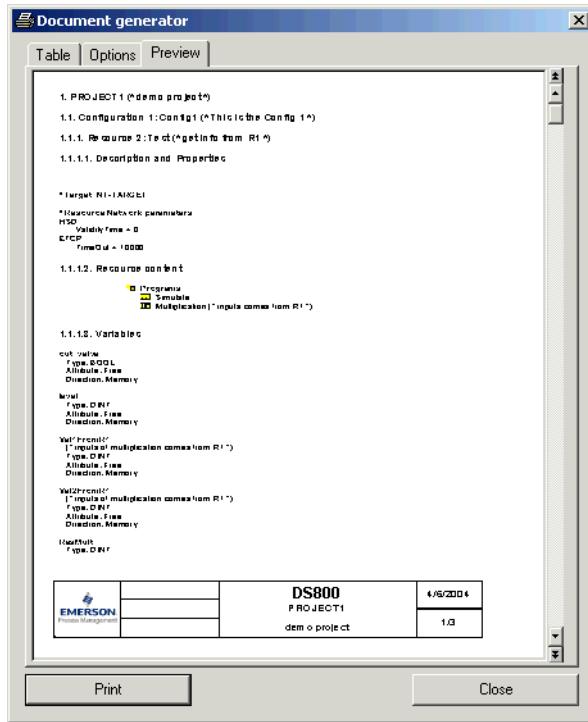
For section number, the page section in the header/footer displays the page number and the table of contents starting count at the Start Section value after the table of contents page; the header and table of contents pages use the lower-case Roman numerals i and ii, then section numbering begins. When no value is specified, section numbering begins at 1.

You can only include page numbering in a header or footer.

- **Cover page.** You can choose to include the header or footer on the cover page of the project documentation. You can also choose to add a printing history. When the printing starts up, a dialogue box is displayed where you can enter a note describing the actual print command. Such notes are stored in a history file and are printed on the first page of any future document (including the present one).
- **Margins.** You can choose to include visible margins on all pages. When checked, the width of each margin (top, bottom, left, right) is user-definable, using the corresponding edit boxes.
- **Fonts.** You can change the font used to print text by clicking Text font and making the desired changes. You can change the font used to print all titles (corresponding to items listed in the table) by clicking Title font and making the desired changes.

Preview

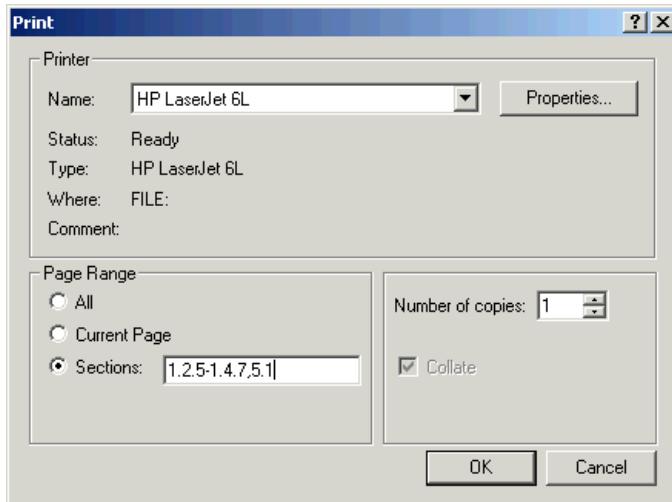
You can choose to preview a document with the selected items before printing. You can scroll the complete document. You can also print previewed pages.



Print

Close

While printing pages, you can choose to print all of a document, the current page, or specific sections of the document. When defining the printing of a range between sections, you need to specify the start and end section.



Since pagination for project documentation is set using the default paper dimensions specified for your computer, when changing the paper dimensions from the Print dialog, the pagination for project documentation differs from the preview.

Code Generator

The Code Generator is launched with the "Build ..." commands of the Workbench and editors. The Code Generator shows compilation errors in the output window.

You can build the code of a single POU, a complete resource, or a whole Project.

Warning: Before building code, you should save all programs currently being editing. Furthermore, for each resource, you could also verify the target type and the type of code to generate.

Build

Before downloading code onto your RAS device target systems, you must first build the code of the whole project. This operation builds the code of all resources of the project, and builds information used to recognize your systems on networks. You cannot build projects open in the read-only mode.

Once a project has been built, subsequent builds only recompile the parts of the project needing recompilation. You can choose to rebuild a project, i.e., recompiling a whole project, to ensure that the complete compiled version is up-to-date with the current Workbench project. You can rebuild projects following a date change on a system or relocation of a project onto a different computer.

You can choose to clean projects. However, after cleaning a project, you cannot perform online changes. Therefore, to retain the ability to perform online changes, you can rebuild a project rather than cleaning then building it.

While performing builds, the security state of unlocked resources and resources having no access control switches to read-only mode. The security state of unlocked POUs and POUs having no access control also switches to read-only mode. Locked resources and locked POUs remain locked.

To build the project

- From the Project menu, choose **Build Project** or click  on the Standard toolbar.

If the hardware architecture view is not changed, building resource code is enough to update one Virtual Machine.

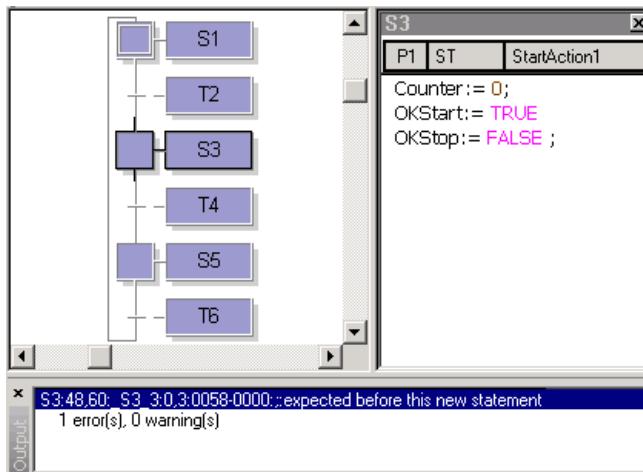
To rebuild a project

- From the Project menu, choose **Rebuild Project** click  on the Standard toolbar.

Build a POU

While editing a POU, the "Build program" command allows you to verify programming syntax errors for the current program.

Error messages are displayed in the Output window. Double-clicking on the error message places the caret on the error or, for graphic programs, selects the erroneous graphic element.



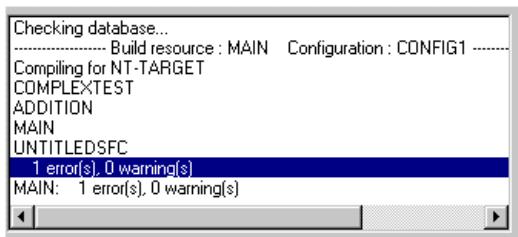
The Build program command verifies the current program even if it has not been modified since its last verification.

While performing builds, the security state of unlocked POU s and POU s having no access control switches to read-only mode. Locked POU s remain locked.

Building Resources / Projects

The "Build Resource" or "Build Project" command displays the number of the error detected in all POU's in the Output window.

Double-clicking on the number of **errors** of a POU opens the corresponding editor for corrections to be made.



The "Build Project" performs the "Build Resource" command for all resources of the project and builds information used to recognize configurations on networks.

Note: While in single resource mode, you cannot build a resource having links to libraries located on a different computer.

The "Build Resource" command constructs the entire code of the resource. Before generating anything, this command checks the syntax of the declarations and programs of each resource. Errors that cannot be detected during single program compiling are detected using these commands. For example, the IO Wiring and Binding Links are checked.

While performing builds, the security state of unlocked resources and resources having no access control switches to read-only mode. Locked resources remain locked.

Programs which have already been checked (with no errors detected) and have not been modified since their last "Build program" operation are not re-compiled. Variable declaration verification and coherence checking are always performed.

Stopping Builds

You can stop a build, i.e., compilation, in progress for a project, resource, or POU. This feature is not available when using a PROPI interface. When a build process is stopped, it can be restarted without affecting the incremental or full compilation. After a build is stopped, online changes can be performed since a copy of the last build is kept until a complete new one is generated.

To stop a build

- ▶ From the Project menu, choose **Stop Build**.

Or

- ▶ On the Standard toolbar, click  .

You can also abort build operations by pressing the <**ESCAPE**> key.

Cleaning Projects

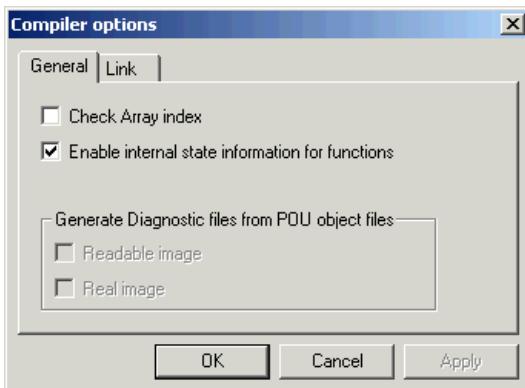
The "Clean Project" or "Clean Resource" commands (on the Project menu of the Workbench) simulate a modification of all the project's (or resource's) programs, so that they are all verified during the next "Build Project" or "Build Resource" operation.

Note: After cleaning a project, you cannot perform online changes. Cleaning projects or resources actually deletes all files generated during the last "Build" command. Therefore, to retain the ability to perform online changes, you can rebuild a project rather than cleaning then building it.

Compiler Options

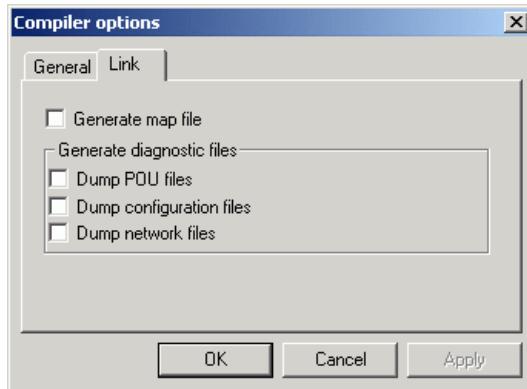
Compiler options are defined for each resource. These options enable setting up the parameters used by the Code Generator to build and optimize the target code. In the Compilation Options of a resource, you select the type of code to generate according to corresponding targets and set up the optimizer parameters according to the expected compilation and run-time requirements. For details on resource compilation options, see page 55.

The general compiler options are the following:



- Check array index, enables the verification of array indices
- Enable internal state information for functions. Functions containing no internal state information denotes that the invocation of a function with the same arguments always yields the same values.
- Generate Diagnostic files from POU object files, *To be defined*

The link compiler options are the following:



- *To be defined*

To access the compiler options for a resource

The general and link compiler options are accessed for individual resources from the resources properties window.

1. From the Window menu, choose **project_name-Link Architecture**.
2. Select a resource.
3. From the Edit menu, choose **Properties**.

The Resource Properties window is displayed.

4. On the Target/Code tab, click **Compiler Options**.

The Compiler Options dialog is displayed.

C Source Code

The workbench compiler produces, by default, TIC code (Target Independent Code) that can be executed by virtual machines. The compiler also enables the production of code in "C". You select code production in the Compilation Options of a resource. For details on resource compilation options, see page 55.

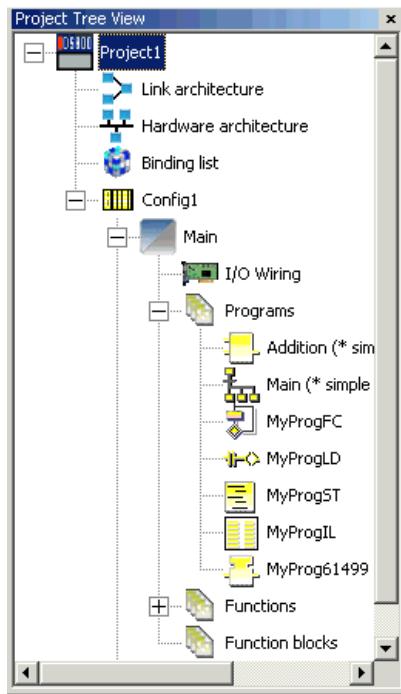
POUs written in FC (Flow Chart), FBD, LD, ST, IL and action blocks and conditions of SFC POU's are generated in "C" source code format.

The "C" source files must be compiled and linked to the target libraries in order to produce the final executable code. For further information about recommended implementation techniques, refer to the "I/O Development Toolkit User's Guide".

Note: Some debugging features such as downloading the resource code, online modification, and breakpoints are not available when the resource is compiled using the "C" language.

Project Tree View

The Project Tree View displays the project structure and enables accessing most aspects of the currently opened project. For instance, you can access the link or hardware architecture views, the internal binding list, elements (programs, functions, and function blocks) defined for resources and I/O wiring. You can also access utilities such as the events viewer, trends logger, and driver monitor.



Contextual menus enabling tasks such as locating and opening project elements are available by right-clicking these elements.

To access the Project Tree View

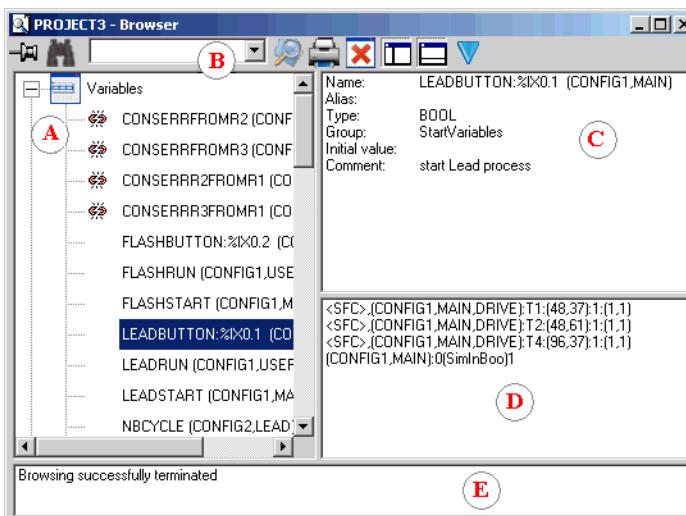
- ▶ From the **Window** menu, choose **Show Project Tree View**.

Cross References Browser

The Cross References Browser is a tool that finds in the POU s of a project all references to global variables, i.e., cross references, defined in a project. It provides a total view of the declared variables in the programs of the project and where these are used. The aim of the browser is to list all the global variables, I/Os, and instances declared in the project, and to localize, in the source of each program the parts of source code where those variables are used. The browser is very useful for a global view of one variable life cycle. This helps localize side effects, and reduce the time to understand the project during maintenance.

The browser is divided into five sections:

- A, the list of global objects declared in a project
- B, the search field where you enter a name to search in the list of objects
- C, the description of the object selected in the list
- D, the locations of the object selected in the list in the project POU s. For variables, the description includes the direction, i.e., READING FROM, WRITING TO.
- E, an output window where messages and error messages are displayed



When viewing global objects in the browser, the  symbol indicates that the object is not used in any POU.

You can perform many tasks from the Browser's toolbar:



keeps the browser always on top



locates the name entered in the Find field (B) from the list of global objects declared in the project (A)



browses, i.e., parse the POU to re-calculate the cross references



prints the cross references



clears the output window



shows or hides the list of declared objects



shows or hides the output window



accesses the available options for the calculation of cross references

To access the browser

You can access the browser using the menu, the toolbars, or from a contextual menu, available by right-clicking in a language editor.

- ▶ From the Tools menu, choose Browser or from the Window Buttons toolbar, click .

Calculating Cross References

When you calculate cross references, these are stored in a cross references file. Such a file is automatically created for each resource of a project. These files eliminate the need to parse POUs each time the browser is closed and re-opened. When files are missing or invalid due to changes in the project, messages are displayed in the output window. The cross reference files are deleted when you clean a project.

When cross references are out of date, the  icon appears in the browser's title bar.

To calculate cross references

- ▶ From the Browser's toolbar, click .

Browsing the POU s of a Project

Occurrences of a selected object in the source files of an open project appear in the locations section of the browser. Double-clicking an occurrence opens the program directly where the object appears.

Defining Search Options

You can define the search options used when finding cross references. The options consist of three types: the global object to search for during the next scan, the objects to list in the browser window, and the exact set of configurations and resources in which to search for selected objects. You can choose to scan the cross references for one or more resources in order to shorten calculation time.

The options for the global object to search for are:

| | |
|-----------------|---|
| Variables | all global variables and I/Os |
| Programs | program names (SFC or FC names can be used in parent programs) |
| Functions | all functions declared in the project or in attached libraries, plus "C" and standard functions available for the corresponding RAS device target |
| Function Blocks | all function blocks declared in the project or in attached libraries, plus "C" and standard function blocks available for the corresponding RAS device target |
| Defined Words | aliases defined in the "Defined Words" section of the dictionary, in the project or in attached libraries |

The options specifying what objects to listed in the browser window are:

| | |
|--------|----------------------------|
| Unused | list unused variables |
| Used | list variables used in POU |

To define search options

Changes only take effect during the next scan.

1. From the Browser's toolbar, click .
2. In the list of available options, check the desired options.

Version Source Control

You can manage the changing versions of Workbench elements including projects, configurations, resources, and POU's by saving them to a version source control database. Saving these elements to a control database enables you to retrieve older versions of the elements at a later time. The information saved in the database also includes advanced options definitions such as alarms and events, field communications, fail-over mechanisms, trending, and Web HMI data servers. Version source control also applies to projects opened in single-resource mode.

You save version source control information to a repository using one of two modes:

- file mode where you specify a path for a local or remote computer
- client/server mode where you specify login information and server location. Before setting this mode, the repository project must exist.

The default uses the file mode and saves this information in a VSC folder in the project folder. A repository folder, defined by the path, can hold multiple version source control projects. You can choose to clear the version source control status for a project. Clearing the version source control status for a project means disabling the version source control for the project. The version source control repository must be removed manually.

Workbench elements are always editable. Therefore, you do not need to check these out of the control database to modify them. At any time, you can check in, i.e., save, changes made to elements in the control database. When you check in an element, all of its descendants are also checked in. For instance, when you check in a project, all of its configurations, resources, and POU's are checked in. You can only check in or get elements available for edition: you cannot check in or get elements having the read-only attribute. However, you can view the history of read-only elements.

When you retrieve, i.e., get, a Workbench element from the control database, this element is automatically updated to the current version. Therefore, a local element containing more current definitions could be overwritten. Before using a retrieved project or configuration, you need to recompile the entire project. Before using a retrieved resource or POU, you need to recompile the resource.

Deleting or renaming previously checked in Workbench elements detaches these from their history in the control database. For instance, before retrieving any part of a deleted resource's history, you need to recreate a new instance of the resource having the same name.

When performing a check in, individual elements are placed in four file types within the control database. For example, a project is split into a project file, a configuration file for each configuration, a resource file for each resource, and a POU file for each POU. The project file contains a list calling its configurations, resources, and POUs. The information retained in each type of file varies:

| Element Type | Retained in Control Database File |
|---------------|---|
| POU | POU properties, local variables, symbols, and advanced options definitions as well as a list of contained child POUs |
| Resource | Resource properties, global variables, internal/external bindings, I/O devices, variable groups, and advanced options definitions as well as a list of contained POUs |
| Configuration | Configuration settings, network connections, and advanced options definitions as well as a list of contained resources |
| Project | Project settings, types, and advanced options definitions as well as a list of contained configurations, and resources |

Stored element files do not retain information such as imported target definitions, compilation output files, driver definitions, and protocols. Each POU also has a second file holding the code and instructions (*POU_name.stf*).

The version control status of an element is indicated in the Workbench. For a project, the status is indicated in textual format in the title bar: Up-to-date or Locally modified. For a configuration or resource, the status is displayed as an icon at the left-hand corner of its title bar. For a POU, the status is applied directly to the POU icon.

-  Up-to-date. The file is identical to the latest version in the source control database or to its retrieved version.
-  Locally modified. The file differs from the latest version in the source control database or from its retrieved version. A modification at any level affects the upwards status of the project elements. For instance, when modifying a resource, the status of the resource as well as the configuration and project to which it belongs become locally modified.

When using version source control with your projects, you can perform the following tasks:

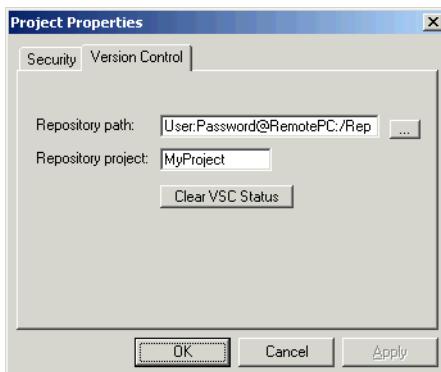
- Performing a Check in of a Workbench Element
- Viewing the History of Workbench Elements

Results and errors for version source control operations are displayed in the output window.

To define a version source control repository

You can choose to save version source control information for a project using the file mode or the client/server mode.

1. With the project open in the Workbench, from the File menu, choose **Project Properties**.
2. In the Project Properties window, select the Version Control tab.



3. In the Repository path and Repository Project field, specify the location in which to save the version source control information by clicking to browse the path.

The syntax to specify a server repository path on a remote computer is as follows:

UserName:Password@RemoteComputer

where *UserName* and *Password* represent the logon information for the remote computer, *RemoteComputer* represents the name or IP address of the computer.

To clear version source control status for a project

1. With the project open in the Workbench, from the File menu, choose **Project Properties**.
2. In the Project Properties window, select the Version Control tab.
3. Make sure the repository path is the correct one for the project, then click **Clear VSC status**.

The version source control information is disabled for the project.

Performing a Check in of a Workbench Element

You can check in, i.e., save, project, configuration, resource, and POU definitions not having the read-only attribute into a version source control database. For elements having access control, the check-in process encrypts the element in the version source control database making them accessible upon entering a valid password.

To check in a project

1. With the project open in the Workbench, from the Tools menu, choose **Check In**, then **Project**.
2. In the Check In dialog, enter a comment (optional), then click **OK**.

The project definitions including all of its configurations, resources, and POUs is saved in the version source control database.

To check in a configuration, resource, or POU

You can check in configurations and resources from the hardware architecture view. You can check in resources and POUs from the link architecture view. You can check in configurations, resources, or POUs using the main menu or from a contextual menu, available by right-clicking the element.

1. In the applicable view, select the element to check in.
2. From the Tools menu, choose **Check In**, then the respective option.
3. In the Check In dialog, enter a comment (optional), then click **OK**.

The element's definitions are saved in the version source control database. For configurations, these definitions include all of its resources and POUs. For resources, these definitions include all of its POUs.

Viewing the History of Workbench Elements

You can view the history of projects, configurations, resources, and POU s that have been checked in repeatedly to the version source control database. Each checked in version appears as a separate entry.

To view the history of a project

- ▶ With the project open in the Workbench, from the Tools menu, choose **View History**, then **Project**.

All previously checked-in versions of the project are displayed.

To view the history of a configuration, resource, or POU

You can view the history of configurations and resources from the hardware architecture view. You can view the history of resources and POU s from the link architecture view. You can view the history of configurations, resources, or POU s using the main menu or from a contextual menu, available by right-clicking the element.

1. In the applicable view, select the element for which to view the history.
2. From the Tools menu, choose **View History**, then the respective option.

All previously checked-in versions of the element are displayed.

Getting a Previous Version

When viewing the history of a project, configuration, resource, or POU, you can choose to get, i.e., retrieve, a previously checked in version of the element. For elements having access control, you can access them upon entering a valid password.

Warning: Since getting an element from the control database automatically updates a locally held version to the retrieved version, a local element or its underlying elements containing more current definitions could be overwritten. For example, getting a project from the control database where a resource and POU have been locally modified since the check in causes the resource and POU to be overwritten with their older definitions contained in the control database.

When you delete or rename a Workbench element that was checked in to the control database, you cannot retrieve any part of the history for this element from the database unless you recreate a new instance of this element having the same name.

To get a previous version of a Workbench element

- ▶ In the History list of elements, select the version to retrieve, then click **Get**.

This older version replaces the current version.

Comparing Current and Previous Versions

When viewing the history of a project, configuration, resource, or POU, you can choose to compare a previously checked in version of the element with the current version or another checked-in version.

To compare a previous and current version of an element

- ▶ In the History list of elements, select the version with which to compare, then click **Diff**.

The response indicates whether the files are different or identical. To navigate between File Differences windows and the Workbench, you need to close the History window.

Accessing Details for a Previous Version

When viewing the history of a project, configuration, resource, or POU, you can access history details on a previously checked in version of the element. These details include the incremental version number, automatically assigned at check in, the date on which the version was checked in, and the identity of the user who checked in the version as well as an optional comment.

To access the history details of a previous version

- ▶ In the History list of elements, select the version for which to access details, then click **Details**.

The History Details dialog is displayed showing the details for the selected version.

Creating a History Report

When viewing the history of a project, configuration, resource, or POU, you can choose to create a report of text format (.txt) on the history of the element. This report lists all or selected incremental checked-in versions, the dates of each check in, and the user that performed each check in. A report can also include the differences from one version to the next. Before sending a report to a file, you can choose to preview it.

To create a history report for an element

1. In the History list of elements, click **Report**.
2. In the History Report dialog, do the following:
 - To include version numbers, check-in dates, and check-in users, check **Include details**.
 - To include the differences between versions, check **Include differences**.
3. To preview the report before sending it to file, click **Preview**.
4. To send the report to file, click **OK**, then choose the location in which to save the file.

Language Reference

This Language Reference is a complete description of all available features for programming ROC applications with this Workbench.

A description of the project architecture, variables and the syntax of each programming language is given, along with a full listing of the standard functions, function blocks and Operators that can be called by programs.

Project Architecture

A Project is composed of configurations. A configuration is one RAS device unit and is composed of one or more resources. A resource represents a DS800 target Virtual Machine. A resource is divided into several programming units called POU (Program Organization Unit). The POU of a resource are linked together in a tree-like architecture. POU can be described using any of **SFC**, **FC**, **ST**, **IL**, **FBD**, or **LD** graphic or literal languages. POU can be programs, functions or function blocks.

Programs

A Program is a logical programming unit, that describes operations between **variables** of the process. Programs describe either **sequential** or **cyclic** operations. Cyclic programs are executed at each target system Cycle. The execution of sequential programs has a Dynamic Behavior.

Programs are linked together in a hierarchy tree. Those placed on the top of the hierarchy are activated by the system. **Child-programs** (lower level of the hierarchy – only for SFC and FC: Child SFC and FC Sub-programs) are activated by their father. A program can be described with any of the available graphic or literal languages:

- **Sequential Function Chart** (SFC)
- **Flow Chart** (FC)
- **Function Block Diagram** (FBD)
- **Ladder Diagram** (LD)
- **Structured Text** (ST)
- **Instruction List** (IL)

The same program cannot mix several languages, except for LD and FBD which can be combined into one diagram.

SFC programs and SFC child programs have dynamic behavior limits which are set at the resource level. Whereas, SFC function blocks and SFC child function blocks each have their own maximum number of tokens which are set in their individual properties.

Cyclic and Sequential Operations

The hierarchy of POU's is divided into three main sections or groups:

- Program Section** Programs located in this part represent the target cycle. Note that inside this section, SFC and FC programs, which represent sequential operations, are grouped together.
- Function Section** Set of functions that can be called by any program.
- Function Block Section** Set of function blocks that can be called by any program.

Programs before and after SFC and FC programs describe cyclic operations, and are not time dependent. They are called **cyclic programs**. SFC and FC programs describe sequential operations, where the time variable explicitly synchronizes basic operations. These are called **Sequential programs**. Cyclic programs are systematically executed at the beginning of each run time cycle. Main sequential programs (at the top of the hierarchy) are executed according to the SFC and FC dynamic behavior.

POU's of the "Functions" section are programs that can be called by any other program in the project. These are called **functions**. A function can call another function.

POU's of the "Function Block" section are programs that can be called by any other POU in the project. These are called **function blocks**. A function block section can call functions or other function blocks.

Main sequential programs must be described with the SFC or the FC language. Cyclic programs cannot be described with the SFC language, neither with the FC language. Any SFC program may own one or more SFC child. Any FC program can "call" one or more FC sub-program.

Functions can be described with the ST, LD, or FBD languages and function blocks can be described with the SFC, ST, LD, or FBD language. Functions and function blocks can be called from actions or conditions of SFC or FC programs.

Programs located at the beginning of the cycle (before sequential programs) are typically used to describe preliminary operations on input devices to build high level filtered variables. Such variables are frequently used by the programs of the sequential programs. Programs located at the end of the cycle (after sequential programs) are typically used to describe security operations on the variables operated on by sequential programs, before sending values to output devices.

Child SFC POUs

Any SFC POU may control other SFC POUs. Such low level units are called **child SFC**. A child SFC POU is a parallel unit that can be started, killed, frozen, or restarted by its parent. The **parent POU** and child POU must both be described with the SFC language. A child SFC POU may have local variables.

When a parent POU starts a child SFC, it puts an **SFC token** (activates) into each initial step of the child. This command is described with the **GSTART** statement or with the name of the child with the S qualifier. When a parent POU kills a child SFC, it clears all the tokens existing in the steps of the child. Such a command is described with the **GKILL** statement or with the name of the child and the R qualifier. When a father POU starts a child, the father continues its execution.

When a parent POU freezes a child SFC, it clears all the tokens existing in the child, and keeps their position in memory. Such a command is described with the **GFREEZE** statement. When a parent POU restarts a frozen child SFC, it restores all the tokens cleared when the child was frozen. Such a command is described with the **GRST** statement.

Child SFC function block instances, as their SFC function block fathers, have a maximum number of tokens, unlike SFC programs whose dynamic behavior limits are set at the resource level. You specify the tokens limit for an SFC function block in its setting properties, accessed by selecting the block, then from the Edit menu, choosing Properties, then the Settings tab.

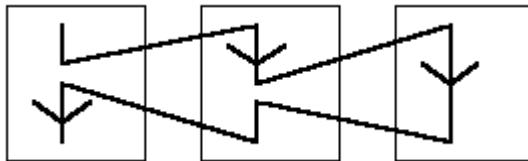
When using an SFC function block with an SFC child, you can access, for read-only purposes, the local values of the child from its father by entering the child's name and the parameter in an action or transition's code. For example, to access the *Local1* parameter of an SFC child named *FB_Child*, in an action or transition defined for the SFC function block father, you would write the following:

```
FB_Child.Local1
```

FC Sub-Programs

Any FC program can call one or more FC program. The **FC Sub-program** execution is driven by its **parent program**. The parent FC program execution is suspended until the FC Sub-program execution ends.

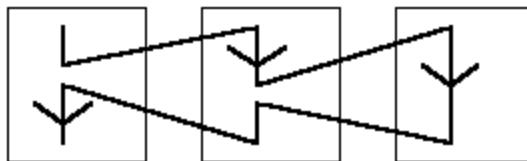
Parent program FC sub-program FC sub-program



Functions

A function execution is driven by its parent program. The execution of the parent program is suspended until the function ends:

Main program Function Function



Any program of any Section may call one or more functions. A function may have local variables. The ST, LD, FBD or IL languages can be used to describe a function.

Warning: The system does not support recursivity during function calls. A run-time error occurs when a program of the "Functions" Section is called by itself or by one of its called functions. Furthermore, a function does not store the local value of its local variables. A function is not instantiated, therefore, cannot call function blocks.

The interface of a function must be explicitly defined, with a type and a unique name for each of its calling (or Input Parameter) or return parameter (or Output Parameter). In order to support the ST language convention, the return parameter must have the same name as the function. There is only one output parameter.

The following information shows how to set the value of the return parameter in the body of a function, in the various languages:

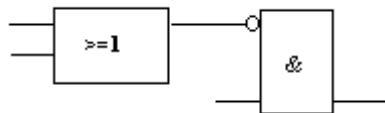
ST: assign the return parameter using its name (the same name as the function):

FunctionName := <expression>;

IL: the value of the current result (IL register) at the end of the sequence is stored in the return parameter:

LD 10
ADD 20 (* return parameter value = 30 *)

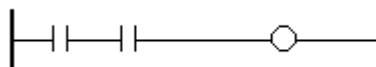
FBD: set the return parameter using its name:



FunctionName

LD: use a coil symbol with the name of the return parameter:

FunctionName



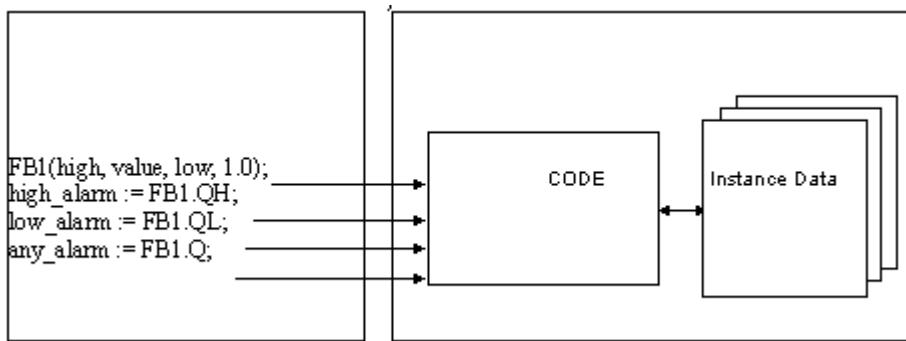
Function Blocks

Function blocks can use the SFC, ST, LD, or FBD languages. Function blocks are instantiated meaning **local variables** of a function block are copied for each Instance. When calling a function block in a program, you actually call the Instance of the block: the same code is called but the data used are the one which have been allocated for the Instance. Values of the variables of the Instance are stored from one cycle to the other.

(* ST Programming *)

Function Block Implementation

(* FB1 is a declared Instance
of the SAMPLE Function Block *)



The interface of a function block must be explicitly defined, with a type and a unique name for each of its calling (or Input Parameter) or return parameters (or output parameters). A function block can have more than one output parameter.

The following information shows how to set the value of an output parameter in the body of a function block, in the various languages:

ST: assign the output parameter using its name concatenated with the function block name

FunctionBlockName.OutputParaName := <expression>;

IL: use LD and ST operator:

LD FunctionBlockName.OutputParaName

ST 20 (* value of Parameter = 20 *)

FBD: set the return parameter using its name:

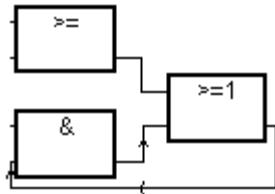
OutputParaName

LD: use a coil symbol with the name of the return parameter:

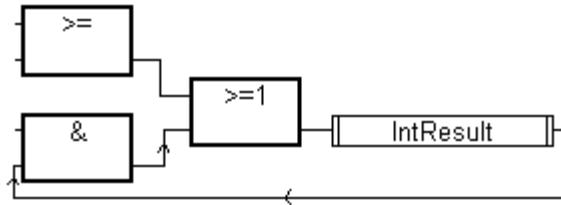
OutputParaName

Warning: When you need a loop in your function block, you must use local variable before doing the loop.

This will not work:



This is OK:



SFC function block instances, as their SFC child blocks, have a maximum number of tokens, unlike SFC programs whose dynamic behavior limits are set at the resource level. You specify the tokens limit for an SFC function block in its setting properties, accessed by selecting the block, then from the Edit menu, choosing Properties, then the Settings tab.

Description Language

A program can be described with any of the following graphic or literal languages:



Sequential Function Chart (SFC) for high level operations



Flow Chart (FC) for high level operations



Function Block Diagram (FBD) for cyclic complex operations



Ladder Diagram (LD) for Boolean operations only



Structured Text (ST) for any cyclic operations



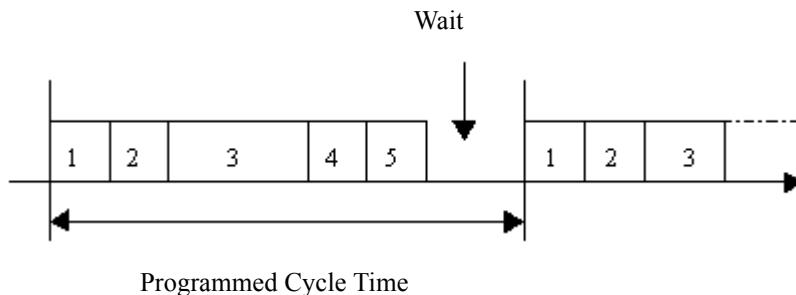
Instruction List (IL) for low level operations

A program cannot contain multiple languages. However, you can combine FBD and LD in a single program. The language used to describe a program is chosen when creating the program and cannot be changed.

Execution Rules

The system is Synchronous. All operations are triggered by a clock. The basic duration of the clock is called the cycle timing:

1. Scan input variables
2. Consume bound variables
3. Execute POU's
4. Produce bound variables
5. Update output devices



In the case where bindings (Data Links between resources) have been defined, variables consumed by this resource are updated after the inputs are scanned, and the variables produced to other resources are "sent" before updating the outputs.

If a cycle time is programmed, the virtual machine waits until this time has elapsed before starting the execution of a new cycle. The POU's execution time varies depending upon the number of active steps in SFC Programs and on instructions such as Jump, IF and Return...

Common Objects

These are main features and common **objects** of the programming data base. Such objects can be used in any POU (Program Organization Unit: programs, functions or function blocks) written with any of the **SFC**, **FC**, **FBD**, **IL**, **ST**, or **LD** languages.

Data Types

Any constant, expression, or variable used in a POU (written in any language) must be characterized by a type. Type coherence must be followed in graphic operations and literal statements.

Types are known by any resource of a Project; types have a common Scope. These types are:

- Standard IEC 61131 Types
- User Types (based on standard IEC 61131 types)

Standard IEC 61131 Types

You can program objects using 17 standard IEC 61131 types:

- BOOL: logic (true or false) value
- SINT: short integer continuous value (8 bit)
- DINT: double integer continuous value (32 bit)
- REAL: real (floating) continuous value (32 bit)
- TIME: time values less than one day; these value types cannot store dates (32 bit)
- STRING: character string having a defined *size*, representing the maximum number of characters the string can contain. For example, to define `MyString` as a string containing 10 characters, enter `MyString(10)`. For information on using string variables, see page 391.

Based on the above standard IEC 61131 types, you can define new user types. Furthermore, you can define arrays or structures using standard IEC 61131 types, arrays, or other user types.

When creating a variable, a dimension can be given to define an array. The following example shows the `MyVar` variable of type `BOOL` having a dimension defined as follows: `[1..10]`

```
FOR i = 1 TO 10 DO  
MyVar[i] := FALSE;  
END_FOR;
```

User Types: Arrays

You can define arrays of standard IEC 61131 types or user types. An array has one or more **dimension**. When an array is defined, a variable can be created with this type and a structure can have a field with this type. Array dimensions are positive DINT constant expressions and array indexes are DINT constant expressions or variables.

Note: Arrays must be declared in the Dictionary View before using them in Functional Block Diagrams (FBD).

Example

1. One-dimensional array:

MyArrayType is an array of 10 BOOL. Its dimension is defined as follows: [1..10].
MyVar is of type MyArrayType.
Ok := MyVar[4];

2. Two-dimensional array:

MyArrayType2 is an array of DINT. It has two dimensions defined as follows:
[1..10,1..3]
MyVar2 is of type MyArrayType2
MyVar2[1,2] := 100;

3. Array of an array:

MyVar3 is an array of MyArrayType; Its dimension is defined as follows [1..3]
FOR I := 1 TO 3 DO
FOR J := 1 TO 10 DO
MyVar3[I][J] := FALSE;
END_FOR;
END_FOR;

User Types: Structures

Users can define structures using standard IEC 61131 types or user types. A structure is composed of sub-entries called **Fields**. When a structure is defined, a variable can be created with this type.

Example

MyStruct1 is composed of:

Field1 which is BOOL

Field2 which is DINT

MyStruct2 is composed of:

Field1 which is DINT

Field2 which is BOOL

Field3 which is an array of 10 DINT

Field4 which is of type MyStruct1

MyVar of type MyStruct2 can be used as follows:

```
Value1 := MyVar.Field1; (* Value1 is of type DINT *)
Ok1 := MyVar.Field2; (* Ok1 is of type BOOL *)
Tab[2] := MyVar.Field3[5]; (* Tab is an array of DINT *)
Value2 := MyVar.Field3[8]; (* Value2 is of type DINT *)
Ok2 := MyVar.Field4.Field1; (* Ok2 is of type BOOL *)
```

Constant Expressions

Constant expressions are relative to one type. The same notation cannot be used to represent constant expressions of different types.

Boolean Constant Expressions

There are only two Boolean constant expressions:

- **TRUE** is equivalent to the integer value 1
- **FALSE** is equivalent to the integer value 0

"True" and "False" keywords are not case-sensitive.

Short Integer Constant Expressions

Short integer constant expressions represent signed integer (8 bit) values:

from **-128 to +127**

Short integer constants may be expressed with one of the following **Bases**. Short integer constants must begin with a **Prefix** that identifies the Bases used:

| Base | Prefix | Example |
|-------------|--------|-------------|
| DECIMAL | (none) | 19 |
| HEXADECIMAL | "16#" | 16#A1 |
| OCTAL | "8#" | 8#28 |
| BINARY | "2#" | 2#0101_0101 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve constant expression readability.

Double Integer Constant Expressions

Double integer constant expressions represent signed double integer (32 bit) values:

from -2147483648 to +2147483647

Double integer constants may be expressed with one of the following **Bases**. Double integer constants must begin with a **Prefix** that identifies the Bases used:

| Base | Prefix | Example |
|-------------|--------|---|
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve constant expression readability.

Real Constant Expressions

Real constant expressions can be written with either **Decimal** or **Scientific** representation. The decimal point ('.') separates the Integer and Decimal parts. The decimal point must be used to differentiate a Real constant expression from an Integer one. The scientific representation uses the letter 'E' to separate the mantissa part and the exponent. The exponent part of a real scientific expression must be a signed integer value from -37 to +37. A real variable has six significant digits.

Example

| | |
|---------|----------|
| 3.14159 | -1.0E+12 |
| +1.0 | 1.0E-15 |
| -789.56 | +1.0E-37 |

The expression "123" does not represent a Real constant expression. Its correct real representation is "123.0".

Timer Constant Expressions

Timer constant expressions represent time values from 0 to 1193h2m47s294ms. The lowest allowed unit is a millisecond. Standard time units used in constant expressions are:

| | |
|-------------|---|
| Hour | The "h" letter must follow the number of hours |
| Minute | The "m" letter must follow the number of minutes |
| Second | The "s" letter must follow the number of seconds |
| Millisecond | The "ms" letters must follow the number of milliseconds |

The time constant expression must begin with "T#" or "TIME#" prefix. Prefixes and unit letters are not case sensitive. Some units may not appear.

Example

T#1H450MS 1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes

The expression "0" does not represent a time value, but an Integer constant.

String Constant Expressions

String constant expressions represent character strings. Characters must be preceded by a quote and followed by an apostrophe. For example:

'THIS IS A MESSAGE'

Warning: The apostrophe "'" character cannot be used within a string constant expression. A string constant expression must be expressed on one line of the program source code. Its length cannot exceed 255 characters, including spaces.

Empty string constant expression is represented by two apostrophes, with no space or tab character between them:

```
" (* this is an empty string *)
```

The dollar ('\$') special character, followed by other special characters, can be used in a string constant expression to represent a non-printable character:

| Sequence | Meaning | ASCII (hex) | Example |
|----------|-----------------|-------------|---------------------------|
| \$\$ | '\$' character | 16#24 | 'I paid \$\$5 for this' |
| \$' | apostrophe | 16#27 | 'Enter \$'Y\$' for YES' |
| \$L | line feed | 16#0a | 'next \$L line' |
| \$R | carriage return | 16#0d | ' llo \$R He' |
| \$N | new line | 16#0d0a | 'This is a line\$N' |
| \$P | new page | 16#0c | 'lastline \$P first line' |
| \$T | tabulation | 16#09 | 'name\$Tsize\$Tdate' |
| \$hh (*) | any character | 16#hh | 'ABCD = \$41\$42\$43\$44' |

(*) "hh" is the hexadecimal value of the ASCII code for the expressed character.

Variables

Variables can be **LOCAL** to one program or **GLOBAL** to a resource. Local variables can be used by one program only. Global variables can be used in any program of the resource. Local or Global information is called the **Scope of the variable**.

Variable names must conform to the following rules:

- Names cannot exceed 128 characters
- The first character must be a letter
- The following characters can be letters, digits or the underscore character

Reserved Keywords

A list of the reserved keywords is shown below. Such Identifiers cannot be used to name a POU or a variable:

- A** ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
- B** BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,
- C** CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
- D** DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
- E** ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END WHILE, ENO, EQ, EXIT, EXP, EXPT,
- F** FALSE, FIND, FOR, FUNCTION,
- G** GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
- I** IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,

| | |
|----------|---|
| J | JMP, JMPC, JMPCN, JMPN, JMPNC, |
| L | LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD, |
| M | MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX, |
| N | NE, NOT, |
| O | OF, ON, OR, OR_MASK, ORN, |
| P | PROGRAM |
| R | R, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR, |
| S | S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SUB_DATE_DATE, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TO_OVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM, |
| T | TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TYPE, |
| U | UDINT, UINT, ULINT, ULLINT, USINT, |
| V | VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, ,VAR_OUTPUT |
| W | WHILE, WITH, WORD |
| X | XOR, XOR_MASK, XORN |

All keywords beginning with an underscore ('_') character are internal keywords and must not be used in textual instructions.

Directly Represented Variables

The system enables the use of directly represented variables in the source of the programs to represent a free Channel. Free Channels are the ones which are not linked to a declared I/O variable. The identifier of a directly represented variable always begins with "%" character.

The naming conventions of a directly represented variable for a channel of a single I/O device. "s" is the slot number of the I/O device. "c" is the number of the Channel:

| | |
|----------------|--|
| %IXs.c | free Channel of a Boolean input I/O device |
| %IBs.c | free Channel of a Short integer input I/O device |
| %IDs.c | free Channel of a Double integer input I/O device |
| %IRs.c | free Channel of a Real input I/O device |
| %ITs.c | free Channel of a Time input I/O device |
| %ISs.c | free Channel of a String input I/O device |
| %QXs.c | free Channel of a Boolean output I/O device |
| %QB.s.c | free Channel of a Short Integer output I/O device |
| %QD.s.c | free Channel of a Double integer output I/O device |
| %QR.s.c | free Channel of a Real output I/O device |
| %QT.s.c | free Channel of a Time output I/O device |
| %QS.s.c | free Channel of a String output I/O device |

The naming conventions of a directly represented variable for a Channel of a complex device. "s" is the slot number of the device. "b" is the index of the single I/O device within the complex device. "c" is the number of the Channel:

| | |
|-----------------|---|
| %IXs.b.c | free Channel of a Boolean input I/O device |
| %IBs.b.c | free Channel of a Short Integer input I/O device |
| %IDs.b.c | free Channel of a Double integer input I/O device |
| %IRs.b.c | free Channel of an Real input I/O device |

| | |
|-----------------|--|
| %ITs.b.c | free Channel of a Time input I/O device |
| %ISs.b.c | free Channel of a String input I/O device |
| %QXs.b.c | free Channel of a Boolean output I/O device |
| %QBs.b.c | free Channel of a Short Integer output I/O device |
| %QDs.b.c | free Channel of a Double integer output I/O device |
| %QRs.b.c | free Channel of a Real output I/O device |
| %QTs.b.c | free Channel of a Time output I/O device |
| %QSs.b.c | free Channel of a String output I/O device |

Example

%QX1.6 6th channel of the I/O device #1 (boolean output)

%ID2.1.7 7th channel of the I/O device #1 in the device #2 (integer input)

Information on Variables

All variables have an Attribute and a Direction.

Variables can have one of the following Attributes:

Free Variable which can be used for reading or writing, with an initial value

Read Read-only variable with an initial value

Write Write-only variable with an initial value

They also have a direction:

Internal Internal variable updated by the programs

Input Variable connected to an input device (refreshed by the system)

Output Variable connected to an output device

Note: Some variables cannot be input or output (Timers for example). Each restriction is indicated in the corresponding section.

Variables of standard IEC 61131 types can be given an Initial Value. The default initial value is 0 or FALSE. The initial value is the value of the variable when the Target starts the first Cycle.

Boolean Variables (BOOL)

Boolean means Logic. Such variables can take one of the Boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in Boolean expressions.

Short Integer Variables (SINT)

Short Integer variables are 8-bit signed integers from -128 to +127.

A **bit of a short integer** variable can be accessed using the following syntax:

MyVar.i

If MyVar is a short Integer.

MyVar.i is a Boolean. "i" must be a constant value from 0 to 7.

Double Integer Variables (DINT)

Double Integer variables are 32-bit signed integers from -2147483648 to +2147483647.

A **bit of a double integer** variable can be accessed using the following syntax:

MyVar.i

If MyVar is an Integer.

MyVar.i is a Boolean. "i" must be a constant value from 0 to 31.

Real Variables (REAL)

Real variables are standard IEEE 32-bit floating values (single precision).

1 sign bit + 23 mantissa bits + 8 exponent bits

The exponent value cannot be less than **-37** or greater than **+37**. A real variable has six significant digits.

Timer Variables (TIME)

Timer means **clock** or **counter**. Such variables have time values and are typically used in Time expressions. A Timer value cannot exceed **1193h2m47s294ms** and cannot be negative. Timer variables are stored in 32 bit words. The internal representation is a positive number of milliseconds.

String Variables (STRING)

String variables contain character strings. The length of the string can change during process operations. The length of a string variable cannot exceed the capacity (maximum length) specified when the variable is declared. String capacity is limited to 255 characters excluding the terminating null character (0).

String variables can contain any character of the standard ASCII table (ASCII code from **0** to **255**). The null character (**0**) can exist in a character string, however, it indicates the end of the string.

Strings have a **size** representing the maximum number of characters that the string can contain. For example, to define the `MyString` string containing 10 characters, you would write `MyString(10)`.

Comments

Comments may be freely inserted in literal languages such as ST and IL. A comment must begin with the special characters "("* and terminate with the characters "*)"". Comments can be inserted anywhere in a ST program, and can be written on more than one line.

Example

```
counter := ivalue; (* assigns the main counter *)  
(* this is a comment expressed  
on two lines *)  
  
c := counter (* you can put comments anywhere *) + base_value + 1;
```

Interleave comments cannot be used. This means that the "(*" characters cannot be used within a comment.

Warning: The IL language only accepts comments as the last component of an instruction line.

Defined Words

The system allows the re-definition of constant expressions, true and false Boolean expressions, keywords or complex **ST** expressions. To achieve this, an identifier name, called a **defined word**, has to be given to the corresponding expression. Defined words have a Common Scope: they can be used in any POU of any resource of the Project.

Example

```
YES is TRUE  
PI is 3.14159  
OK is (auto_mode AND NOT (alarm))
```

When such an equivalence is defined, its **identifier** can be used anywhere in an ST program to replace the attached expression. This is an example of ST programming using defines:

```
If OK Then  
angle := PI / 2.0;  
isdone := YES;  
End_if;
```

Warning: When the same identifier is defined twice with different ST equivalencies, the last defined expression is used. For example:

Define: OPEN is FALSE
 OPEN is TRUE
means: OPEN is TRUE

Naming defined words must conform to following rules:

- name cannot exceed 128 characters
- first character must be a letter
- following characters can be letters, digits or underscore ('_') character

Warning: A defined word can not use a defined word in its definition, for example, you can not have:

PI is 3.14159
PI2 is PI*2

write the complete equivalence using constants or variables and operations:

PI2 is 6.28318

SFC Language

Sequential Function Chart (SFC) is a graphic language used to describe **sequential operations**. The process is represented as a set of well defined **Steps**, linked by Transitions. A Boolean Condition is attached to each Transition. A set of Actions are attached to each Step. For programs, Conditions and Actions are detailed using three other languages: ST, IL, or LD. For function blocks, Conditions and Actions are detailed using only two other languages: ST or LD. From Conditions and Actions, any Function or Function Block in any language can be called.

SFC Main Format

An SFC Program is a graphic set of Steps and Transitions, linked together by oriented **Links**. Multiple connection Links are used to represent divergences and convergences. The basic graphic rules of the SFC are:

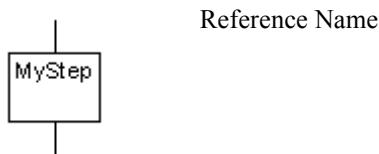
- SFC Programs must have at least one Initial Step
- A Step cannot be followed by another Step
- A Transition cannot be followed by another Transition

SFC Basic Components

The basic components (graphic symbols) of the SFC language are: Steps and Initial Steps, Transitions, Oriented Links, and Jumps to a Step.

Steps and Initial Steps

A step is represented by a single square. Each step is referenced by a name, written in the step square symbol. The above information is called the level 1 of the step:



At run time, a token indicates that the step is active:

Active Step: Inactive Step:



The initial situation of an SFC program is expressed with initial steps. An initial step has a double bordered graphic symbol. A token is automatically placed in each initial step when the program is started.

Initial Step:



An SFC program must contain at least one initial step.

These are the attributes of a step. Such fields may be used in any of the other languages:

StepName.x activity of the Step (Boolean value)

StepName.t activation **duration** of the Step (time value)

(where **StepName** is the name of the step)

When reading a child active step or duration from a father:

ChildName._S1.x activity of the Step (Boolean value)

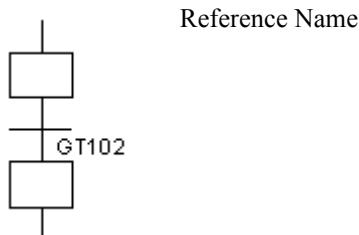
ChildName._S1.t activation **duration** of the Step (time value)

(where **ChildName** is the name of the child. Note that S1 is preceded by two underscore (_) characters)

For details about ST extensions, see page 467.

Transitions

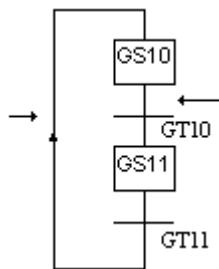
Transitions are represented by a small horizontal bar that crosses the connection link. Each transition is referenced by a name, written next to the transition symbol. The above information is called the level 1 of the transition:



Oriented Links

Single lines are used to link steps and transitions. These are oriented links. When the orientation is not explicitly given, the link is oriented from the top to the bottom.

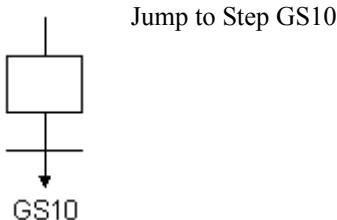
Explicit orientation from
Transistion GT11 to Step GS10



Implicit orientation from
Step GS10 to Transition
GT10

Jump to a Step

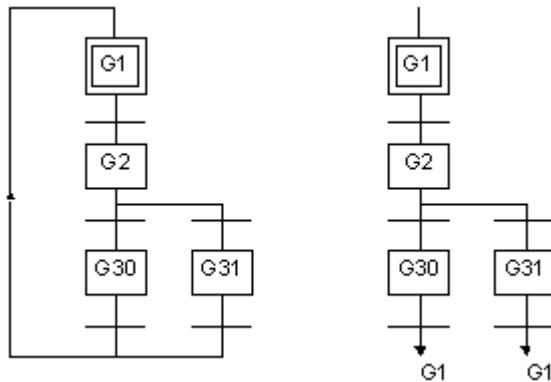
Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the name of the destination step:



A Jump symbol cannot be used to represent a Link from a Step to a Transition.

Example

The following charts are equivalent:



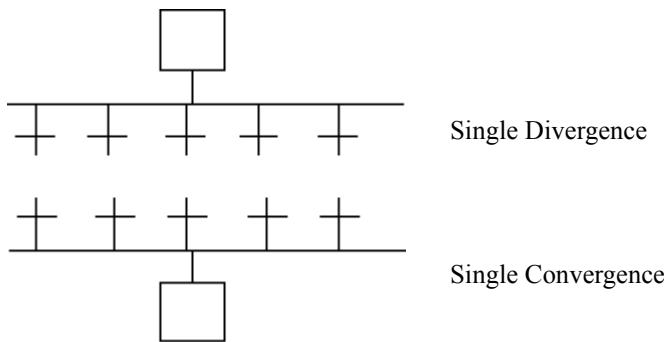
Divergences and Convergences

Divergences are multiple connection links from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

Single Divergences (OR)

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence.

Single divergences and convergences are represented by single horizontal lines.

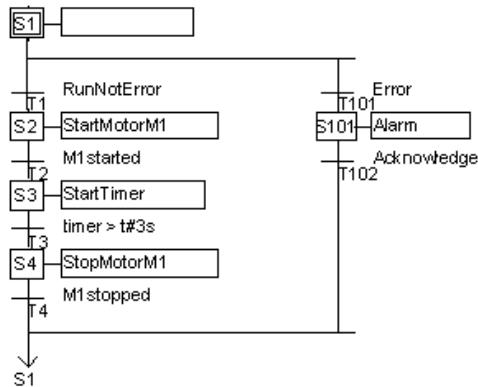


Warning: The conditions attached to the different Transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of transitions to ensure that only one Token progresses in one Branch of the divergence at run time.

Example

(* SFC Program with single divergence and convergence *)

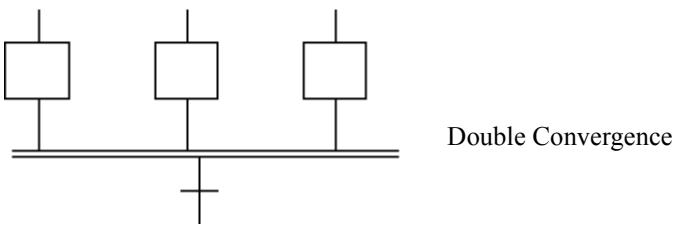
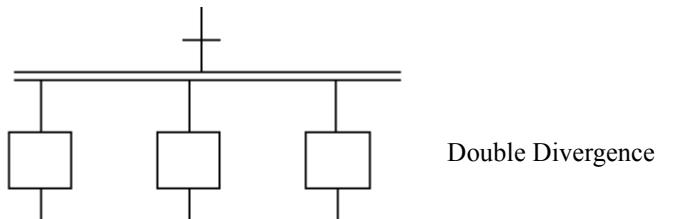
Normal OR divergence:



Double Divergences (AND)

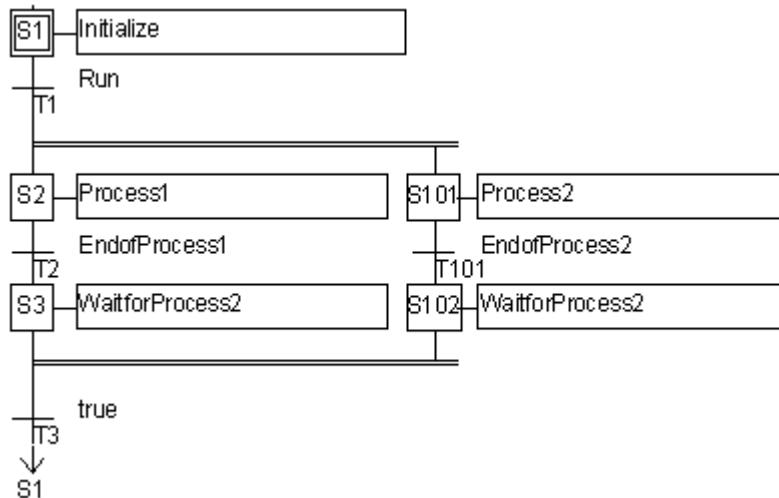
A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence.

Double divergences and convergences are represented by double horizontal lines.



Example

(* SFC program with double divergence and convergence *)



Actions Within Steps

The level 2 of an SFC step is the detailed description of the actions executed during the step activity. This description is made by using SFC literal features, and other languages such as Structured Text (**ST**) or Ladder Diagram(**LD**). The basic types of Actions are:

- Boolean actions with Set, Reset or Non-Stored Qualifier.
- List of instructions programmed in ST, LD or IL with Pulse or Non-Stored Qualifier
- SFC Actions (management of SFC children) with Set, Reset or Non-Stored Qualifier.

Several Actions (with same or different types) can be described in the same Step.

The special feature that enables the use of any of the other language is calling Functions and Function blocks (written in ST, LD, and FBD)

Boolean Actions

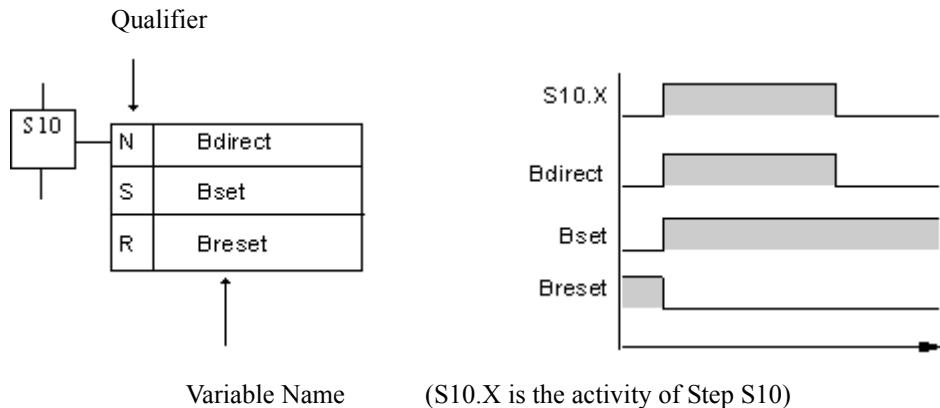
Boolean Actions assign a Boolean Variable with the activity of the Step. The Boolean Variable can be an output or a memory Variable. It is assigned each time the Step activity starts or stops. This is the meaning of the basic Boolean Actions:

N on a Boolean Variable assigns the Step activity signal to the Variable

S on a Boolean Variable sets the Variable to TRUE when the Step activity signal becomes TRUE

R on a Boolean Variable resets the Variable to FALSE when the Step activity signal becomes TRUE

The Boolean Variable must be an OUTPUT or a MEMORY variable. The following SFC programming leads to the indicated behavior:



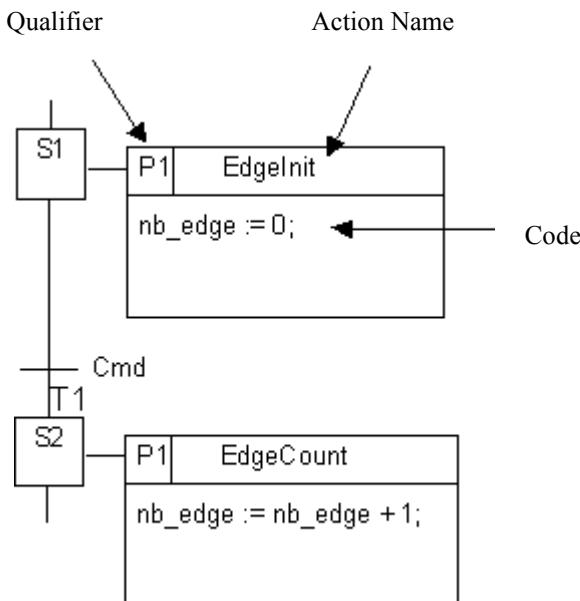
Pulse Actions

A pulse action is a list of instructions, which are **executed only once at the activation of the Step: P1 Qualifier, or executed only once at the deactivation of the Step: P0 Qualifier**. Instructions are written using the ST, IL or LD syntax.

The following shows the results of a pulse Action with the P1 Qualifier:



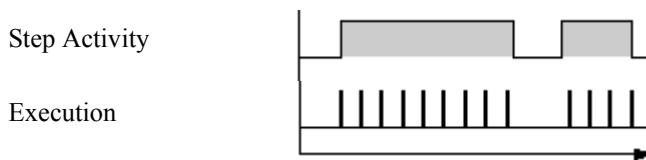
Example



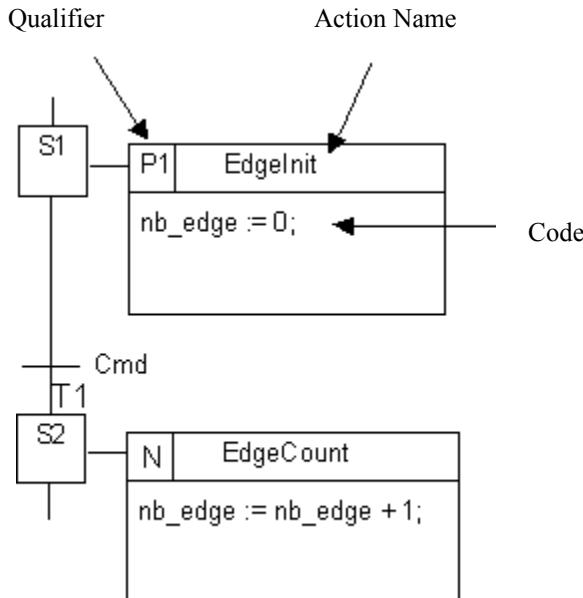
Non-stored Actions

A non-stored (normal) action is a list of ST, IL or LD instructions which are executed **at each** cycle during the whole **active** period of the step. Instructions are written according to the used language syntax. Non-stored actions have the "N" qualifier.

The following is the results of a non-stored Action:



Example



SFC Actions

An SFC action is a child SFC sequence, started or killed according to the change of the step activity signal. An SFC action can have the **N** (Non stored), **S** (Set), or **R** (Reset) Qualifier. This is the meaning of the actions on SFC child:

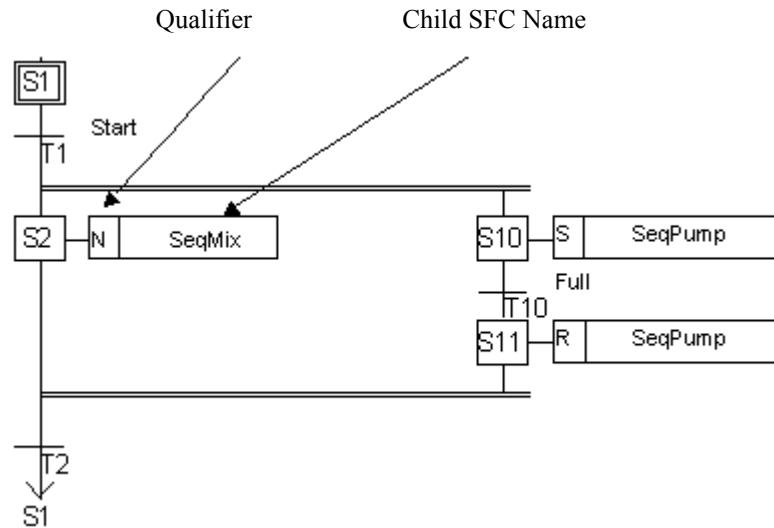
- | | |
|---------------------|---|
| N on a child | starts the child sequence when the Step becomes active, and kills the child sequence when the Step becomes inactive |
| S on a child | starts the child sequence when the Step becomes active. nothing is done when the Step becomes inactive |
| R on a child | kills the child sequence when the Step becomes active. nothing is done when the Step becomes inactive |

The SFC sequence specified as an Action must be a **child SFC Program** of the program currently being edited. Note that using the **S** (Set) or **R** (Reset) Qualifiers for an SFC Action has exactly the same effect as using the **GSTART** and **GKILL** statements, programmed in an **ST** pulse Action.

Example

(* SFC Program using SFC Actions *)

The main SFC Program is named **Father**. It has two SFC children, called **SeqMix** and **SeqPump**. The SFC programming of the father SFC Program is:



List of Instructions

Actions corresponding to several operations can be written as a program using ST, IL and LD syntax. Such actions can have N, P0 or P1 qualifiers.

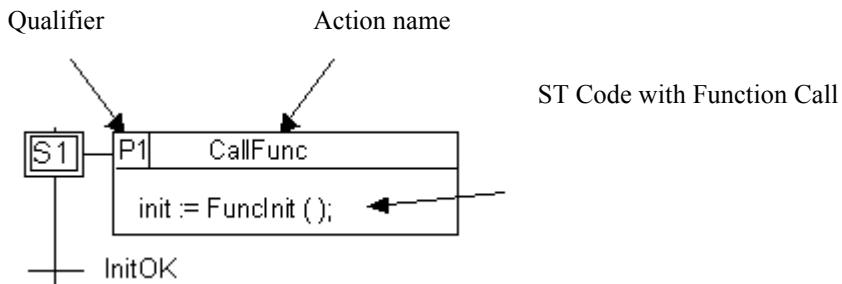
Calling Functions and Function Blocks

Functions (written in ST, LD, or FBD) or Function Blocks (written in SFC, ST, LD, or FBD) or "C" Functions and "C" Function Blocks, can be directly *called from* an SFC action block, based on the syntax of the language used in the action block.

Detailed syntax can be found in the corresponding language section.

Example

(* SFC program with a Function call in an Action Block *)



Conditions Attached to Transitions

At each Transition, a Boolean expression is attached that conditions the clearing of the Transition. The condition is usually expressed with ST or LD language. This is the Level 2 of the Transition. Ways to program a condition:

- Conditions programmed in ST or LD
- Calling Function from a Transition

Warning: When no expression is attached to the Transition, the default condition is **TRUE**.

Condition Programmed in ST

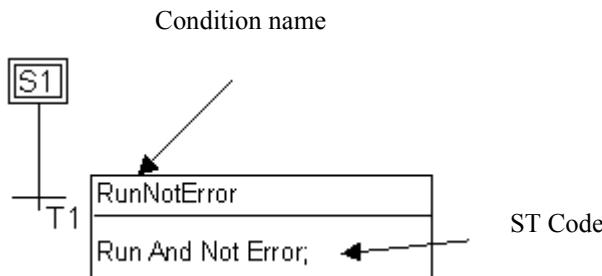
The Structured Text (ST) language can be used to describe the condition attached to a Transition. The complete expression must have Boolean type and may be terminated by a semi colon, according to the following syntax:

<boolean_expression>;

The expression may be a TRUE or FALSE constant expression, a single input or an internal Boolean Variable, or a combination of Variables that leads to a Boolean value.

Example

(* SFC Program with ST programming for Transitions *)

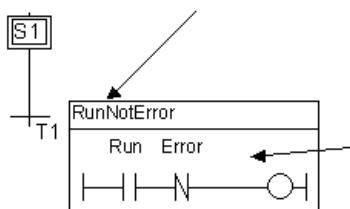


Condition Programmed in LD

The Ladder Diagram (LD) language can be used to describe the condition attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transition's value.

Example

(* SFC Program with LD programming for transitions *)



Condition Programmed in IL

Instruction List (IL) programming may be directly used to describe an SFC **condition**, according to the following syntax:

<IL instruction>

<IL instruction>

....

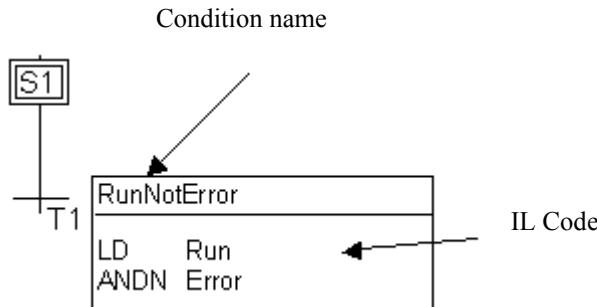
The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the Transition:

current result = 0 or FALSE -> **condition is FALSE**

current result <> 0 or TRUE -> **condition is TRUE**

Example

(* SFC program with a condition programmed in IL *)



Calling Functions from a Transition

Any Function (written in ST, LD, or FBD), or a "C" Function can be called to evaluate the condition attached to a **Transition**, according to the following syntax in ST:

<function> () ;

The value returned by the Function must be Boolean and yields the resulting condition:

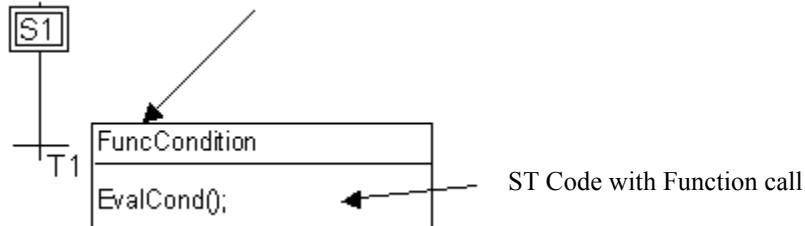
return value = **FALSE** -> condition is **FALSE**

return value = **TRUE** -> condition is **TRUE**

Example

(* SFC program with function call for transitions *)

Condition name



Note: The syntax of Function call for LD and IL is given in the corresponding language section.

Calling Function Blocks from a Transition

It is not recommended to call a Function Block in an SFC condition for the following reasons:

- A Function Block should be called at each Cycle, typically in a cyclic Program. For example, counting blocks make incremental operation at each Cycle, Trigger Blocks need to store value of a Boolean at each Cycle to test rising or falling edges...
- An SFC condition is evaluated only when all its preceding Steps are active (not at each Cycle)

SFC Dynamic Behavior

The dynamic behaviors of the SFC language are:

Initial situation

The **Initial Situation** is characterized by the **Initial Steps** which are, by definition, in the active state at the beginning of the operation. **At least one** Initial Step must be present in each SFC Program.

Clearing of a transition

A Transition is either **enabled** or **disabled**. It is said to be enabled when all immediately preceding Steps linked to its corresponding Transition symbol are **active**, otherwise it is disabled. A Transition cannot be **cleared** unless:

- it is enabled, and
- the associated Transition condition is TRUE

Changing of state of Active Steps

The clearing of a Transition simultaneously leads to the **active** state of the immediately following Steps and to the inactive state of the immediately preceding Steps.

Simultaneous clearing of Transitions

All Transitions (of all SFC Programs) that can be cleared (enabled and condition to true), are cleared simultaneously.

SFC Program Hierarchy

The system enables the description of the vertical structure of SFC Programs. SFC Programs are organized in a **hierarchy tree**. Each SFC Program can control (start, kill...) other SFC Programs. Such Programs are called **children** of the SFC Program which controls them. SFC Programs are linked together into a main **hierarchy tree**, using a "father - child" relationship:

Father Program



The basic rules implied by the hierarchy structure are:

- SFC Programs which have no father are called "main" SFC Programs
- Main SFC Programs are activated by the system when the application starts
- A Program can have several child Programs
- A child of a Program cannot have more than one father
- A child Program can only be controlled by its father
- A Program cannot control the children of one of its own children

The basic actions that a father SFC Program can take to control its child Program are:

Start (GSTART)

Starts the child Program: activates each of its Initial Steps.
Children of this child Program are not automatically started.

Kill (GKILL)

Kills the child Program by deactivating each of its active Steps.
All the children of the child Program are also killed.

Freeze (GFREEZE)

Deactivates each of the active Steps of the Program, and
memorizes them so the program can be restarted. All the
children of the child Program are also frozen.

Restart (GRST)

Restarts a frozen SFC Program by reactivating all the suspended Steps. Children of the Program are not automatically restarted.

Get status (GSTATUS)

Gets the current status (active, inactive or frozen) of a child Program.

Refer to "SFC Actions" or to the ST sub-sections "GSTART" "GKILL" "GFREEZE" "GRST" and "GSTATUS" for more details.

FC Language

Flow Chart (FC) is a graphic language used to describe **sequential operations**. A Flow Chart diagram is composed of **actions** and **tests**.

Between actions and test are **oriented links** representing data flow.

Actions and tests can be described with ST, LD or IL languages. Functions and Function blocks of any language (except SFC) can be called from actions and tests.

A Flow Chart program can call another Flow Chart program. The called FC program is a **sub-program** of the calling FC program.

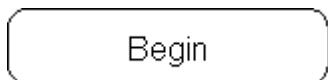
FC Basic Components

The basic components of the Flow Chart language are:

- beginning of chart
- ending of chart
- actions
- tests (conditions)
- oriented links and connectors

FC BEGIN

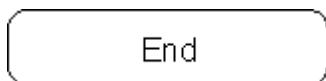
A "**Begin**" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated. Below is the drawing of a "Begin" symbol:



The "Begin" symbol always has a connection (on the bottom) to the other objects of the chart. A flow chart is not valid if no connection is drawn from "Begin" to another object.

FC END

An "**End**" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It represents the final state of the chart, when its execution has been completed. Below is the drawing of an "End" symbol:



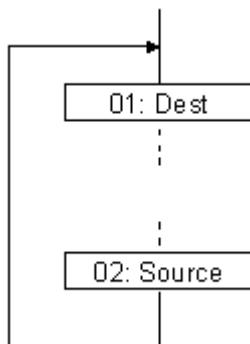
The "End" symbol generally has a connection (on the top) to the other objects of the chart. A flow chart may have no connection to the "End" object (always looping chart). The "End" object is still visible at the bottom of the chart in this case.

FC Flow Links

A flow link is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Below is the drawing of a flow link:



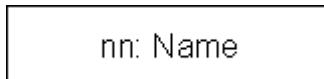
Example



Two links cannot be connected to the same source connection point.

FC Actions

An **action** symbol represents actions to be performed. An action is identified by a number and a name. Below is the drawing of an "Action" symbol:



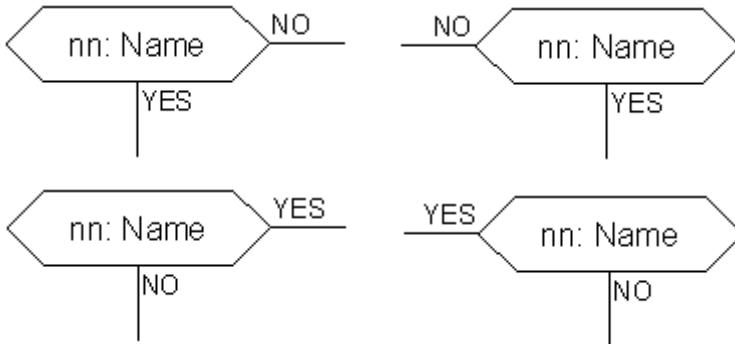
Two different objects of the same chart cannot have the same name or logical number.

Programming language for an action can be ST, LD or IL.

An action is always connected with links, one arriving to it, one starting from it.

FC Conditions

A **Condition** represents a Boolean **test**. A Condition is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. Below are the possible drawings for a Condition symbol:



Two different objects of the same chart cannot have the same name or logical number.

The programming of a test is either:

- an expression in ST, or
- a single rung in LD, with no symbol attached to the unique Coil, or
- several instructions in IL. The IL register (or current result) is used to evaluate the Condition.

When programmed in ST text, the expression may optionally be followed by a semi-colon.

When programmed in LD, the unique coil represents the condition value.

A condition equal to:

- 0 or FALSE directs the flow to NO
- 1 or TRUE directs the flow to YES

A test is always connected with an arriving link, and both forward connections must be defined.

Other FC Components

In addition to basic components, more complex flow charts are built using **FC sub-programs**.

You can also use Connectors instead of flow links. This leads to more readable charts, when too many flow links "cross" many elements.

FC Sub-Program

The system enables the description of the vertical structure of FC programs. FC programs are organized in a **hierarchy tree**. Each FC program can call other FC programs. Such a program is called a **child program** of the FC program which calls them. FC programs which call FC sub-programs are called **father programs**. FC programs are linked together into a main hierarchy tree, using a "Father - Child" relation:

Father Program



Child Program

A **sub-program** symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete. A Flow Chart sub-program is identified by a number and a name, as other programs, Functions or Function Blocks. Below is the drawing of a "Sub-Program call" symbol:



Two different objects of the same chart cannot have the same logical number.

The basic rules implied by the FC hierarchy structure are:

- FC programs which have no father are called **main FC programs**.
- Main FC programs are activated by the system when the application starts

- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can be called only by its father
- A program cannot call the children of one of its own children

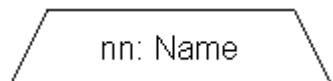
The same sub-program may appear several times in the father chart.

A Flow Chart sub-program call represents the complete execution of the sub chart. The father chart execution is suspended during the child chart is performed.

The sub-program calling Blocks must follow the same Connection rules as the ones defined for an action.

FC I/O Specific Actions

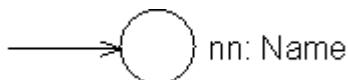
An **I/O specific action** symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. The drawing of an "I/O Specific Action" symbol is:



I/O specific actions have exactly the same behavior as standard actions. This covers their properties, ST, LD or IL programming, and connection rules.

FC Connectors

Connectors are used to represent a link between two points of the diagram without drawing it. A Connector is represented as a circle and is connected to the source of the flow. The drawing of the Connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). Below is the standard drawing of a connector:



A Connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

FC Comments

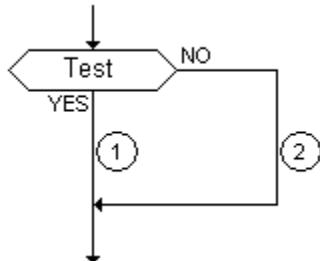
A **Comment** Block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on an unused space of the Flow Chart document window, and is used to document the program. Below is the drawing of a "Comment" symbol:



FC Complex Structure Examples

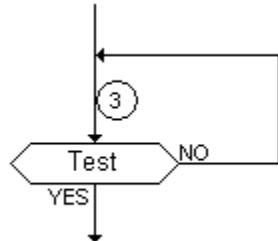
This section shows **Complex Structure** examples that can be defined in a Flow Chart diagram. Such Structures are combinations of basic objects linked together.

IF / THEN / ELSE



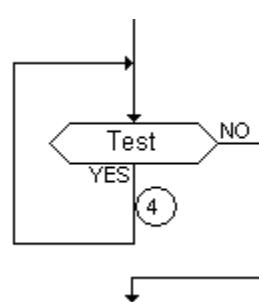
- (1) place for "THEN" actions to be inserted
- (2) place for "ELSE" actions to be inserted

REPEAT / UNTIL



- (3) place for repeated actions to be inserted

WHILE / DO



- (4) place for repeated actions to be inserted

FC Dynamic Behavior

The execution of a Flow Chart diagram can be explained as follows:

- The Begin symbol takes one Target Cycle
- The End symbol takes one Target Cycle and ends the execution of the chart. After this symbol is reached, no more actions of the chart are executed.
- The flow is broken each time an item (action, decision) is encountered that has already been reached in the same Cycle. In such a case the flow will continue on the next Cycle.

Note: Contrary to SFC, an action is not a stable state.

FC Checking

Apart from attached ST, LD, or IL programming, some other syntactic rules apply to Flow Chart itself. The following are the main rules:

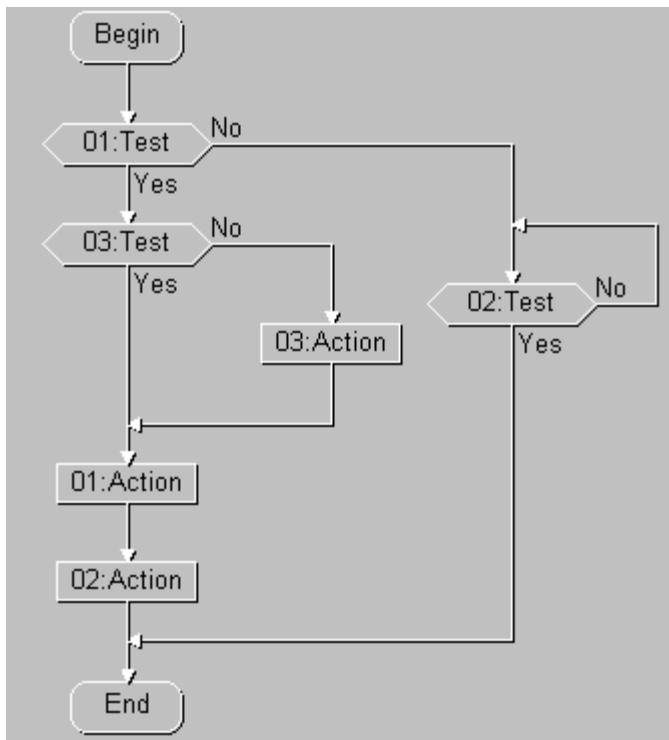
- All "connection" points of all symbols must be wired (connection to "End" symbol may be omitted)
- All symbols must be linked together (no isolated part should appear)
- All connectors should have valid destinations

FC Examples

Two examples of Flow Chart are provided.

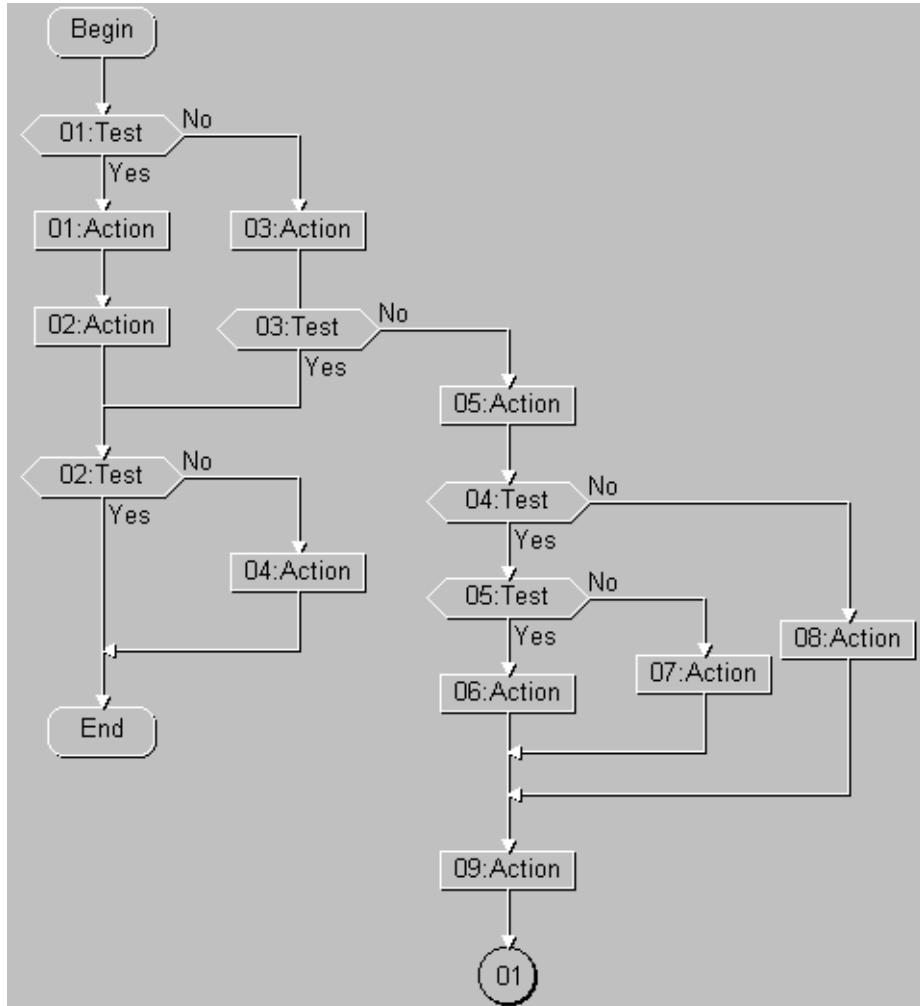
A structured chart using IF/THEN/ELSE and REPEAT/UNTIL structures

This first example shows a structured chart using IF/THEN/ELSE and REPEAT/UNTIL Structures:



A non-structured chart using a Connector

This example shows a non-structured chart using a Connector. The use of Connectors in such a case avoid the drawing of very long links that could be hard to follow in the case of a large chart, when source and destination of a link cannot be visible together on the screen:

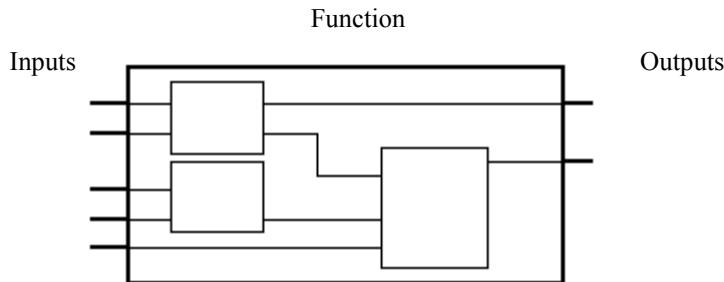


FBD Language

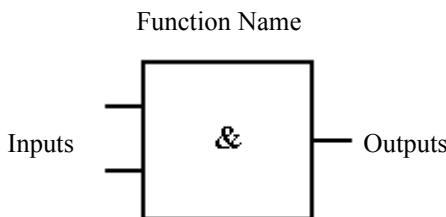
The **Functional Block Diagram (FBD)** is a graphic language. It allows the programmer to build complex procedures by taking existing functions from the standard library or from the function or function block section.

FBD Diagram Main Format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. An output of a block may also be connected to an input of another block.



An entire function operated by an FBD program is built with standard elementary **blocks** from the standard library or from the function or function block section. Each block has a fixed number of input connection points and a fixed number of output connection points. A block is represented by a single rectangle. The inputs are connected on its left border. The outputs are connected on its right border. An elementary block performs a single function between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well defined type.



Input variables of an FBD program must be connected to input connection points of blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a Constant Expression, any internal or input variable, or an output variable. For information on constant expressions, see page 381.

Output variables of an FBD program must be connected to output connection points of blocks. The type of each variable must be the same as the type expected for the associated block output. An output for FBD diagram can be any internal or output variable, or the name of the Function (for functions only). When an output is the name of the currently edited function, it represents the assignment of the return value for the function (returned to the calling program).

Input and output variables, inputs and outputs of the blocks are wired together with connection lines, or **links**. Single lines may be used to connect two logical points of the diagram:

- An input variable and an input of a block
- An output of a block and an input of another block
- An output of a block and an output variable

For information on variables, see page 389.

The connection is oriented, meaning that the line carries associated data from the left end to the right end. The left and right ends of the connection line must be of the same data type.

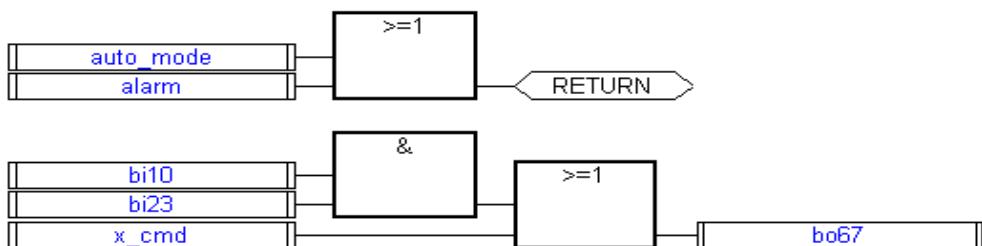
Multiple right connection, also called **divergence** can be used to broadcast an information from its left end to each of its right ends. All the ends of the connection must be of the same data type.

For information on data types, see page 377.

RETURN Statement

The "<RETURN>" keyword may occur as a diagram output. It must be connected to a Boolean output connection point of a block. The RETURN statement represents a **Conditional End** of the program: if the output of the box connected to the statement has the Boolean value **TRUE**, the end (remaining part) of the diagram is not executed.

Example



(* ST equivalence: *)

```
If auto_mode OR alarm Then
Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

Jumps and Labels

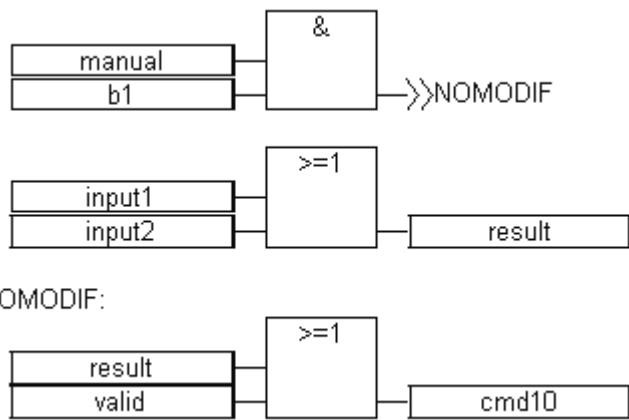
Labels and **jumps** are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notation is used:

>>LAB Jump to a label (label name is "LAB")

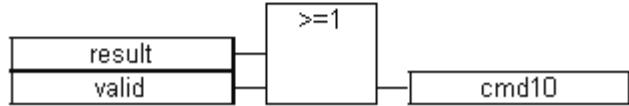
LAB: Definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the Boolean state **TRUE**, the execution of the program directly jumps to after the corresponding label symbol.

Example



NOMODIF:



(* IL Equivalence: *)

| | |
|----------|-----------|
| ld | manual |
| and | b1 |
| jmpc | NOMODIF |
| ld | input1 |
| or | input2 |
| st | result |
| NOMODIF: | ld result |
| | or valid |
| | st cmd10 |

Boolean Negation

A single connection line with its right end connected to an input of a block can be terminated by a **Boolean negation**. The negation is represented by a small circle. When a Boolean negation is used, the left and right ends of the connection line must have the **BOOL** type.

Example

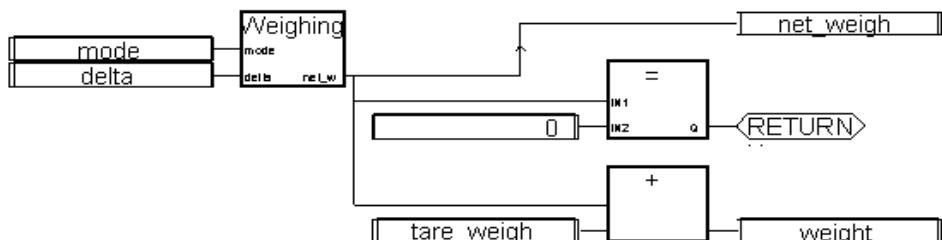


Calling Functions and Function Blocks

The FBD language enables the **calling of functions or function blocks**. A Function or Function Block is represented by a box. The name written in the box is the name of the function or function blocks.

In the case of a function, the return value is the only output from the box. Function blocks can have more than one output.

Example

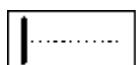


(* ST Equivalence – in ST, we have to define an intermediate variable: net_weight *)

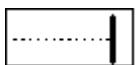
```
net_weight := Weighing (mode, delta); (* call function *)
If (net_weight = 0) Then Return; End_if;
weight := net_weight + tare_weight;
```

LD Language

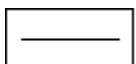
Ladder Diagram (LD) is a graphic representation of Boolean equations, combining **Contacts** (input arguments) with **Coils** (output results). The LD language enables the description of tests and modifications of **Boolean** data by placing graphic symbols into the program chart. LD graphic symbols are organized within the chart exactly as an electric Contact diagram. LD diagrams are connected on the left side and on the right side to vertical **Power Rails**. These are the basic graphic components of an LD diagram:



Left vertical power rail



Right vertical power rail



Horizontal connection line



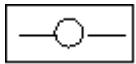
Vertical connection line



Multiple connection lines (all connected together)



Contact associated with a variable

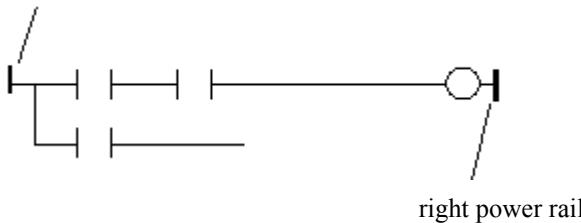


Coil associated to an output or to an internal variable

Power Rails and Connection Lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.

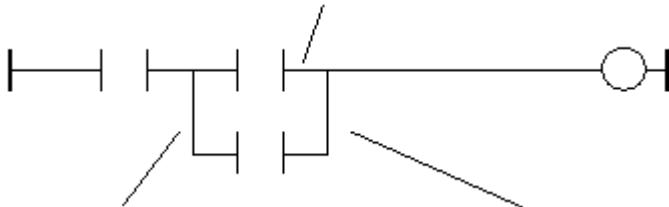
left power rail



right power rail

LD diagram graphic symbols are connected to power rails or to other symbols by connection lines, or **links**. Connection lines are horizontal or vertical.

horizontal connection lines



vertical connection lines

vertical connection lines with OR meaning

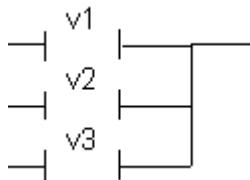
Each line segment has a boolean state **FALSE** or **TRUE**. The Boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left *vertical power rail* has the **TRUE** state.

Multiple Connections

The Boolean state given to a single horizontal connection line is the same on the left and on the right ends of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The Boolean state of the ends of a multiple connection follows logic rules.

A *multiple connection on the left* combines *more than one* horizontal lines connected on the *left* side of a vertical line, and *one* line connected on its *right* side. The Boolean state of the right end is the LOGICAL OR between all the left extremities.

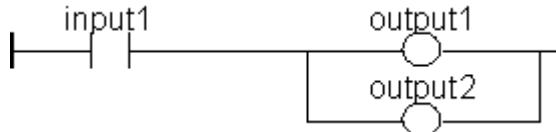
(* Example of multiple LEFT connection *)



(* right end state is $(v1 \text{ OR } v2 \text{ OR } v3)$ *)

A **multiple connection on the right** combines *one* horizontal line connected on the *left* side of a vertical line, and *more than one* line connected on its *right* side. The Boolean state of the left end is propagated into each of the right ends.

(* Example of multiple RIGHT connection *)

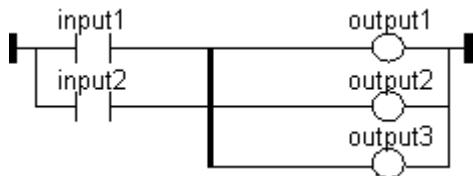


(* ST equivalence: *)

```
output1 := input1;  
output2 := input1;
```

A *multiple connection on the left and on the right* combines *more than one* horizontal line connected on the **left** side of a vertical line, and *more than one* line connected on its right side. The Boolean state of each of the right ends is the LOGICAL OR between all the left ends.

(* Example of multiple LEFT and RIGHT connection *)



(* ST Equivalence: *)

```
output1 := input1 OR input2;  
output2 := input1 OR input2;  
output3 := input1 OR input2;
```

Basic LD Contacts and Coils

Several symbols are available for input **contacts**:

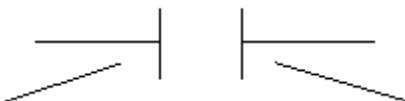
- Direct Contact
- Inverted Contact
- Contact with Rising Edge Detection
- Contact with Falling Edge Detection

Several symbols are available for output **coils**:

- Direct Coil
- Inverted Coil
- SET Coil
- RESET Coil
- Coil with Rising Edge Detection
- Coil with Falling Edge Detection

The name of the variable is written above any of these graphic symbols:

Name of the associated Boolean variable

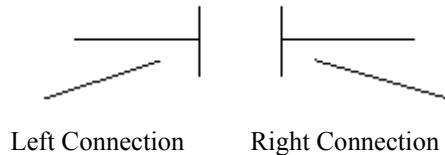


Left Connection

Right Connection

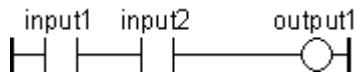
Direct Contact

A **Direct Contact** enables a *Boolean operation* between a *connection line* state and a *Boolean variable*.



The state of the connection line on the right of the Contact is the Logical AND between the state of the left connection line and the value of the variable associated with the Contact.

Example

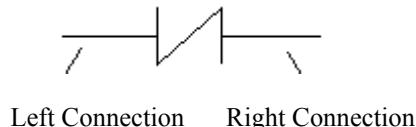


(* ST Equivalence: *)

```
output1 := input1 AND input2;
```

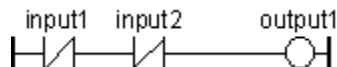
Inverted Contact

An **Inverted Contact** enables a *Boolean operation* between a *connection line* state and the Boolean negation of a *Boolean variable*.



The state of the connection line on the right of the Contact is the Logical AND between the state of the left connection line and the *Boolean negation* of the value of the variable associated with the Contact.

Example

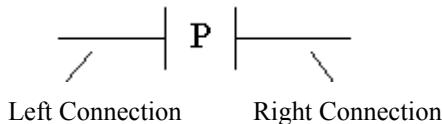


(* ST Equivalence: *)

output1 := NOT (input1) AND NOT (input2);

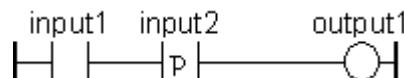
Contact with Rising Edge Detection

A Contact with rising edge detection (positive) enables a Boolean operation between a connection line state and the rising edge of a Boolean variable.



The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. It is reset to FALSE in all other cases.

Example



```
(* ST Equivalence: *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)
```

Contact with Falling Edge Detection

A Contact with **Falling Edge** detection (negative) enables a *Boolean operation* between a *connection line* state and the falling edge of a *Boolean variable*.

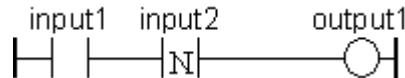


Left Connection

Right Connection

The state of the connection line on the right of the Contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable *falls* from **TRUE** to **FALSE**. It is reset to **FALSE** in all other cases.

Example

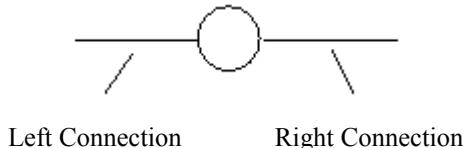


(* ST Equivalence: *)

```
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)
```

Direct Coil

Direct Coils enable a Boolean output of a connection line Boolean state.

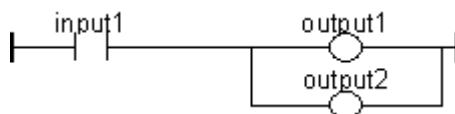


The associated variable is assigned with the Boolean *state of the left connection*. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical Power Rail. For information on variables, see page 389.

The associated Boolean variable must be OUTPUT or MEMORY.

The associated name can be the name of the program (for Function only). This corresponds to the assignment of the return value of the function.

Example

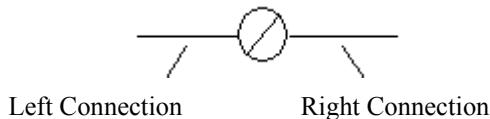


(* ST Equivalence: *)

```
output1 := input1;  
output2 := input1;
```

Inverted Coil

Inverted Coils enable a Boolean output according to the Boolean negation of a connection line state.



The associated variable is assigned with the Boolean *negation* of the *state of the left connection*. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail. For information on variables, see page 389.

The associated Boolean variable must be OUTPUT or MEMORY.

The associated name can be the name of the program (for Function only). This corresponds to the assignment of the return value of the function.

Example

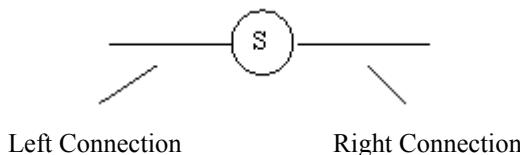


(* ST Equivalence: *)

```
output1 := NOT (input1);  
output2 := input1;
```

SET Coil

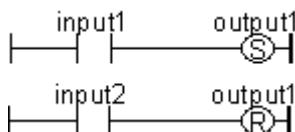
"Set" Coils enable a Boolean output of a connection line Boolean state.



The associated variable is SET TO TRUE when the boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. For information on variables, see page 389. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated Boolean variable must be OUTPUT or MEMORY.

Example

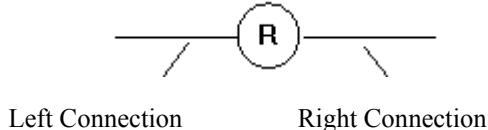


(* ST Equivalence: *)

```
IF input1 THEN
    output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END_IF;
```

RESET Coil

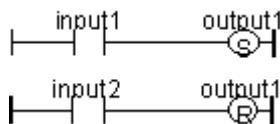
"Reset" Coils enable Boolean output of a connection line Boolean state.



The associated variable is RESET TO FALSE when the Boolean *state of the left connection* becomes **TRUE**. The output variable keeps this value until an inverse order is made by a "SET" coil. For information on variables, see page 389. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical Power Rail.

The associated Boolean variable must be OUTPUT or MEMORY.

Example

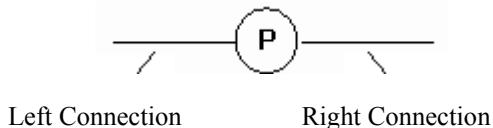


(* ST Equivalence: *)

```
IF input1 THEN
    output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END_IF;
```

Coil with Rising Edge Detection

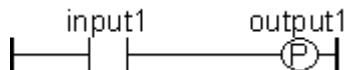
Coils with rising edge detection or "Positive" coils enable Boolean output of a connection line Boolean state.



The associated variable is set to **TRUE** when the Boolean *state of the left connection* rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. For information on variables, see page 389. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated Boolean variable must be OUTPUT or MEMORY.

Example



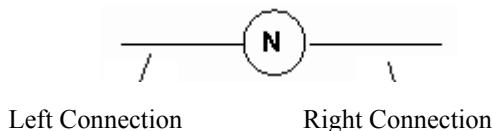
(* ST Equivalence: *)

```
IF (input1 and NOT(input1prev)) THEN
    output1 := TRUE;
ELSE
    output1 := FALSE;
END_IF;
```

(* input1prev is the value of input1 at the previous cycle *)

Coil with Falling Edge Detection

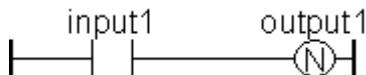
Coils with falling edge detection or "Negative" coils enable Boolean output of a connection line Boolean state.



The associated variable is set to **TRUE** when the Boolean *state of the left connection* falls from **TRUE** to **FALSE**. The output variable resets to **FALSE** in all other cases. For information on variables, see page 389. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated Boolean variable must be OUTPUT or MEMORY.

Example



(* ST Equivalence: *)

```
IF (NOT(input1) and input1prev) THEN
    output1 := TRUE;
ELSE
    output1 := FALSE;
END_IF;
```

(* input1prev is the value of input1 at the previous cycle *)

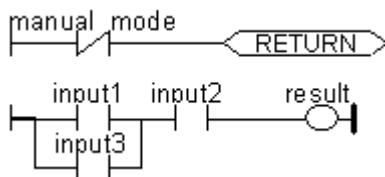
RETURN Statement

The **RETURN** label can be used as an output to represent a conditional end of the program. No connection can be put on the right of a RETURN symbol.

If the *left connection* line has the **TRUE** Boolean state, the program ends without executing the equations entered on the following lines of the diagram.

When the LD program is a function, its name has to be associated with an output coil to set the return value (returned to the calling program).

Example



(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;  
result := (input1 OR input3) AND input2;
```

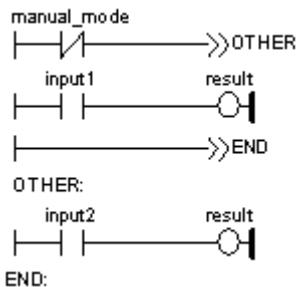
Jumps and Labels

Labels, conditional and unconditional jumps symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB jump to label named "LAB"
LAB: definition of the label named "LAB"

If the *connection on the left* of the jump symbol has the TRUE Boolean state, the program execution is driven after the label symbol.

Example



(* IL Equivalence: *)

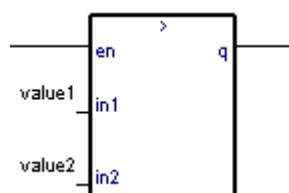
```
ldn    manual_mode
jmpc  OTHER
ld    input1
st    result
jmp   END
OTHER: ld    input2
st    result
END:   (* end of program *)
```

BLOCKS in LD

Using the LD editor, you connect function boxes to Boolean lines. A function can actually be an operator, a function block or a function. As all blocks do not have always a Boolean input and/or a Boolean output, inserting blocks in an LD diagram leads to the addition of new parameters **EN**, **ENO** to the block interface.

The "EN" input

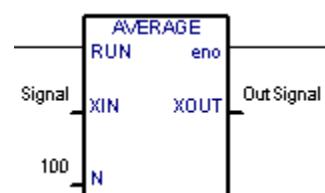
On some operators, functions or function blocks, the first input does not have Boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "EN". The block is executed only if the EN input is TRUE. Below is the example of a comparison operator, and the equivalent code expressed in ST:



```
IF rung_state THEN
    q := (value1 > value 2);
ELSE
    q := FALSE;
END_IF;
(* continue rung with q state *)
```

The "ENO" output

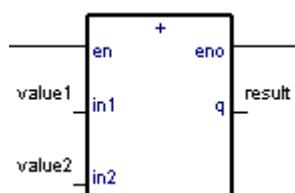
On some operators, functions or function blocks, the first output does not have Boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "ENO". The ENO output always takes the same state as the first input of the block. Below is an example with AVERAGE function block, and the equivalent code expressed in ST:



```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```

The "EN" and "ENO" parameters

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in ST:



```
IF rung_state THEN  
    result := (value1 + value2);  
END_IF;  
eno := rung_state;  
(* continue rung with eno state *)
```

ST Language

ST (Structured Text) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST language can be used for the description of the actions within the Steps and conditions attached to the Transitions of the SFC or the Actions and Tests of the FC Language.

ST Main Syntax

An ST program is a list of ST **statements**. Each statement ends with a semi-colon (";") separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with **inactive separators** (space character, end of line or tab stops) or by **active separators**, which have a well defined significance (for example, the ">" separator indicates a "greater than" comparison. **Comments** may be freely inserted into the text. A comment must begin with "(*)" and ends with "*)". These are basic types of ST statements:

- assignment statement (variable := expression;)
- function call
- function block call
- selection statements (IF, THEN, ELSE, CASE...)
- iteration statements (FOR, WHILE, REPEAT...)
- control statements (RETURN, EXIT...)
- special statements for links with other languages such as **SFC**

When entering ST syntax, basic coding is black while other items are displayed using color:

- Keywords are pink
- Numbers are brown
- Comments are green

Inactive separators may be freely entered between active separators, constant expressions and identifiers. ST inactive separators are: *Space* (blank) character, *Tabs* and *End of line* character. Unlike line-formatted languages such as IL, end of lines may be entered anywhere in the program. The rules shown below should be followed when using inactive separators to increase ST program readability:

- Do not write more than one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase readability of lines or paragraphs

Example

Low Readability

```
imax := max_ite; cond := X12;
if not(cond (* alarm *))
then return; end_if;
for i (* index *) := 1 to max_ite
do if i <> 2 then Spcall();
end_if; end_for;
(* no effect if alarm *)
```

High Readability

```
(* imax : number of iterations *)
(* i: FOR statement index *)
(* cond: process validity *)
imax := max_ite;
cond := X12;
if not (cond) then
return;
end_if;

(* process loop *)

for i := 1 to max_ite do
if i <> 2 then
Spcall ();
end_if;
end_for;
```

Expressions and Parentheses

ST expressions combine ST *operators* and variable or constant *operands*. For each single expression (combining operands with one ST operator), the type of the operands must be the same. This single expression has the same data type as its operands, and can be used in a more complex expression. For example:

| | |
|-------------------------|--------------------------|
| (boo_var1 AND boo_var2) | has BOOL type |
| not (boo_var1) | has BOOL type |
| (sin (3.14) + 0.72) | has REAL type |
| (t#1s23 + 1.78) | is an invalid expression |

For information on data types, see page 377.

Parentheses are used to isolate sub parts of the expression, and to explicitly order the priority of the operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default *priority* between ST operators. For example:

| | | |
|-------------|----------------|---|
| 2 + 3 * 6 | equals 2+18=20 | because multiplication operator has a higher priority |
| (2 + 3) * 6 | equals 5*6=30 | priority is given by parenthesis |

Functions or Function Block Calls

Standard ST function calls may be used for each of following objects:

- Functions and function blocks written in IEC 61131 languages
- "C" functions and function blocks

Calling Functions

Calling Functions from ST:

Name: name of the called function written in IEC 61131 language or in "C"

Meaning: calls a ST, IL, LD or FBD Functions or a "C" function and gets its return value

Syntax: <variable> := <funct> (<par1>, ... <parN>);

Operands: The type of return value and calling parameters must follow the interface defined for the function.

Return value: value returned by the function

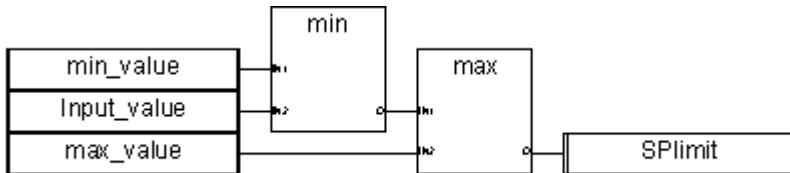
Function calls may be used in any expression.

Example

Example1: IEC 61131 function call

```
(* Main ST program *)
(* gets an integer value and converts it into a limited time value *)
ana_timeprog := SPlimit ( tprog_cmd );
appl_timer := ANY_TO_TIME (ana_timeprog * 100);
```

```
(* Called FBD function named 'SPlimit' *)
```



Example2: "C" function call – same syntax as for IEC 61131 function calls

```
(* Functions used in complex expressions: min, max, right, mlen and  
left are standard "C" functions *)  
limited_value := min (16, max (0, input_value) );  
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

Calling Function Blocks

Calling Function Blocks from ST:

Name: name of the function block instance

Meaning: calls a function block from the standard library or from the user's library and accesses its return parameters

Syntax: (* call of the function block *)
<blockname> (<p1>, <p2> ...);
(* gets its return parameters *)
<result> := <blockname>. <ret_param1>;
...
<result> := <blockname>. <ret_paramN>;

Operands: parameters are expressions which match the type of the parameters specified for that function block

Return value: See Syntax to get the return parameters.

Consult the 'Standard Function Blocks' section to find the meaning and type of each function block parameter. The function block instance (name of the copy) must be declared in the dictionary

Example

```
(* ST program calling a function block *)
(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)
(* Function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;
```

ST Operators

Standard operators such as AND, NOT, OR, XOR, etc. are described in the Standard Operators section.

ST Basic Statements

Assignment

Name: :=

Meaning: Assigns a variable to an expression

Syntax: <variable> := <any_expression>;

Operands: Variable must be an internal or output variable and the expression must have the same type

The expression can be a call to a function.

Example

```
(* ST program with assignments *)
(* variable <= variable *)
bo23 := bo10;

(* Variable <= expression *)

bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

RETURN Statement

Name: RETURN

Meaning: terminates the execution of the current program

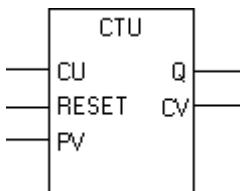
Syntax: RETURN ;

Operands: (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

Example

(* FBD specification of the program: programmable counter *)



(* ST implementation of the program, using RETURN statement *)

```
If NOT (CU) then
    Q := false;
    CV := 0;
    RETURN; (* terminates the program *)
end_if;

if RESET then
    CV := 0;
else
    if (CV < PV) then
        CV := CV + 1;
    end_if;
end_if;
Q := (CV >= PV);
```

IF-THEN-ELSIF-ELSE Statement

Name: **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**

Meaning: executes one of several lists of ST statements
selection is made according to the value of a Boolean expression

Syntax:

```
IF <Boolean_expression> THEN
    <statement>;
    <statement>;
    ...
    ELSIF <Boolean_expression> THEN
        <statement>;
        <statement>;
        ...
    ELSE
        <statement>;
        <statement>;
        ...
    END_IF;
```

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE. The ELSIF statement may be used more than once. The ELSE statement, if used, must appear only once at the end of the 'IF, ELSIF...' sequence.

Example

```
(* ST program using IF statement *)

IF manual AND not (alarm) THEN
    level := manual_level;
    bx126 := bi12 OR bi45;
ELSIF over_mode THEN
    level := max_level;
ELSE
    level := (lv16 * 100) / scale;
END_IF;
```

```
(* IF structure without ELSE *)
If overflow THEN
    alarm_level := true;
END_IF;
```

CASE Statement

Name: CASE ... OF ... ELSE ... END_CASE

Meaning: executes one of several lists of ST statements
selection is made according to an integer expression

Syntax: CASE <integer_expression> OF
 <value> : <statements> ;
 <value>, <value> : <statements> ;
 ...
 ELSE
 <statements> ;
 END_CASE;

Case values must be integer constant expressions. Several values, separated by commas, can lead to the same list of statements. The ELSE statement is optional.

Example

```
(* ST program using CASE statement *)

CASE error_code OF
    255: err_msg := 'Division by zero';
    fatal_error := TRUE;
    1: err_msg := 'Overflow';
    2, 3: err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
END_CASE;
```

WHILE Statement

Name: **WHILE ... DO ... END_WHILE**

Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated BEFORE any iteration

Syntax: **WHILE <Boolean_expression> DO**
 <statement>;
 <statement>;
 ...
END WHILE ;

Warning: Since the virtual machine is a **synchronous** system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

Example

```
(* ST program using WHILE statement *)
(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END WHILE;
```

REPEAT Statement

Name: REPEAT ... UNTIL ... END_REPEAT

Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration

Syntax: REPEAT

```
<statement>;  
<statement>;
```

...

```
UNTIL <Boolean_condition>  
END_REPEAT ;
```

Warning: Because the virtual machine is a **synchronous** system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

Example

```
(* ST program using REPEAT statement *)  
  
(* this program uses specific "C" functions to read characters *)  
(* on a serial port *)  
  
string := ''; (* empty string *)  
nbchar := 0;  
IF ComIsReady ( ) THEN  
    REPEAT  
        string := string + ComGetChar ( );  
        nbchar := nbchar + 1;  
    UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )  
    END_REPEAT;  
END_IF;
```

FOR Statement

Name: **FOR ... TO ... BY ... DO ... END_FOR**

Meaning: executes a limited number of iterations, using an integer index variable

Syntax: **FOR <index> := <mini> TO <maxi> BY <step> DO**
 <statement>;
 <statement>;
END_FOR;

Operands: **index:** internal integer variable increased at each loop
mini: initial value for index (before first loop)
maxi: maximum allowed value for index
step: index increment at each loop

The [BY step] statement is optional. If not specified, the increment step is 1

Warning: Because the virtual machine is a **synchronous** system, input variables are not refreshed during FOR iterations.

This is the "WHILE" equivalent of a FOR statement:

```
index := mini;
while (index <= maxi) do
    <statement>;
    <statement>;
    index := index + step;
end_while;
```

Example

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
    END_IF;
END_FOR;
```

EXIT Statement

Name: **EXIT**

Meaning: exit from a FOR, WHILE or REPEAT iteration statement

Syntax: **EXIT;**

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

Example

```
(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
  code := ascii (message, index);
  IF (code = searched_char) THEN
    found := YES;
    EXIT;
  END_IF;
END_FOR;
```

ST Extensions

The following statements and functions are available to control the execution of the SFC child programs. They may be used inside action blocks written in ST in SFC steps.

| | |
|---------|---|
| GSTART | starts an SFC program or function block |
| GKILL | kills an SFC program |
| GFREEZE | freezes an SFC program |
| GRST | restarts a frozen SFC program or function block |
| GSTATUS | gets current status of an SFC program |

Warning: These functions are not in the IEC 61131standard.

Easy equivalents can be found for GSTART and GKILL using the following syntax in the SFC step:

- child_name with the S qualifier (* equivalent to GSTART(child_name); *)
- child_name with the R qualifier (* equivalent to GKILL(child_name); *)

The following fields can be used to access the status of an SFC step or child (from its father):

| | |
|-----------------|--|
| StepName.x | Boolean value that represents the activity of the Step |
| StepName.t | time elapsed since the last activation of the step: activity duration ("StepName" represents the name of the SFC step) |
| ChildName._S1.x | Boolean value that represents the activity of the child |
| ChildName._S1.t | time elapsed since the last activation of the step: activity duration ("ChildName" represents the name of the SFC child) |

GSTART Statement in SFC Action

| | |
|----------------------|---|
| Name: | GSTART |
| Meaning: | Starts an SFC child program or function block by placing a token into each of its initial Steps. The abbreviated syntax is equivalent to an SFC Child action block having the S qualifier. The extended syntax only applies to SFC child function blocks. |
| Syntax: | GSTART (<child_name>); or GSTART (<child_name,step_name,input1,input2,...inputn>) where <i>child_name</i> represents the name of the SFC child POU <i>step_name</i> represents the name of the active step. <i>step_name</i> must be preceded by two underscore characters (e.g., __S1) <i>input1, input2, ..., inputn</i> indicate the values of the input parameters of the SFC child POU |
| Operands: | the specified SFC program must be a child of the one in which the statement is written |
| Return value: | (none) |

Children of the child program are not automatically started by the GSTART statement. For details about SFC actions, see page 407.

Note: Since GSTART is not in the IEC 61131 standard, it is preferable to use the S qualifier attached to the child name.

GKILL Statement in SFC Action

Name: **GKILL**

Meaning: Kills a child SFC program by removing the Tokens currently existing in its Steps. The syntax is equivalent to an SFC Child action block having the R qualifier.

Syntax: **GKILL (<child_name>);**

where

child_name represents the name of the SFC child POU

Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: (none)

Children of the child program are automatically killed with the specified program. For details on SFC actions, see page 407.

Note: Since GKILL is not in the IEC 61131 standard, it is preferable to use the R qualifier attached to the child name.

Example

See GSTART

GFREEZE Statement in SFC Action

Name: **GFREEZE**

Meaning: freezes a child SFC (program or function block); suspends its execution. The suspended SFC POU can then be restarted using the GRST statement.

Syntax: **GFREEZE (<child_name>);**

where

child_name represents the name of the SFC child POU

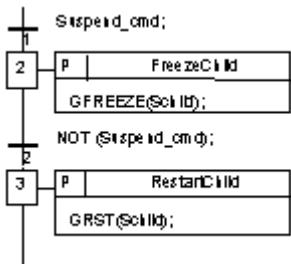
Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: (none)

Children of the child program are automatically frozen along with the specified program.

Note: GFREEZE is not in the IEC 61131 standard.

Example



GRST Statement in SFC Action

Name: **GRST**

Meaning: restarts a child SFC program frozen by the GFREEZE statement: all the Tokens removed by GFREEZE are restored. The extended syntax only applies to SFC child function blocks.

Syntax: **GRST** (*<child_name>*);

or

GRST (*<child_name,input1,input2,...inputn>*);

where

child_name represents the name of the SFC child POU

inputn indicates the value of the input parameter of the SFC child POU

Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: (none)

Children of the child program are automatically restarted by the GRST statement.

GRST is not in the IEC 61131 standard.

GSTATUS Statement in SFC Action

Name: GSTATUS

Meaning: returns the current status of an SFC program

Syntax: <var> := GSTATUS (<child_name>);

where

child_name represents the name of the SFC child POU

Operands: the specified SFC program must be a child of the one in which the statement is written

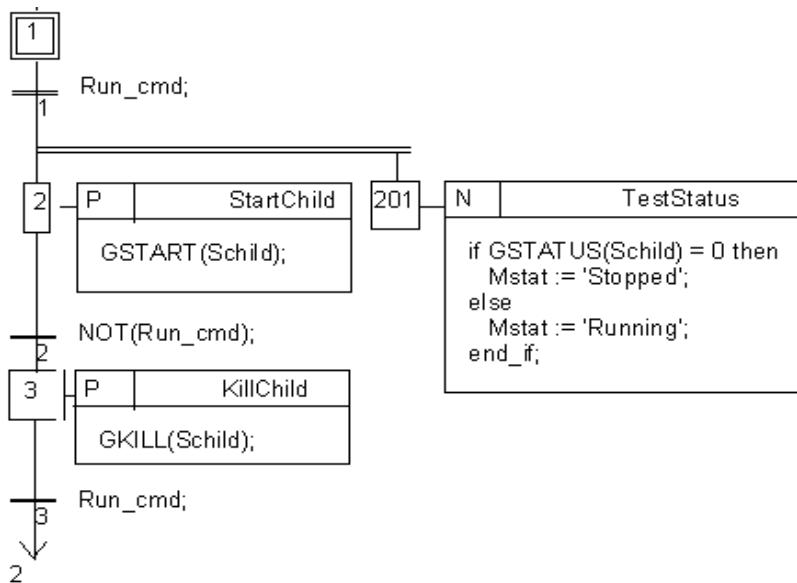
Return value: 0 = Program is inactive (killed)

1 = Program is active (started)

2 = Program is frozen

Note: GSTATUS is not in the IEC 61131 standard.

Example



IL Language

Instruction List, or **IL** is a low level language. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

IL Main Syntax

An IL program is a list of **instructions**. Each instruction must begin on a new line, and must contain an **operator**, completed with optional **modifiers** and, if necessary, for the specific operation, one or more **operands**, separated with commas ','. A **label** followed by a colon (':') may precede the instruction. If a **comment** is attached to the instruction, it must be the last component of the line. Comments always begin with '(*' and ends with '*)'. Empty lines may be entered between instructions. Comments may be placed on empty lines. Furthermore, when entering IL syntax, basic coding is black while other items are displayed using color:

- Keywords are pink
- Numbers are brown
- Comments are green

Example

| Label | Operator | Operand | Comments |
|--------|----------|---------|--------------------------------|
| Start: | LD | IX1 | (* push button *) |
| | ANDN | MX5 | (* command is not forbidden *) |
| | ST | QX2 | (* start motor *) |

Labels

A **label** followed by a colon (':') may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Labels must conform to the following naming rules:

- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters, digits** or '**_**' character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a Variable.

Operator Modifiers

The available operator **modifiers** are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

- | | |
|---|---------------------------------|
| N | Boolean negation of the operand |
| (| delayed operation |
| C | conditional operation |

The '**N**' modifier indicates a Boolean negation of the operand. For example, the instruction **ORN IX12** is interpreted as: **result := result OR NOT (IX12)**.

The parenthesis '(' modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis ')' operator is encountered.

The '**C**' modifier indicates that the attached instruction must be executed only if the current result has the Boolean value TRUE (different than 0 for non-Boolean values). The '**C**' modifier can be combined with the '**N**' modifier to indicate that the instruction must be executed only if the current result has the Boolean value FALSE (or 0 for non-Boolean values).

Delayed Operations

Because there is only one IL register (**current result**), some operations may have to be **delayed**, so that the execution order of the instructions can be changed. Parentheses are used to indicate delayed operations:

| | | |
|-----|----------------|---------------------------------------|
| '(| is a modifier | indicates the operation to be delayed |
| ')' | is an operator | executes the delayed operation |

The opening **parenthesis** '(' modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis ')' operator is encountered.

Example

```
AND (    IX12
OR      IX35
)
```

is interpreted as:

```
result := result AND ( IX12 OR IX35 )
```

IL Operators

The following table summarizes the standard operators of the IL language:

| Operator | Modifier | Operand | Description |
|----------|----------|------------------------------|----------------------------|
| LD | N | Variable, constant | Loads operand |
| ST | N | Variable | Stores current result |
| S | | BOOL variable | Sets to TRUE |
| R | | BOOL variable | Resets to FALSE |
| AND | N (| BOOL | Boolean AND |
| & | N (| BOOL | Boolean AND |
| OR | N (| BOOL | Boolean OR |
| XOR | N (| BOOL | exclusive OR |
| ADD | (| Variable, constant | Addition |
| SUB | (| Variable, constant | Subtraction |
| MUL | (| Variable, constant | Multiplication |
| DIV | (| Variable, constant | Division |
| GT | (| Variable, constant | Test: > |
| GE | (| Variable, constant | Test: >= |
| EQ | (| Variable, constant | Test: = |
| LE | (| Variable, constant | Test <= |
| LT | (| Variable, constant | Test < |
| NE | (| Variable, constant | Test <> |
| CAL | C N | Function block instance name | Calls a function block |
| JMP | C N | Label | Jumps to label |
| RET | C N | | Returns from function |
|) | | | Executes delayed operation |

In the next section, only operators which are specific to the IL language are described, other standard operators can be found in the section "standard operators, Function Blocks and Functions".

LD Operator

Operation: loads a value in the current result

Allowed modifiers: N

Operand: constant expression

internal, input or output Variable

Example

```
LDex:    LD      false      (* result := FALSE Boolean constant *)
          LD      true       (* result := TRUE Boolean constant *)
          LD      123        (* result := integer constant *)
          LD      123.1      (* result := real constant *)
          LD      t#3ms      (* result := time constant *)
          LD      boo_var1   (* result := Boolean Variable *)
          LD      ana_var1   (* result := integer Variable *)
          LD      tmr_var1   (* result := timer Variable *)
          LDN     boo_var2   (* result := NOT ( Boolean Variable ) *)
```

ST Operator

- Operation:** stores the current result in a variable.
The current result is not modified by this operation.
- Allowed modifiers:** N
- Operand:** internal or output Variable

Example

```
STboo:    LD      false
          ST      boo_var1 (* boo_var1 := FALSE *)
          STN     boo_var2 (* boo_var2 := TRUE *)
STana:    LD      123
          ST      ana_var1 (* ana_var1 := 123 *)
STtmr:    LD      t#12s
          ST      tmr_var1 (* tmr_var1 := t#12s *)
```

S Operator

- Operation:** stores the Boolean value TRUE in a Boolean Variable, if the current result has the Boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation
- Allowed modifiers:** (none)
- Operand:** output or internal Boolean Variable

Example

```
SETex: LD true      (* current result := TRUE *)
       S boo_var1  (* boo_var1 := TRUE *)
                  (* current result is not modified *)
       LD false     (* current result := FALSE *)
       S boo_var1  (* nothing done - boo_var1 unchanged *)
```

R Operator

Operation: stores the Boolean value FALSE in a Boolean Variable, if the current result has the Boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation

Allowed modifiers: (none)

Operand: output or internal Boolean Variable

Example

```
RESETex: LD true      (* current result := TRUE *)
          R boo_var1  (* boo_var1 := FALSE *)t
                  (* current result is not modified *)
          ST boo_var2  (* current result is not modified *)
          LD false     (* current result := FALSE *)
          R boo_var1  (* nothing done - boo_var1 unchanged *)
```

JMP Operator

Operation: jumps to the specified label

Allowed modifiers: C N

Operand: label defined in the same IL program

Example

(* the following example tests the value of an integer selector (0 or 1 or 2) *)

(* to set one from 3 output Booleans. *)

(*Test "is equal to 0" is made with the JMPC operator *)

```
JMPex: LD selector      (* selector is 0 or 1 or 2 *)
        ANY_TO_BOOL          (* conversion to Boolean *)
        JMPC test1           (* if selector = 0 then *)
        LD true
        ST bo0               (* bo0 := true *)
        JMP JMPend            (* end of the program *)
test1:  LD selector
        SUB 1                 (* decrease selector: is now 0
                                or 1 *)
        ANY_TO_BOOL          (* conversion to Boolean *)
        JMPC test2           (* if selector = 0 then *)
        LD true
        ST bo1               (* bo1 := true *)
        JMP JMPend            (* end of the program *)
test2:  LD true             (* last possibility *)
        ST bo2               (* bo2 := true *)
JMPend:                      (* end of the IL program *)
```

RET Operator

Operation: ends the current IL program. If the IL sequence is a Function, the current result is returned to the calling program

Allowed modifiers: C N

Operand: (none)

Example

(* the following example tests the value of an integer selector (0 or 1 or 2) *)
(* to set one from 3 output Booleans. *)
(*Test "is equal to 0" is made with the JMPC operator *)

```
JMPex: LD selector      (* selector is 0 or 1 or 2 *)
       ANY_TO_BOOL      (* conversion to Boolean *)
       JMPC test1      (* if selector = 0 then *)
       LD true
       ST bo0          (* bo0 := true *)
       RET             (* end - return 0 *)
                   (* decrease selector *)

test1: LD selector
       SUB 1           (* selector: is now 0 or 1 *)
       ANY_TO_BOOL      (* conversion to Boolean *)
       JMPC test2      (* if selector = 0 then *)
       LD true
       ST bo1          (* bo1 := true *)
       LD 1            (* load real selector value *)
       RET             (* end - return 1 *)
                   (* last possibility *)
test2: RETNC          (*returns if the selector has *)
```

```

ST          bo2      (* bo2 := true *)
:       LD         2       (* load real selector value *)
                      (* end - return 2 *)

```

) Operator

Operation: executes a delayed operation. the delayed operation was notified by "("

Allowed modifiers: (none)

Operand: (none)

Example

(* The following program interleaves delayed operations: *)

(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)

```

Delayed: LD     a1      (* result := a1; *)
          ADD( a2      (* delayed ADD - result := a2; *)
          MUL( a3      (* delayed MUL - result := a3; *)
          SUB  a4      (* result := a3 - a4; *)
          )        (* execute delayed MUL - result := a2 *
                     (a3-a4); *)
          MUL   a5      (* result := a2 * (a3 - a4) * a5; *)
          )        (* execute delayed ADD *)
                     (* result := a1 + (a2 * (a3 - a4) * a5); *)
          ADD   a6      (* result := a1 + (a2 * (a3 - a4) * a5) + a6;
                     *)
          ST    res     (* store current result in variable res *)

```

Calling Functions

A **Function call from IL** (written in any of the ST, LD, FBD, or "C" language), uses its name as an operator.

- Operation:** executes a Function - the value returned by the function is stored into the IL current result
- Allowed modifiers:** (none)
- Operand:** The first calling parameter must be stored in the current result before the call. The following ones are expressed in the operand field, separated by commas.

Example

(* Calling Function : converts an integer value into a time value *)

```
Main: LD          bi0
      MYFUNC      bi1,bi2 (* call function to get integer value
                           *)
      ST          result (* result := value returned by
                           function *)
      GT          vmax   (* test value overflow *)
      RETC        (* return if overflow *)
      LD          result
      MUL         1000   (* converts seconds in milliseconds
                           *)
      ANY_TO_TIME      (* converts to a timer *)
      ST          tmval  (* stores converted value in a timer
                           *)
```

(* Called Function named MYFUNC : evaluates the integer value *)

(* given as a binary value on three Boolean inputs: in0, in1, in2 are the three Boolean input parameters of the Function *)

```

LD           in2
ANY_TO_DINT          (* result = ANY_TO_DINT (in2); *)
MUL          2          (* result := 2*ANY_TO_DINT (in2); *)
ST           temporary  (* temporary := result *)
LD           in1
ANY_TO_DINT
ADD          temporary (* result := 2*ANY_TO_DINT (in2) +
ANY_TO_DINT (in1); *)
MUL          2          (* result := 4*ANY_TO_DINT (in2) +
2*ANY_TO_DINT (in1); *)
ST           temporary (* temporary := result *)
LD           in0
ANY_TO_DINT
ADD          temporary (* result := 4*ANY_TO_DINT (in2) +
2*ANY_TO_DINT (in1)+ANY_TO_DINT (in0); *)

```

Calling Function Blocks: CAL Operator

Operation: calls a function block

**Allowed
modifiers:** C N

Operand: Name of the function block instance.

The input parameters of the blocks must be assigned before the call using LD/ST operations sequence.

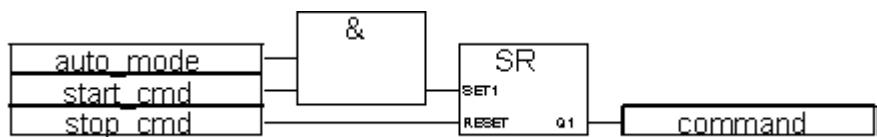
Output parameters are known if used.

Example

(* Calling function block SR : SR1 is an instance of SR *)

```
LD      auto_mode
AND    start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL    SR1
LD      SR1.Q1
ST      command
```

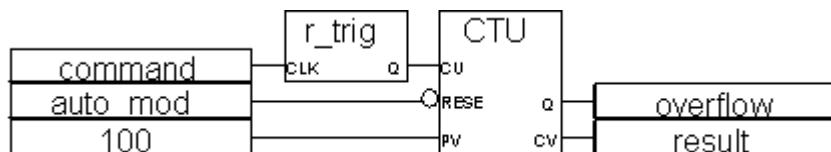
(* FBD equivalent : *)



(*We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)

```
LD      command
ST      R_TRIG1.clk
CAL    R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN   auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL    CTU1
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result
```

(* FBD equivalent: *)



Standard Operators

The following are **Standard Operators** of the IEC languages:

| | | |
|------------------------------|-----------------------|--|
| Arithmetic operations | Addition | Adds two or more variables |
| | Division | Divides two variables |
| | Multiplication | Multiplies two or more variables |
| | Subtraction | Subtracts a variable from another |
| | 1 GAIN | Assigns one variable into another |
| | NEG | Integer negation |
| Boolean operations | AND | Boolean AND |
| | OR | Boolean OR |
| | XOR | Boolean exclusive OR |
| | NOT | Boolean negation |
| Comparator | Less Than | Tests if one value is less than another |
| | Less Than or Equal | Tests if one value is less than or equal to another |
| | Greater Than | Tests if one value is greater than another |
| | Greater Than or Equal | Tests if one value is greater than or equal to another |
| | Equal | Tests if one value is equal to another |
| | Not Equal | Tests if one value is not equal to another |
| Data conversion | ANY_TO_BOOL | Converts to Boolean |
| | ANY_TO_SINT | Converts to Short integer |
| | ANY_TO_DINT | Converts to Double integer |
| | ANY_TO_REAL | Converts to Real |
| | ANY_TO_TIME | Converts to Time |
| | ANY_TO_STRING | Converts to String |

*



Note: For this operator, the number of inputs can be extended to more than two.

Arguments:

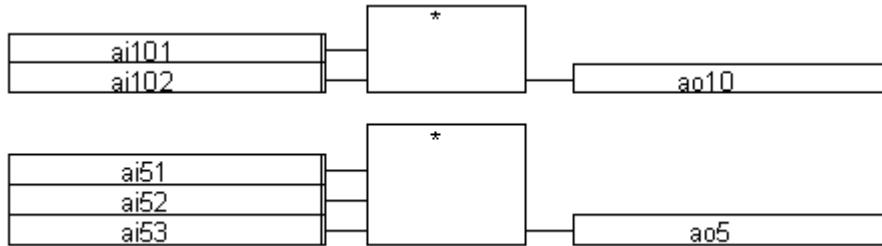
| | | |
|----------|--------------------|--|
| (inputs) | SINT - DINT - REAL | can be INTEGER or REAL (all inputs must have the same format) |
| output | SINT - DINT - REAL | multiplication of the input terms |

Description:

Multiplication of two or more integer or real variables.

Example

(* FBD example with Multiplication Operators *)



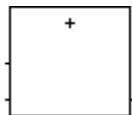
(* ST equivalence *)

```
ao10 := ai101 * ai102;  
ao5 := (ai51 * ai52) * ai53;
```

(* IL equivalence: *)

```
LD      ai101
MUL    ai102
ST      ao10
LD      ai51
MUL    ai52
MUL    ai53
ST      ao5
```

+



Note: For this **Operator**, the number of inputs can be extended to more than two.

Arguments:

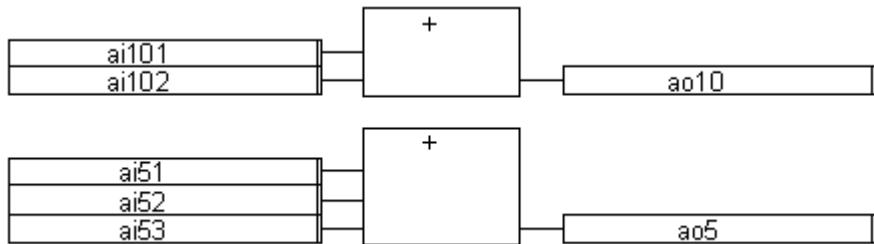
| | | |
|----------|---------------------------------------|--|
| (inputs) | SINT - DINT - REAL - TIME - STRING | can be of any integer, real, TIME, or STRING format (all inputs must have the same format) |
| output | SINT - DINT - REAL - TIME - STRING | addition of the input terms |

Description:

Addition of two or more integer, real, TIME, or STRING variables.

Example

(* FBD example with Addition Operators *)



(* ST equivalence: *)

```
ao10 := ai101 + ai102;  
ao5 := (ai51 + ai52) + ai53;
```

(* IL equivalence: *)

```
LD    ai101  
ADD  ai102  
ST    ao10  
LD    ai51  
ADD  ai52  
ADD  ai53  
ST    ao5
```



Arguments:

IN1 SINT - DINT - REAL - TIME

can be of any integer, real, or TIME format

IN2 SINT - DINT - REAL - TIME

(IN1 and IN2 must have the same format)

Q SINT - DINT - REAL - TIME

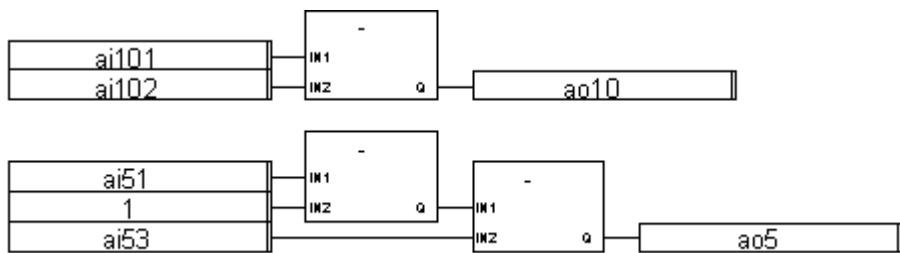
subtraction (first - second)

Description:

Subtraction of two integer, real, or TIME variables (first - second).

Example

(* FBD example with Subtraction Operators *)



(* ST equivalence: *)

```

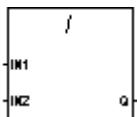
ao10 := ai101 - ai102;
ao5  := (ai51 - 1) - ai53;

```

(* IL equivalence: *)

```
LD    ai101
SUB  ai102
ST    ao10
LD    ai51
SUB  1
SUB  ai53
ST    ao5
```

/



Arguments:

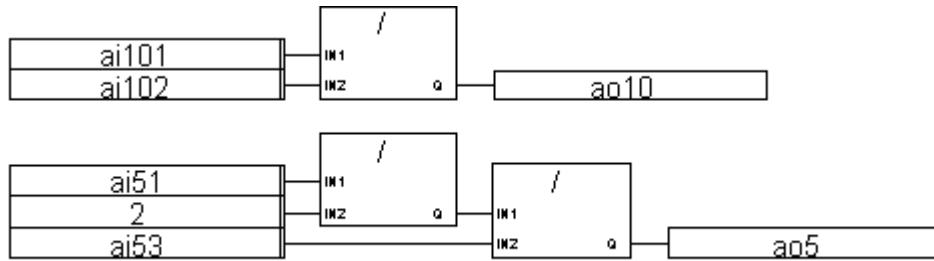
| | | |
|-----|--------------------|---|
| IN1 | SINT - DINT - REAL | can be of any integer or real format (operand) |
| IN2 | SINT - DINT - REAL | non-zero integer or real value (divisor) (IN1 and IN2 must have the same format) |
| Q | SINT - DINT - REAL | integer or real division of IN1 by IN2 |

Description:

Division of two integer or real variables (the first divided by the second).

Example

(* FBD example with Division Operators *)



(* ST Equivalence: *)

```
ao10 := ai101 / ai102;  
ao5 := (ai51 / 2) / ai53;
```

(* IL equivalence: *)

```
LD      ai101  
DIV    ai102  
ST      ao10  
LD      ai51  
DIV    2  
DIV    ai53  
ST      ao5
```

1 GAIN



Arguments:

IN SINT - DINT - REAL - TIME - STRING

Q SINT - DINT - REAL - TIME - STRING IN and Q must have the same format

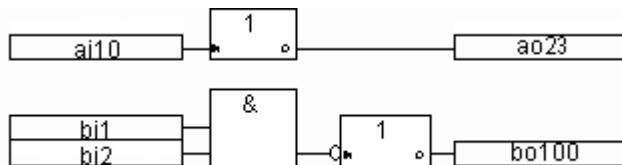
Description:

assignment of one variable into another one

This Block is very useful to directly link a diagram input and a diagram output. It can also be used (with a Boolean negation line) to invert the state of a line connected to a diagram output.

Example

(* FBD example with assignment Operators *)



(* ST equivalence: *)

```
ao23 := ai10;  
bo100 := NOT (bi1 AND bi2);
```

(* IL equivalence: *)

```
LD      ai10  
ST      ao23  
LD      bi1
```

```
AND      bi2
STN      bo100
```

AND



Note: For this Operator, the number of inputs can be extended to more than two.

Arguments:

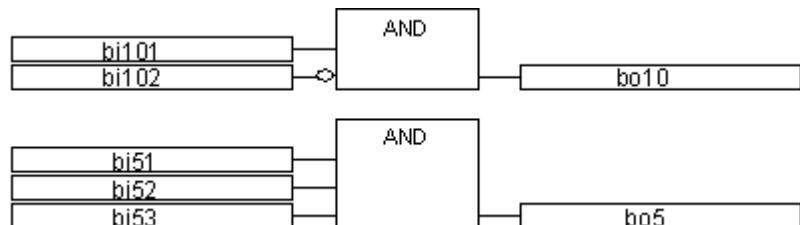
(inputs) BOOL
output BOOL Boolean AND of the input terms

Description:

Boolean **AND** between two or more terms.

Example

(* FBD example with "AND" Operators *)



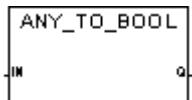
(* ST equivalence: *)

```
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
```

(* IL equivalence *)

```
LD      bi101    (* current result := bi101 *)
ANDN   bi102    (* current result := bi101 AND not(bi102) *)
ST      bo10     (* bo10 := current result *)
LD      bi51     (* current result := bi51; *)
&      bi52     (* current result := bi51 AND bi52 *)
&      bi53     (* current result := (bi51 AND bi52) AND bi53 *)
ST      bo5      (* bo5 := current result *)
```

ANY_TO_BOOL



Arguments:

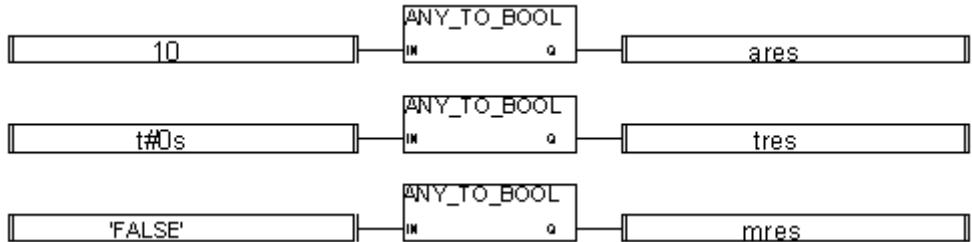
| | | |
|----|------------------------------------|---|
| IN | SINT - DINT - REAL - TIME - STRING | any non-Boolean value |
| Q | BOOL | TRUE for non-zero numerical value FALSE for zero numerical value TRUE for 'TRUE' string FALSE for 'FALSE' string |

Description:

Converts **any variable** to a **Boolean variable**

Example

(* FBD example with "Convert to Boolean" Operators *)



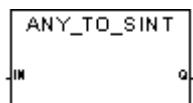
(* ST Equivalence: *)

```
ares := ANY_TO_BOOL (10);          (* ares is TRUE *)
tres := ANY_TO_BOOL (t#0s);        (* tres is FALSE *)
mres := ANY_TO_BOOL ('FALSE');     (* mres is FALSE *)
```

(* IL equivalence: *)

```
LD      10
ANY_TO_BOOL
ST      ares
LD      t#0s
ANY_TO_BOOL
ST      tres
LD      'FALSE'
ANY_TO_BOOL
ST      mres
```

ANY_TO_SINT



Arguments:

| | | |
|----|------------------------------------|---|
| IN | BOOL - DINT - REAL - TIME - STRING | any value other than a short integer |
| Q | SINT | 0 if IN is FALSE / 1 if IN is TRUE number of milliseconds for a timer integer part for real decimal number represented by a string |
| | | |
| | | |

Description:

Converts **any variable** to a **Short integer variable** (8-bit)

Example

(* FBD example with "Convert to Short Integer" Operators *)



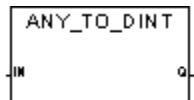
(* ST Equivalence: *)

```
bres := ANY_TO_SINT (true);          (* bres is 1 *)
tres := ANY_TO_SINT (t#0s46ms);      (* tres is 46 *)
mres := ANY_TO_SINT ('0198');         (* mres is 198 *)
```

(* IL equivalence: *)

```
LD          true
ANY_TO_SINT
ST          bres
LD          t#1s46ms
ANY_TO_SINT
ST          tres
LD          '0198'
ANY_TO_SINT
ST          mres
```

ANY_TO_DINT



Arguments:

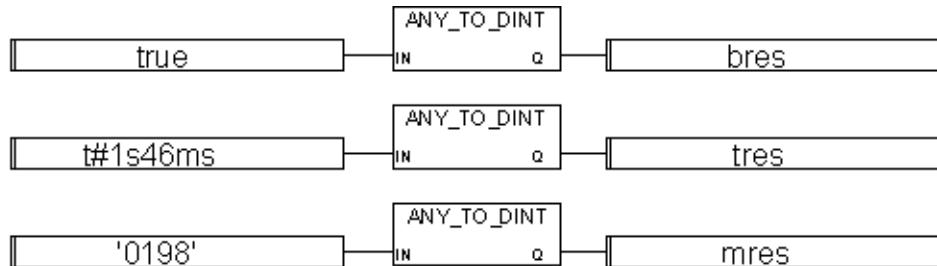
| | | |
|----|------------------------------------|--|
| IN | BOOL - SINT - REAL - TIME - STRING | any value other than a double integer |
| Q | DINT | 0 if IN is FALSE / 1 if IN is TRUE |
| | | number of milliseconds for a timer |
| | | integer part for real |
| | | decimal number represented by a string |

Description:

Converts **any variable** to a **double integer variable** (32-bit)

Example

(* FBD example with "Convert to Double Integer" Operators *)



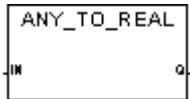
(* ST Equivalence: *)

```
bres := ANY_TO_DINT (true);          (* bres is 1 *)
tres := ANY_TO_DINT (t#1s46ms);      (* tres is 1046 *)
mres := ANY_TO_DINT ('0198');        (* mres is 198 *)
```

(* IL equivalence: *)

```
LD          true
ANY_TO_DINT
ST          bres
LD          t#1s46ms
ANY_TO_DINT
ST          tres
LD          '0198'
ANY_TO_DINT
ST          mres
```

ANY_TO_REAL



Arguments:

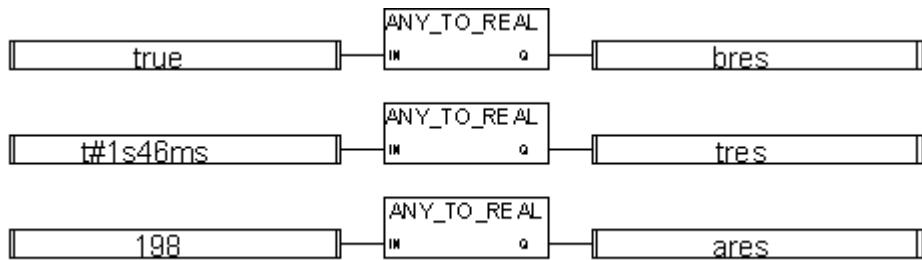
| | | |
|----|------------------------------------|---|
| IN | BOOL - SINT - DINT - TIME - STRING | any value other than a real |
| Q | REAL | 0.0 if IN is FALSE / 1.0 if IN is TRUE number of milliseconds for a timer equivalent number for integer |

Description:

Converts **any variable** to a **real variable**

Example

(* FBD example with "Convert to Real" Operators *)



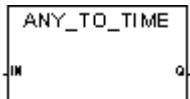
(* ST Equivalence: *)

```
bres := ANY_TO_REAL (true);          (* bres is 1.0 *)
tres := ANY_TO_REAL (t#1s46ms);      (* tres is 1046.0 *)
ares := ANY_TO_REAL (198);           (* ares is 198.0 *)
```

(* IL equivalence: *)

```
LD          true
ANY_TO_REAL
ST          bres
LD          t#1s46ms
ANY_TO_REAL
ST          tres
LD          198
ANY_TO_REAL
ST          ares
```

ANY_TO_TIME



Arguments:

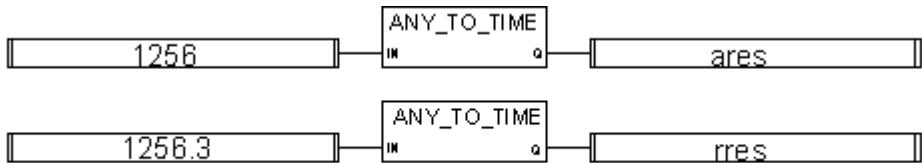
- | | | |
|----|------------------------------------|--|
| IN | BOOL - SINT - DINT - REAL - STRING | any positive value other than a time and date format IN (or integer part of IN if it is real) is the number of milliseconds STRING (number of milliseconds, for example, a value of 300032 represents 5 minutes and 32 milliseconds) |
| Q | TIME | time value represented by IN. A value of 1193h2m47s295ms indicates an invalid time. |

Description:

Converts **any variable other than a time or date type** to a **timer variable**. The SUB_DATE_DATE function enables the conversion of a date type to a time format. For details on the SUB_DATE_DATE function, see page 567.

Example

(* FBD example with "Convert to Timer" Operators *)



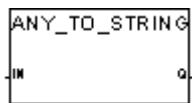
(* ST Equivalence: *)

```
ares := ANY_TO_TIME (1256);          (* ares := t#1s256ms *)
rres := ANY_TO_TIME (1256.3);        (* rres := t#1s256ms *)
```

(* IL equivalence: *)

```
LD      1256
ANY_TO_TIME
ST      ares
LD      1256.3
ANY_TO_TIME
ST      rres
```

ANY_TO_STRING



Arguments:

IN BOOL - SINT - DINT - REAL - any non-string value
TIME

Q STRING

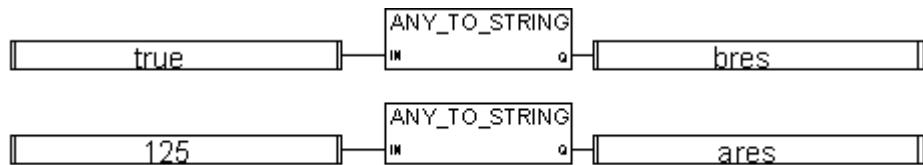
If IN is a Boolean, 'FALSE' or 'TRUE'
If IN is an integer or a real, decimal representation
If IN is a TIME:
TIME time1
STRING s1
time1 :=13 ms;
s1 :=ANY_TO_STRING(time1);
(* s1 = '0s13' *)

Description:

Converts **any variable** to a **string variable**

Example

(* FBD example with "Convert to string" Operators *)



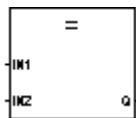
(* ST Equivalence: *)

```
bres := ANY_TO_STRING (TRUE);      (* bres is 'TRUE' *)
ares := ANY_TO_STRING (125);        (* ares is '125' *)
```

(* IL equivalence: *)

| | |
|---------------|------|
| LD | true |
| ANY_TO_STRING | |
| ST | bres |
| LD | 125 |
| ANY_TO_STRING | |
| ST | ares |

Equal



Arguments:

IN1 BOOL - SINT - DINT - REAL - TIME -
 STRING

Both inputs must have the same format.
The TIME input only applies to the ST
and IL languages. The BOOL input is
not accepted in the IL language.

IN2 BOOL - SINT - DINT - REAL - TIME -
 STRING

Q BOOL

TRUE if IN1 = IN2

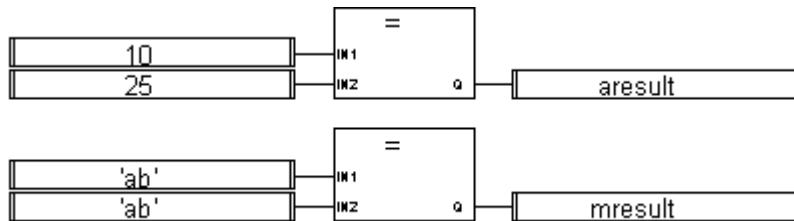
Description

Test if one value is **EQUAL TO** another one (on integer, real, time, date, and string variables)

Note: The equality test on a TIME variable is not recommended for testing output of TIME
blocks such as TON, TP, TOF, BLINK and for testing StepName.t in SFC chart.

Example

(* FBD example with "Is Equal to" Operators *)



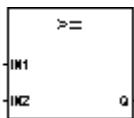
(* ST Equivalence: *)

```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```

(* IL equivalence: *)

```
LD    10
EQ    25
ST    aresult
LD    'ab'
EQ    'ab'
ST    mresult
```

Greater Than or Equal



Arguments:

IN1 SINT - DINT - REAL - TIME - STRING

Both inputs must have the same type.
The TIME input only applies to the ST
and IL languages.

IN2 SINT - DINT - REAL - TIME - STRING

TRUE if $\text{IN1} \geq \text{IN2}$

Q BOOL

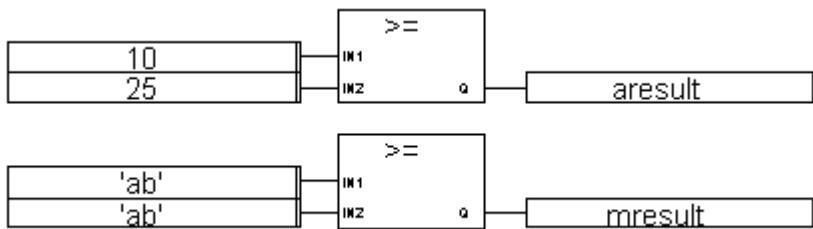
Description:

Test if one value is **GREATER THAN or EQUAL TO** another one (on integers, reals, times, dates, or strings)

Note: The equality test on a TIME variable is not recommended for testing output of TIME blocks such as TON, TP, TOF, BLINK and for testing StepName.t in SFC chart.

Example

(* FBD example with "Greater or Equal to" Operators *)



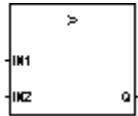
(* ST Equivalence: *)

```
aresult := (10 >= 25); (* aresult is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```

(* IL equivalence: *)

```
LD    10
GE    25
ST    aresult
LD    'ab'
GE    'ab'
ST    mresult
```

Greater Than



Arguments:

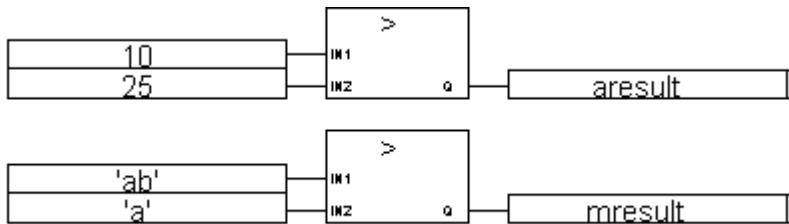
| | | |
|-----|------------------------------------|-------------------------------------|
| IN1 | SINT - DINT - REAL - TIME - STRING | Both inputs must have the same type |
| IN2 | SINT - DINT - REAL - TIME - STRING | |
| Q | BOOL | TRUE if IN1 > IN2 |

Description:

Test if one value is **GREATER THAN** another one (on integers, reals, times, dates, or strings)

Example

(* FBD example with "Greater than" Operators *)



(* ST Equivalence: *)

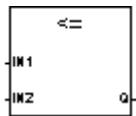
```
aresult := (10 > 25); (* aresult is FALSE *)
```

```
mresult := ('ab' > 'a'); (* mresult is TRUE *)
```

(* IL equivalence: *)

```
LD    10
GT    25
ST    aresult
LD    'ab'
GT    'a'
ST    mresult
```

Less Than or Equal



Arguments:

IN1 SINT - DINT - REAL - TIME - STRING

IN2 SINT - DINT - REAL - TIME - STRING

Q BOOL

Both inputs must have the same type.
The TIME input only applies to the ST
and IL languages.

TRUE if $\text{IN1} \leq \text{IN2}$

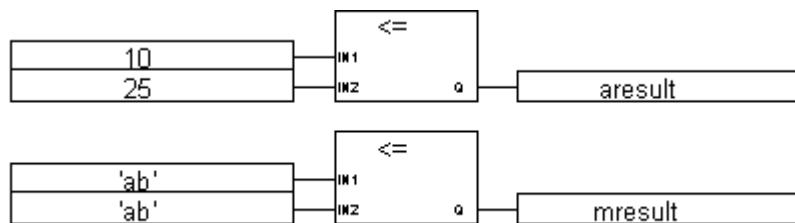
Description:

Tests if one value is **LESS THAN or EQUAL TO** another one (on integers, reals, times, dates, or strings)

Note: The equality test on a TIME variable is not recommended for testing output of TIME blocks such as TON, TP, TOF, BLINK and for testing StepName.t in SFC chart.

Example

(* FBD example with "Less or equal to" Operators *)



(* ST Equivalence: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```

(* IL equivalence: *)

```
LD    10
LE    25
ST    aresult
LD    'ab'
LE    'ab'
ST    mresult
```

Less Than



Arguments:

IN1 SINT - DINT - REAL - TIME - STRING Both inputs must have the same type

IN2 SINT - DINT - REAL - TIME - STRING

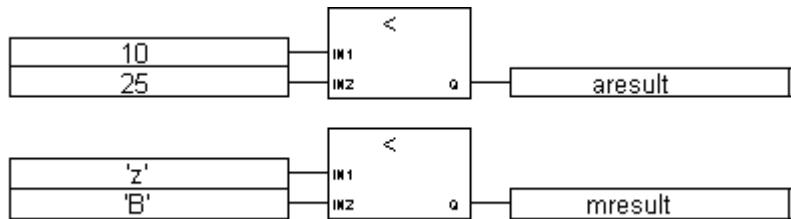
Q BOOL TRUE if IN1 < IN2

Description:

Test if one value is **LESS THAN** another one (on integers, reals, times, dates, or strings)

Example

(* FBD example with "Less than" Operators *)



(* ST Equivalence: *)

```
aresult := (10 < 25); (* aresult is TRUE *)
mresult := ('z' < 'B'); (* mresult is FALSE *)
```

(* IL equivalence: *)

```
LD    10
LT    25
ST    aresult
LD    'z'
LT    'B'
ST    mresult
```

NEG



Arguments:

IN SINT - DINT - REAL

Input and output must have the same format

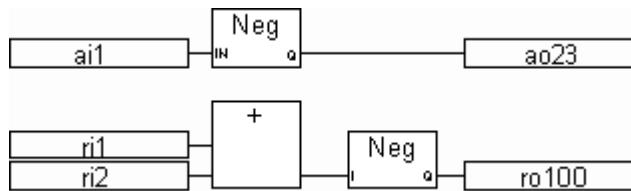
Q SINT - DINT - REAL

Description:

Assignment of the **negation** of a variable.

Example

(* FBD example with Negation Operators *)



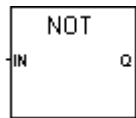
(* ST equivalence: *)

```
ao23 := - (ai10);  
ro100 := - (ri1 + ri2);
```

(* IL equivalence: *)

```
LD      ai10  
MUL    -1  
ST      ao23  
LD      ri1  
ADD    ri2  
MUL    -1.0  
ST      ro100
```

NOT



Arguments:

IN: Any Boolean variable or complex expression

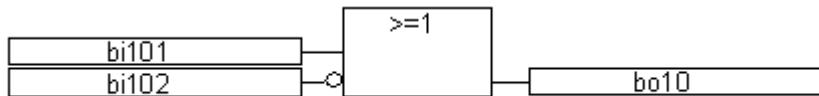
Q: TRUE when IN is FALSE
FALSE when IN is TRUE

Description:

Returns the negation of a complete Boolean expression.

Example

(* FBD example with "NOT" Operator *)



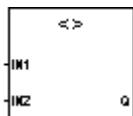
(* ST equivalence: *)

```
bo10 := bi101 XOR NOT (bi102);
```

(* IL equivalence: *)

```
LD    bi101
XORN bi102
ST    bo10
```

Not Equal



Arguments:

IN1 BOOL - DINT - REAL - TIME - STRING both inputs must have the same type

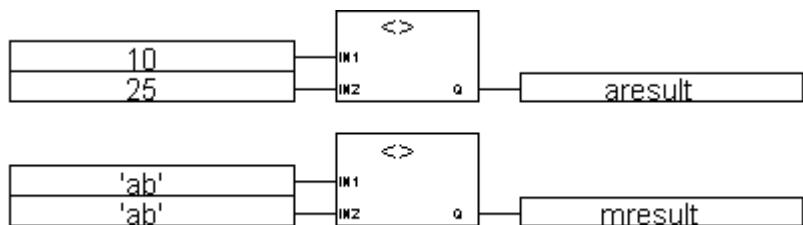
IN2 BOOL - DINT - REAL - TIME - STRING

Description:

Test if one value is **NOT EQUAL TO** another one (on integer, real, time, date, and string variables)

Example

(* FBD example with "Is Not Equal to" Operators *)



(* ST Equivalence: *)

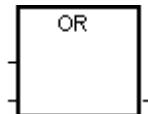
```
aresult := (10 <> 25); (* aresult is TRUE *)  
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```

(* IL equivalence: *)

LD 10
NE 25

```
ST      aresult
LD      'ab'
NE      'ab'
ST      mresult
```

OR



Note: For this Operator, the number of inputs can be extended to more than two.

Arguments:

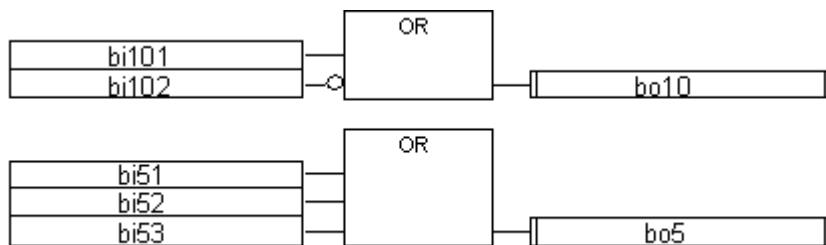
(inputs) BOOL
output BOOL Boolean **OR** of the input terms

Description:

Boolean OR of two or more terms.

Example

(* FBD example with "OR" Operators *)



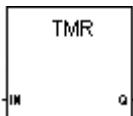
(* ST equivalence: *)

```
bo10 := bi101 OR NOT (bi102);  
bo5 := (bi51 OR bi52) OR bi53;
```

(* IL equivalence: *)

```
LD    bi101  
ORN   bi102  
ST    bo10  
LD    bi51  
OR    bi52  
OR    bi53  
ST    bo5
```

TMR



Arguments:

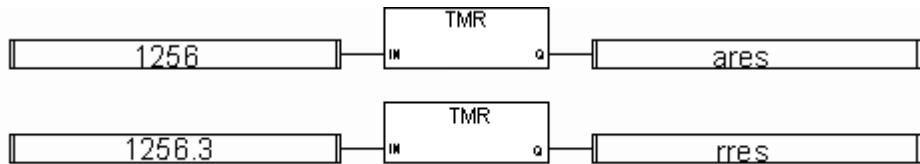
| | | |
|----|------|---|
| IN | DINT | A non-TIME value IN (or integer part of IN if it is real) is the number of milliseconds |
| Q | TIME | Time value represented by IN |

Description:

Converts an integer or real variable to a time one.

Example

(* FBD example with "Convert to Timer" Operators *)



(* ST Equivalence: *)

```
ares := TMR (1256);          (* ares := t#1s256ms *)
rres := TMR (1256.3);        (* rres := t#1s256ms *)
```

(* IL equivalence: *)

```
LD      1256
TMR
ST      ares
LD      1256.3
TMR
ST      rres
```

XOR



Arguments:

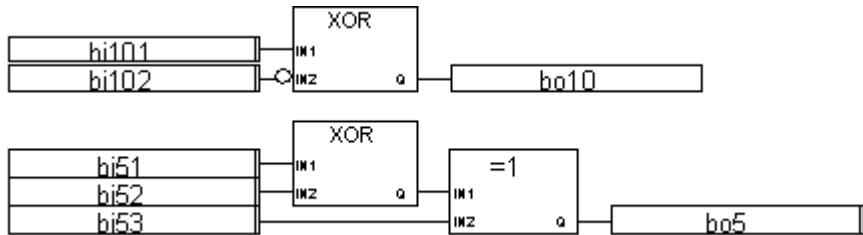
| | | |
|-----|------|--|
| IN1 | BOOL | |
| IN2 | BOOL | |
| Q | BOOL | Boolean exclusive OR of the two input terms |

Description:

Boolean exclusive OR between two terms.

Example

(* FBD example with "XOR" operators *)



(* ST equivalence: *)

```
bo10 := bi101 XOR NOT (bi102);  
bo5 := (bi51 XOR bi52) XOR bi53;
```

(* IL equivalence: *)

```
LD      bi101  
XORN   bi102  
ST      bo10  
LD      bi51  
XOR    bi52  
XOR    bi53  
ST      bo5
```


Standard Functions

The following are the standard functions supported by the system:

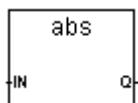
| | | |
|------------------------------|------------------|---|
| Arithmetic Operations | ABS | Absolute value of a real value |
| | EXPT, POW | Exponent, power calculation of real values |
| | LOG | Logarithm of a real value |
| | MOD | Modulo |
| | SQRT | Square root of a real value |
| | RAND | Random value |
| | TRUNC | Truncate decimal part of a real value |
| | ACOS, ASIN, ATAN | Arc cosine, Arc sine, Arc tangent of a real value |
| | COS, SIN, TAN | Cosine, Sine, Tangent of a real value |
| Binary operations | AND_MASK | Integer bit-to-bit AND mask |
| | OR_MASK | Integer bit-to-bit OR mask |
| | XOR_MASK | Integer bit-to-bit Exclusive OR mask |
| | NOT_MASK | Integer bit-to-bit negation |
| | ROL, ROR | Rotate Left, Rotate Right an integer value |
| | SHL, SHR | Shift Left, Shift Right an integer value |
| Boolean operations | ODD | Odd parity |
| Data manipulation | MIN, MAX, LIMIT | Minimum, Maximum, Limit |
| | MUX4, MUX8 | Multiplexer (4 or 8 entries) |
| | SEL | Binary selector |
| String manipulation | ASCII | Character -> ASCII code |
| | CHAR | ASCII code -> Character |
| | MLEN | Get string length |
| | DELETE, INSERT | Delete sub-string, Insert string |
| | FIND, REPLACE | Find sub-string, Replace sub-string |
| | LEFT, MID, RIGHT | Extract left, middle or right of a string |

Time operations

CURRENT_ISA_DATE Gets the current date

SUB_DATE_DATE Compares two dates and gives the difference in TIME format

ABS



Arguments:

IN REAL Any signed real value

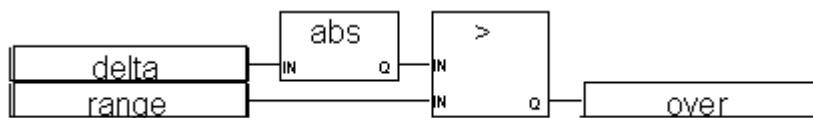
Q REAL Absolute value (always positive)

Description:

Gives the absolute (positive) value of a real value.

Example

(* FBD Program using "ABS" Function *)



(* ST Equivalence: *)

```
over := (ABS (delta) > range);
```

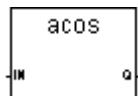
(* IL Equivalence: *)

```
LD    delta
```

```
ABS
```

GT range
ST over

ACOS



Arguments:

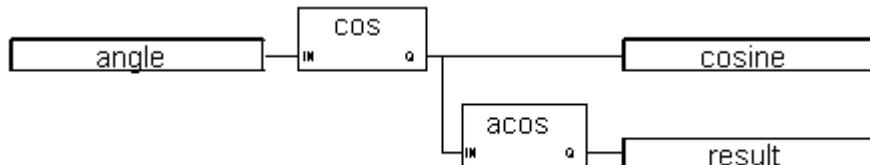
IN REAL Must be in set [-1.0 .. +1.0]
Q REAL Arc-cosine of the input value (in set [0.0 .. PI])
= 0.0 for invalid input

Description:

Calculates the Arc cosine of a real value.

Example

(* FBD Program using "COS" and "ACOS" Functions *)



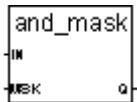
(* ST Equivalence: *)

```
cosine := COS (angle);  
result := ACOS (cosine); (* result is equal to angle *)
```

(* IL Equivalence: *)

```
LD      angle
COS
ST      cosine
ACOS
ST      result
```

AND_MASK



Arguments:

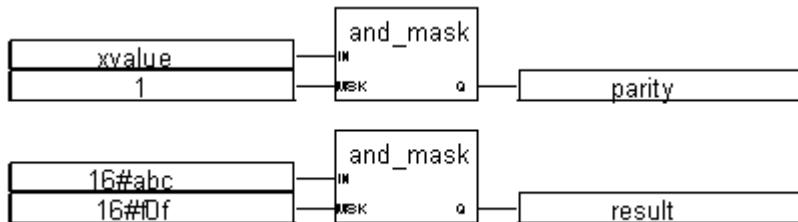
| | | |
|-----|------|--|
| IN | DINT | Must have integer format |
| MSK | DINT | Must have integer format |
| Q | DINT | Bit-to-bit logical AND between IN and MSK |

Description:

Integer AND bit-to-bit mask.

Example

(* FBD example with AND_MASK Operators *)



```
(* ST Equivalence: *)
parity := AND_MASK (xvalue, 1); (* 1 if xvalue is odd *)
result := AND_MASK (16#abc, 16#f0f); (* equals 16#a0c *)
```

```
(* IL equivalence: *)
```

```
LD      xvalue
AND_MASK 1
ST      parity
LD      16#abc
AND_MASK 16#f0f
ST      result
```

ASCII



Arguments:

| | | |
|------|--------|---|
| IN | STRING | Any non-empty string |
| Pos | DINT | Position of the selected character in set [1.. len] (len is the length of the IN string) |
| Code | DINT | Code of the selected character (in set [0 .. 255]) returns 0 if Pos is out of the string |

Description:

Gives the ASCII code of one character in a string.

Example

```
(* FBD Program using "ASCII" Function *)
```



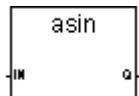
(* ST Equivalence: *)

```
FirstChr := ASCII (message, 1);
(* FirstChr is the ASCII code of the first character of the string *)
```

(* IL Equivalence: *)

```
LD      message
ASCII   1
ST      FirstChr
```

ASIN



Arguments:

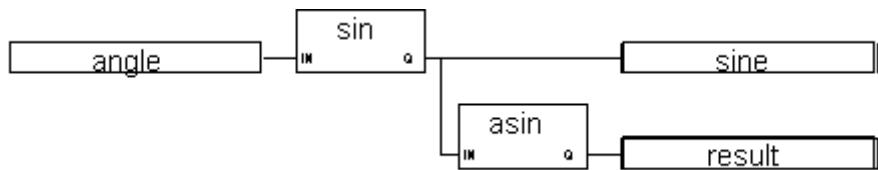
| | | |
|----|------|--|
| IN | REAL | Must be in set [-1.0 .. +1.0] |
| Q | REAL | Arc-sine of the input value (in set [-PI/2 .. +PI/2]) = 0.0 for invalid input |

Description:

Calculates the Arc sine of a real value.

Example

(* FBD Program using "SIN" and "ASIN" Functions *)



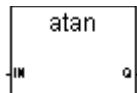
(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

(* IL Equivalence: *)

```
LD      angle
SIN
ST      sine
ASIN
ST      result
```

ATAN



Arguments:

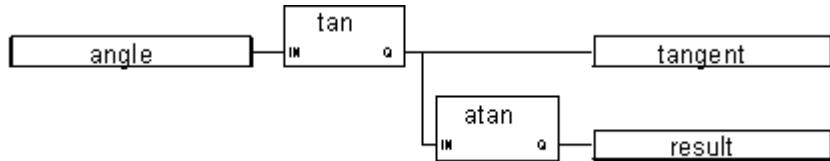
| | | |
|----|------|---|
| IN | REAL | Any real value |
| Q | REAL | Arc-tangent of the input value (in set [-PI/2 .. +PI/2]) = 0.0 for invalid input |

Description:

Calculates the arc tangent of a real value.

Example

(* FBD Program using "TAN" and "ATAN" Function *)



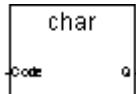
(* ST Equivalence: *)

```
tangent := TAN (angle);  
result := ATAN (tangent); (* result is equal to angle*)
```

(* IL Equivalence: *)

```
LD      angle  
TAN  
ST      tangent  
ATAN  
ST      result
```

CHAR



Arguments:

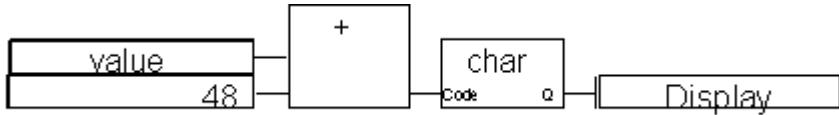
| | | |
|------|--------|---|
| Code | DINT | Code in set [0 .. 255] |
| Q | STRING | One character string the character has the ASCII code given in input Code (ASCII code is used modulo 256) |

Description:

Gives a one character string from a given ASCII code.

Example

(* FBD Program using "CHAR" Function *)



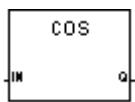
(* ST Equivalence: *)

```
Display := CHAR ( value + 48 );  
(* value is in set [0..9] *)  
(* 48 is the ascii code of '0' *)  
(* result is one character string from '0' to '9' *)
```

(* IL Equivalence: *)

```
LD      value  
ADD    48  
CHAR  
ST      Display
```

COS



Arguments:

IN REAL Any REAL value

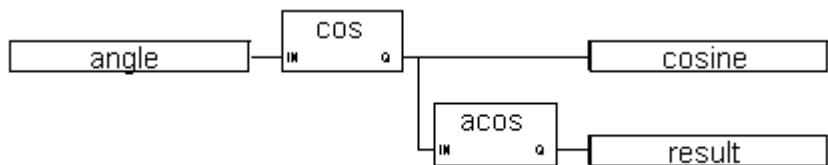
Q REAL Cosine of the input value (in set [-1.0 .. +1.0])

Description:

Calculates the cosine of a real value.

Example

(* FBD Program using "COS" and "ACOS" Functions *)



(* ST Equivalence: *)

```
cosine := COS (angle);  
result := ACOS (cosine); (* result is equal to angle *)
```

(* IL Equivalence: *)

```
LD      angle  
COS  
ST      cosine  
ACOS  
ST      result
```

CURRENT_ISA_DATE



Arguments:

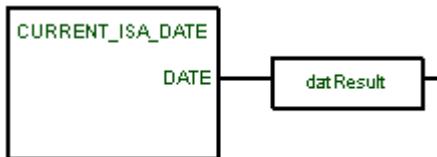
DATE DATE The current date

Description:

Gets the current date. The ANY_TO_DINT function enables the conversion of DATE to the number of seconds since 1970/01/01 00:00:00:000 GMT (Greenwich Mean Time).

Example

(* FBD Program using "CURRENT_ISA_DATE" Function *)



(* ST Equivalence: *)

```
datResult := CURRENT_ISA_DATE();
```

(* IL Equivalence: *)

```
CURRENT_ISA_DATE  
ST           datResult
```

DELETE



Arguments:

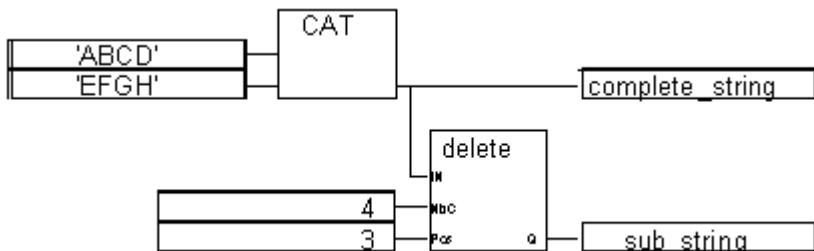
| | | |
|-----|--------|--|
| IN | STRING | Any non-empty string |
| NbC | DINT | Number of characters to be deleted |
| Pos | DINT | Position of the first deleted character (first character of the string has position 1) |
| Q | STRING | modified string empty string if Pos < 1 initial string if Pos > IN string length initial string if NbC <= 0 |

Description:

Deletes a part of a string.

Example

(* FBD Program using "DELETE" Function *)



```

(* ST Equivalence: *)

complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
sub_string := DELETE (complete_string, 4, 3); (* sub_string is 'ABGH'*)

(* IL Equivalence: *)

LD      'ABCD'
ADD    'EFGH'
ST     complete_string
DELETE 4,3
ST     sub_string

```

EXPT



Arguments:

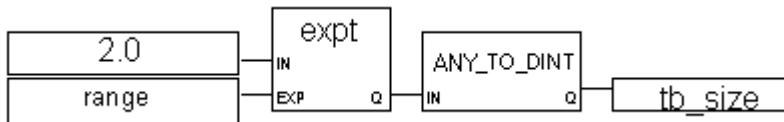
| | | |
|-----|------|-----------------------|
| IN | REAL | Any signed real value |
| EXP | DINT | Integer exponent |
| Q | REAL | (IN EXP) |

Description:

Gives the real result of the operation: (base ^{exponent}) 'base' being the first argument and 'exponent' the second one.

Example

(* FBD Program using "EXPT" Function *)



(* ST Equivalence: *)

```
tb_size := ANY_TO_DINT (EXPT (2.0, range) );
```

(* IL Equivalence: *)

```
LD          2.0
EXPT      range
ANY_TO_DINT
ST          tb_size
```

FIND



Arguments:

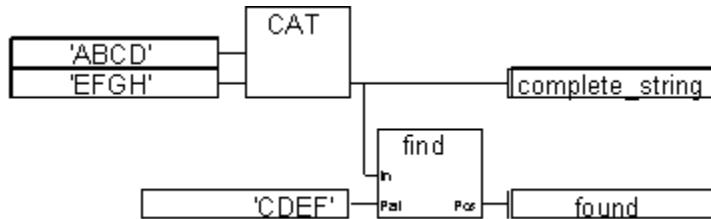
| | | |
|-----|--------|--|
| In | STRING | Any string |
| Pat | STRING | Any non-empty string (Pattern) |
| Pos | DINT | = 0 if sub string Pat not found = position of the first character of the first occurrence of the sub-string Pat (first position is 1) this function is case sensitive |

Description:

Finds a sub-string in a string. Gives the position in the string of the sub-string.

Example

(* FBD Program using "FIND" Function *)



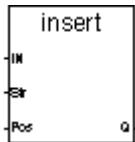
(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```

(* IL Equivalence: *)

```
LD      'ABCD'
ADD    'EFGH'
ST     complete_string
FIND   'CDEF'
ST     found
```

INSERT



Arguments:

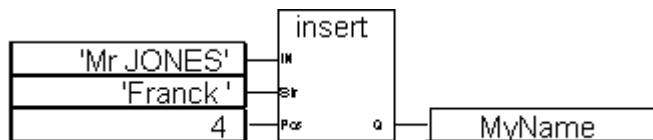
| | | |
|-----|--------|---|
| IN | STRING | Initial string |
| Str | STRING | String to be inserted |
| Pos | DINT | Position of the insertion the insertion is done before the position (first valid position is 1) |
| Q | STRING | Modified string empty string if Pos <= 0 concatenation of both strings if Pos is greater than the length of the IN string |

Description:

Inserts a sub-string in a string at a given position.

Example

(* FBD Program using "INSERT" Function *)



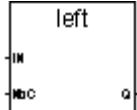
(* ST Equivalence: *)

```
MyName := INSERT ('Mr JONES', 'Frank ', 4);  
(* MyName is 'Mr Frank JONES' *)
```

(* IL Equivalence: *)

```
LD      'Mr JONES'  
INSERT  'Frank ',4  
ST      MyName
```

LEFT



Arguments:

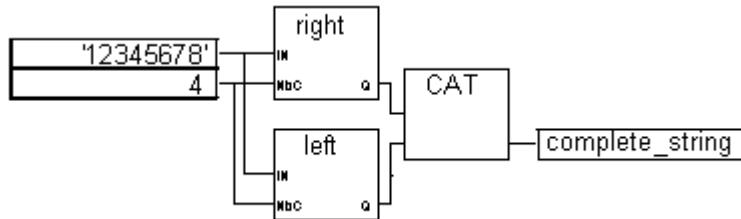
| | | |
|-----|--------|--|
| IN | STRING | Any non-empty string |
| NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| Q | STRING | Left part of the IN string (its length = NbC) empty string if NbC <= 0 complete IN string if NbC >= IN string length |

Description:

Extracts the left part of a string. The number of characters to be extracted is given.

Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)



(* ST Equivalence: *)

```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

(* IL Equivalence: First done is call to LEFT *)

```
LD      '12345678'  
LEFT    4  
ST      sub_string (* intermediate result *)  
LD      '12345678'  
RIGHT   4  
ADD    sub_string  
ST      complete_string
```

LIMIT



Arguments:

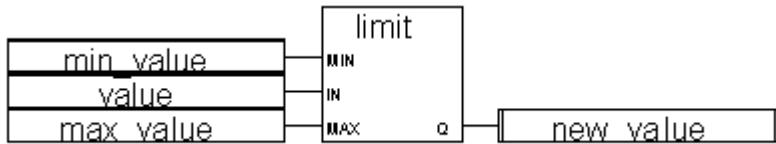
| | | |
|-----|------|--------------------------------------|
| MIN | DINT | Minimum allowed value |
| IN | DINT | Any signed integer value |
| MAX | DINT | Maximum allowed value |
| Q | DINT | Input value bounded to allowed range |

Description:

Limits an integer value into a given interval. Whether it keeps its value if it is between minimum and maximum, or it is changed to maximum if it is above, or it is changed to minimum if it is below.

Example

(* FBD Program using "LIMIT" Function *)



(* ST Equivalence: *)

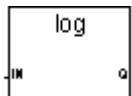
```
new_value := LIMIT (min_value, value, max_value);
```

(* bounds the value to the [min_value..max_value] set *)

(* IL Equivalence: *)

```
LD      min_value
LIMIT  value, max_value
ST      new_value
```

LOG



Arguments:

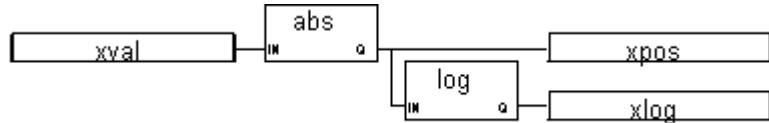
IN REAL Must be greater than zero
Q REAL Logarithm (base 10) of the input value

Description:

Calculates the logarithm (base 10) of a real value.

Example

(* FBD Program using "LOG" Function *)



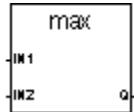
(* ST Equivalence: *)

```
xpos := ABS (xval);  
xlog := LOG (xpos);
```

(* IL Equivalence: *)

```
LD      xval
ABS
ST      xpos
LOG
ST      xlog
```

MAX



Arguments:

IN1 DINT Any signed integer value
IN2 DINT (cannot be REAL)
Q DINT Maximum of both input values

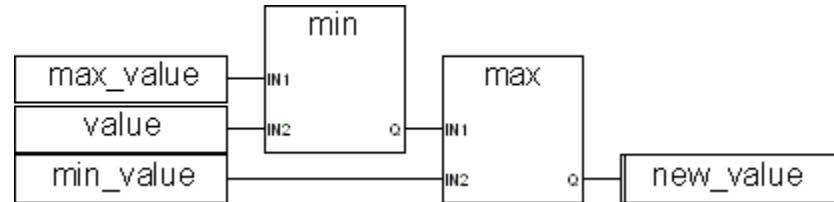
Description:

Gives the maximum of two integer values.

Example

(* FBD Program using "MIN" and "MAX" Function *)

)



(* ST Equivalence: *)

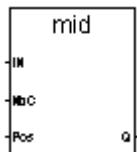
```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

(* IL Equivalence: *)

```
LD      max_value  
MIN    value  
MAX    min_value  
ST      new_value
```

MID



Arguments:

IN STRING Any non-empty string

NbC DINT Number of characters to be extracted cannot be greater than the length of the IN string

Pos DINT Position of the sub-string
the sub-string first character will be the one pointed to by Pos
(first valid position is 1)

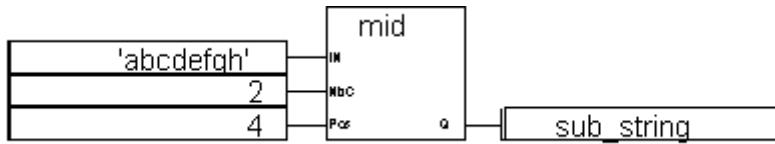
Q STRING Middle part of the string (its length = NbC)
empty string if parameters are not valid

Description:

Extracts a part of a string. The number of characters to be extracted and the position of the first character are given.

Example

(* FBD Program using "MID" Function *)



(* ST Equivalence: *)

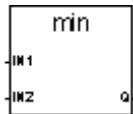
```
sub_string := MID ('abcdefgh', 2, 4);
```

(* sub_string is 'de' *)

(* IL Equivalence: *)

```
LD      'abcdefgh'  
MID    2, 4  
ST      sub_string
```

MIN



Arguments:

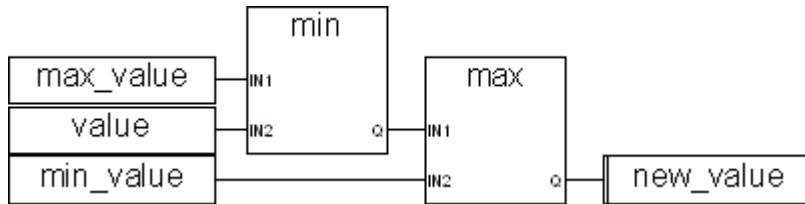
| | | |
|-----|------|------------------------------|
| IN1 | DINT | Any signed integer value |
| IN2 | DINT | (cannot be REAL) |
| Q | DINT | Minimum of both input values |

Description:

Gives the minimum of two integer values.

Example

(* FBD Program using "MIN" and "MAX" Function *)



(* ST Equivalence: *)

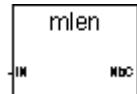
```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

(* IL Equivalence: *)

```
LD      max_value
MIN    value
MAX    min_value
ST      new_value
```

MLEN



Arguments:

IN STRING Any string

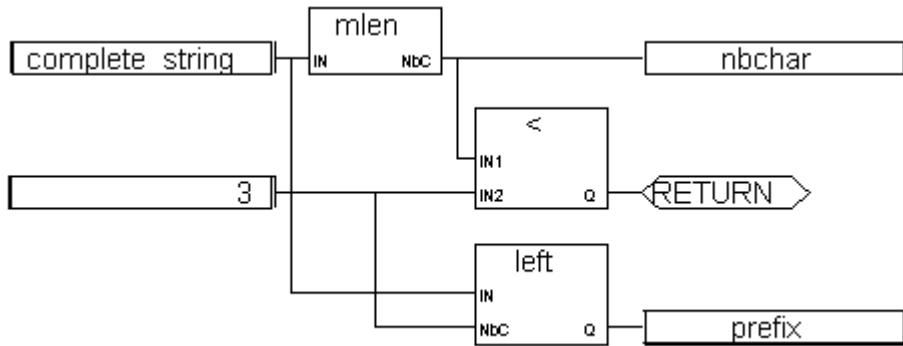
NbC DINT Number of characters in the IN string

Description:

Calculates the length of a string.

Example

(* FBD Program using "MLEN" Function *)



(* ST Equivalence: *)

```
nbchar := MLEN (complete_string);  
If (nbchar < 3) Then Return; End_if;  
prefix := LEFT (complete_string, 3);
```

(* this program extracts the 3 characters on the left of the string and put the result in the prefix string variable

nothing is done if the string length is less than 3 characters *)

(* IL Equivalence: *)

```
LD      complete_string  
MLEN  
ST      nbchar  
LT      3  
RETC  
LD      complete_string  
LEFT   3  
ST      prefix
```

MOD



Arguments:

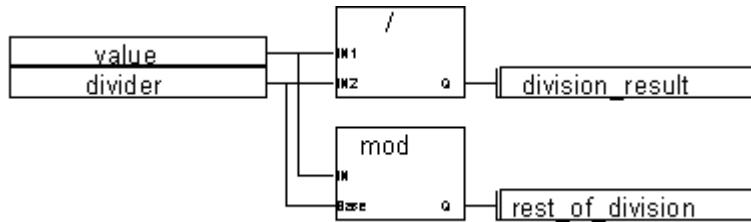
- | | | |
|------|------|--|
| IN | DINT | Any signed INTEGER value |
| Base | DINT | Must be greater than zero |
| Q | DINT | Modulo calculation (input MOD base) returns -1 if Base <= 0 |

Description:

Calculates the modulo of an integer value.

Example

(* FBD Program using "MOD" Function *)



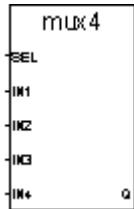
(* ST Equivalence: *)

```
division_result := (value / divider); (* integer division *)
rest_of_division := MOD (value, divider); (* rest of the division *)
```

(* IL Equivalence: *)

```
LD      value
DIV    divider
ST      division_result
LD      value
MOD    divider
ST      rest_of_division
```

MUX4



Arguments:

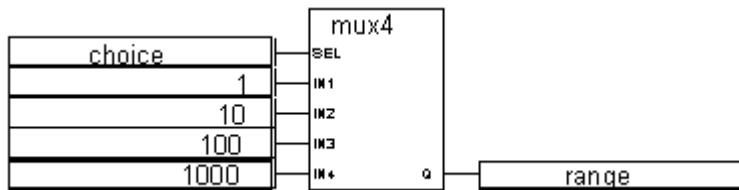
| | | |
|----------|------|--|
| SEL | DINT | Selector integer value (must be in set [0..3]) |
| IN1..IN4 | DINT | Any integer values |
| Q | DINT | = value1 if SEL = 0 = value2 if SEL = 1 = value3 if SEL = 2 = value4 if SEL = 3 = 0 for all other values of the selector |

Description:

Multiplexer with four entries: selects a value between four integer values.

Example

(* FBD Program using "MUX4" Function *)



(* ST Equivalence: *)

```
range := MUX4 (choice, 1, 10, 100, 1000);
```

(* select from 4 predefined ranges, for example, if choice is 1, range will be 10 *)

(* IL Equivalence: *)

```
LD      choice
MUX4   1,10,100,1000
ST      range
```

MUX8



Arguments:

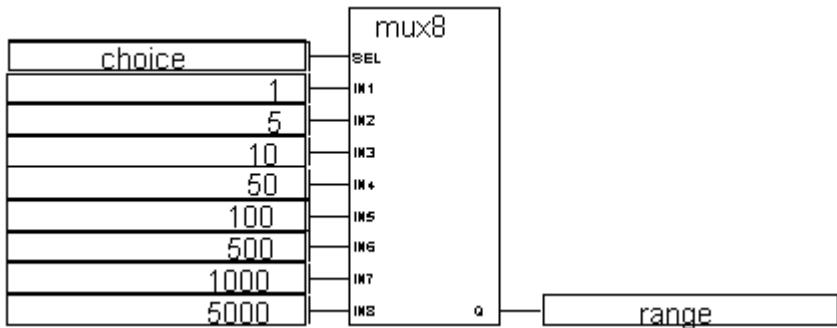
| | | |
|----------|------|---|
| SEL | DINT | Selector integer value (must be in set [0..7]) |
| IN1..IN8 | DINT | Any integer values |
| Q | DINT | = value1 if selector = 0 = value2 if selector = 1 ... = value8 if selector = 7 = 0 for all other values of the selector |

Description:

Multiplexer with eight entries: selects a value between eight integer values.

Example

(* FBD Program using "MUX8" Function *)



(* ST Equivalence: *)

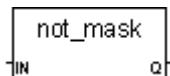
```
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
```

(* select from 8 predefined ranges, for example, if choice is 3, range will be 50 *)

(* IL Equivalence: *)

```
LD      choice
MUX8   1,5,10,50,100,500,1000,5000
ST      range
```

NOT_MASK



Arguments:

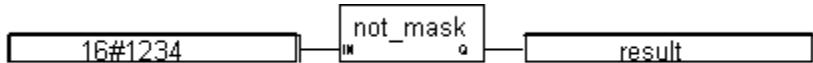
IN DINT Must have integer format
Q DINT Bit-to-bit negation on 32 bits of IN

Description:

Integer bit-to-bit negation mask.

Example

(* FBD example with NOT_MASK Operators *)



(*ST equivalence: *)

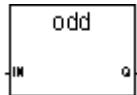
```
result := NOT_MASK (16#1234);
```

(* result is 16#FFFF_EDCB *)

(* IL equivalence: *)

```
LD      16#1234
NOT_MASK
ST      result
```

ODD



Arguments:

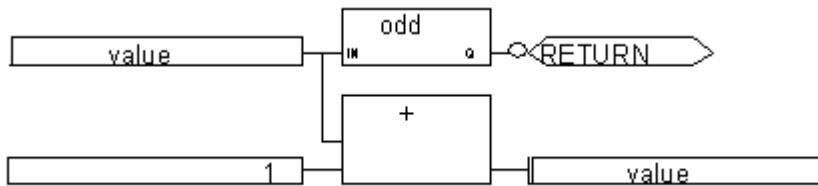
| | | |
|----|------|--|
| IN | DINT | Any signed integer value |
| Q | BOOL | TRUE if input value is odd FALSE if input value is even |

Description:

Tests the parity of an integer: result is odd or even.

Example

(* FBD Program using "ODD" Function *)



(* ST Equivalence: *)

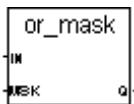
```
If Not (ODD (value)) Then Return; End_if;  
value := value + 1;
```

(* makes value always even *)

(* IL Equivalence: *)

```
LD      value  
ODD  
RETNC  
LD      value  
ADD    1  
ST      value
```

OR_MASK



Arguments:

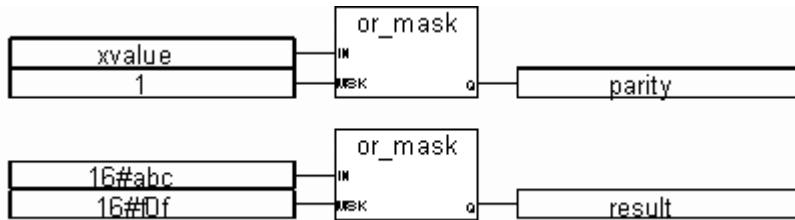
- | | | |
|-----|------|---|
| IN | DINT | Must have integer format |
| MSK | DINT | Must have integer format |
| Q | DINT | Bit-to-bit logical OR between IN and MSK |

Description:

Integer OR bit-to-bit mask.

Example

(* FBD example with OR_MASK Operators *)



(* ST Equivalence: *)

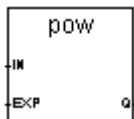
```
parity := OR_MASK (xvalue, 1); (* makes value always odd *)
result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)
```

(* IL equivalence: *)

```
LD      xvalue
OR_MASK 1
ST      parity
```

```
LD      16#abc
OR_MASK 16#f0f
ST      result
```

POW



Arguments:

IN REAL Real number to be raised

EXP REAL Power (exponent)

Q REAL (IN EXP)

1.0 if IN is not 0.0 and EXP is 0.0

0.0 if IN is 0.0 and EXP is negative

0.0 if both IN and EXP are 0.0

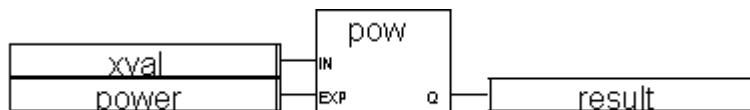
0.0 if IN is negative and EXP does not correspond to an integer

Description:

Gives the real result of the operation: (base ^{exponent}) 'base' being the first argument and 'exponent' the second one. The exponent is a real value.

Example

(* FBD Program using "POW" Function *)

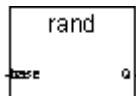


(* ST Equivalence: *)
result := POW (xval, power);

(* IL Equivalence: *)

LD xval
POW power
ST result

RAND



Arguments:

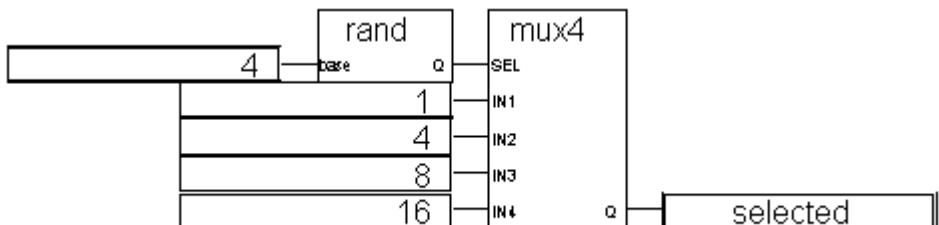
base DINT Defines the allowed set of number
Q DINT Random value in set [0..base-1]

Description:

Gives a random integer value in a given range.

Example

(* FBD Program using "RAND" function *)

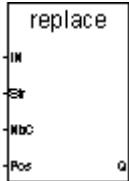


```

(* ST Equivalence: *)
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
(*
random selection of 1 of 4 pre-defined values
the value issued of RAND call is in set [0..3],
so 'selected' issued from MUX4, will get 'randomly' the value
1 if 0 is issued from RAND,
or 4 if 1 is issued from RAND,
or 8 if 2 is issued from RAND,
or 16 if 3 is issued from RAND,
*)

```

REPLACE



Arguments:

IN STRING Any string

Str STRING String to be inserted (to replace NbC chars)

NbC DINT Number of characters to be deleted

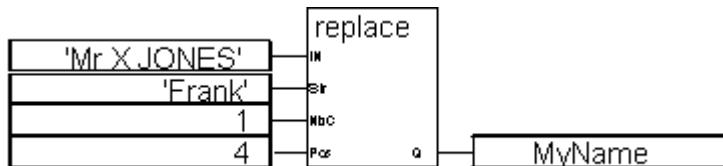
| | | |
|-----|--------|---|
| Pos | DINT | Position of the first modified character (first valid position is 1) |
| Q | STRING | <p>Modified string:</p> <ul style="list-style-type: none"> - NbC characters are deleted at position Pos - then substring Str is inserted at this position <p>returns empty string if Pos <= 0</p> <p>returns strings concatenation (IN+Str) if Pos is greater than the length of the IN string</p> <p>returns initial string IN if NbC <= 0</p> |

Description:

Replaces a part of a string by a new set of characters.

Example

Replaces a part of a string by a new set of characters.



(* ST Equivalence: *)

```
MyName := REPLACE ('Mr X JONES', 'Frank', 1, 4);
```

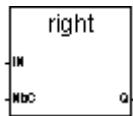
(* MyName is 'Mr Frank JONES' *)

(* IL Equivalence: *)

```

LD      'Mr X JONES'
REPLACE 'Frank', 1, 4
ST      MyName
  
```

RIGHT



Arguments:

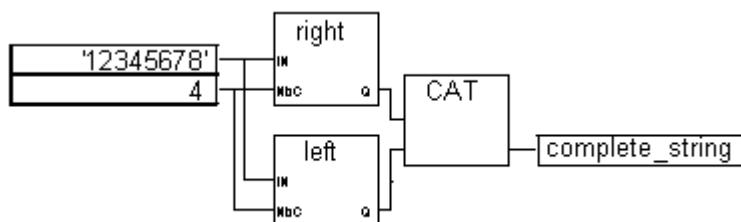
| | | |
|-----|--------|--|
| IN | STRING | Any non-empty string |
| NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| Q | STRING | Right part of the string (length = NbC) empty string if NbC <= 0 complete string if NbC >= string length |

Description:

Extracts the right part of a string. The number of characters to be extracted is given.

Example

(* FBD Program using "LEFT" and "RIGHT" Functions *) (* ST Equivalence: *)



```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

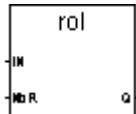
the value issued from LEFT call is '1234'

*)

(* IL Equivalence: First done is call to LEFT *)

```
LD      '12345678'  
LEFT    4  
ST      sub_string (* intermediate result *)  
LD      '12345678'  
RIGHT   4  
ADD    sub_string  
ST      complete_string
```

ROL

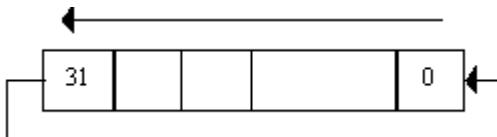


Arguments:

| | | |
|-----|------|---|
| IN | DINT | Any integer value |
| NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| Q | DINT | Left rotated value no effect if NbR <= 0 |

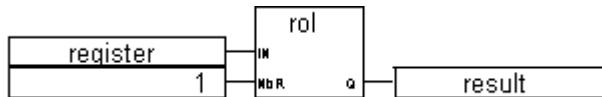
Description:

Make the bits of an integer rotate to the left. Rotation is made on 32 bits:



Example

(* FBD Program using "ROL" Function *)



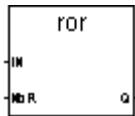
(* ST Equivalence: *)

```
result := ROL (register, 1);  
(* register = 2#0100_1101_0011_0101*)  
(* result = 2#1001_1010_0110_1010*)
```

(* IL Equivalence: *)

```
LD      register  
ROL    1  
ST      result
```

ROR

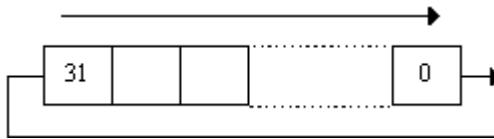


Arguments:

| | | |
|-----|------|--|
| IN | DINT | Any integer value |
| NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| Q | DINT | Right rotated value no effect if NbR <= 0 |

Description:

Make the bits of an integer rotate to the right. Rotation is made on 32 bits:



Example

(* FBD Program using "ROR" Function *)



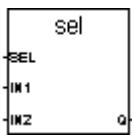
(* ST Equivalence: *)

```
result := ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1010_0110_1001_1010 *)
```

(* IL Equivalence: *)

```
LD      register
ROR    1
ST      result
```

SEL



Arguments:

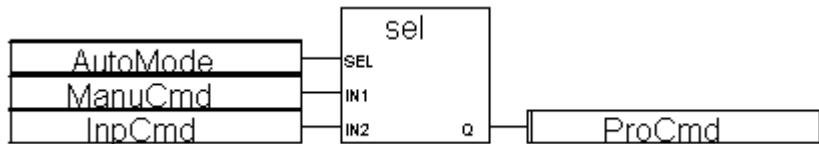
| | | |
|----------|------|---|
| SEL | BOOL | Indicates the chosen value |
| IN1, IN2 | DINT | Any integer values |
| Q | DINT | = IN1 if SEL is FALSE = IN2 if SEL is TRUE |

Description:

Binary selector: selects a value between two integer values.

Example

(* FBD Program using "SEL" Function *)



(* ST Equivalence: *)

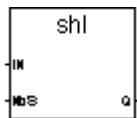
```
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
```

(* process command selection *)

(* IL Equivalence: *)

```
LD      AutoMode
SEL    ManuCmd, InpCmd
ST     ProCmd
```

SHL

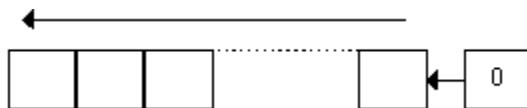


Arguments:

| | | |
|-----|------|--|
| IN | DINT | Any integer value |
| NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| Q | DINT | Left shifted value no effect if NbS <= 0 0 is used to replace lowest bit |

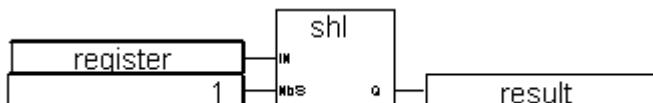
Description:

Make the bits of an integer shift to the left. Shift is made on 32 bits:



Example

(* FBD Program using "SHL" Function *)



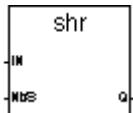
(* ST Equivalence: *)

```
result := SHL (register,1);  
(* register = 2#0100_1101_0011_0101 *)  
(* result = 2#1001_1010_0110_1010 *)
```

(* IL Equivalence: *)

```
LD      register
SHL     1
ST      result
```

SHR

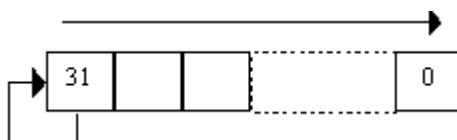


Arguments:

| | | |
|-----|------|---|
| IN | DINT | Any integer value |
| NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| Q | DINT | Right shifted value no effect if NbS <= 0 highest bit is copied at each shift |

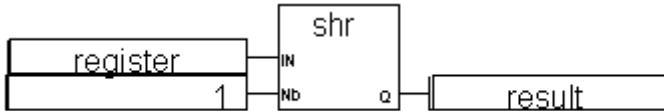
Description:

Make the bits of an integer shift to the right. Shift is made on 32 bits:



Example

(* FBD Program using "SHR"Function *)



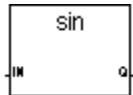
(* ST Equivalence: *)

```
result := SHR (register,1);  
(* register = 2#1100_1101_0011_0101 *)  
(* result = 2#1110_0110_1001_1010 *)
```

(* IL Equivalence: *)

```
LD      register  
SHR     1  
ST      result
```

SIN



Arguments:

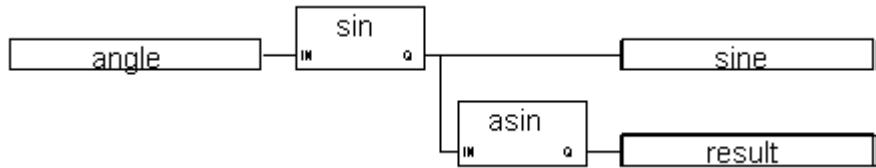
IN REAL Any REAL value
Q REAL Sine of the input value (in set [-1.0 .. +1.0])

Description:

Calculates the Sine of a real value.

Example

(* FBD Program using "SIN" and "ASIN" Functions *)



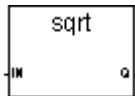
(* ST Equivalence: *)

```
sine := SIN (angle);  
result := ASIN (sine); (* result is equal to angle *)
```

(* IL Equivalence: *)

```
LD      angle  
SIN  
ST      sine  
ASIN  
ST      result
```

SQRT



Arguments:

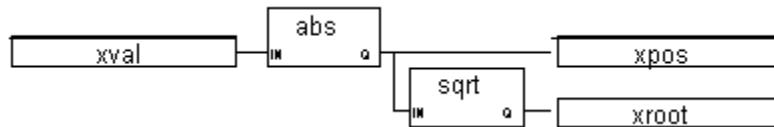
IN REAL Must be greater than or equal to zero
Q REAL Square root of the input value

Description:

Calculates the square root of a real value.

Example

(* FBD Program using "SQRT" Function *)



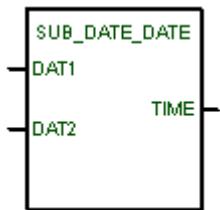
(* ST Equivalence: *)

```
xpos := ABS (xval);  
xroot := SQRT (xpos);
```

(* IL Equivalence: *)

```
LD      xval  
ABS  
ST      xpos  
SQRT  
ST      xroot
```

SUB_DATE_DATE



Arguments:

DAT1 DATE First date in a comparison

| | | |
|------|------|---|
| DAT2 | DATE | Second date in a comparison |
| TIME | TIME | <p>Difference in TIME format between DAT1 and DAT2. The possible date difference values range from t#0h to t#1193h2m47s294ms inclusively.</p> <p>A value of 1193h2m47s295ms indicates an error for either of the following conditions:</p> <ul style="list-style-type: none"> - DAT1 is less than DAT2 - The difference between DAT1 and DAT2 is greater than 1193h2m47s294ms |

Description:

Compares two dates and gives the difference in TIME format.

Example

(* FBD Program using "SUB_DATE_DATE" Function *)



(* ST Equivalence: *)

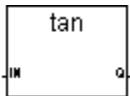
```
timResult := SUB_DATE_DATE (datVal1, datVal2);
```

(* IL Equivalence: *)

```

LD          datVal1
SUB_DATE_DATE datVal2
ST          timResult
  
```

TAN



Arguments:

IN REAL Cannot be equal to PI/2 modulo PI

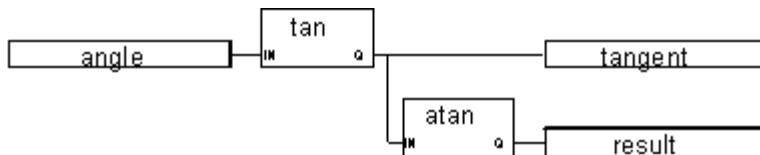
Q REAL Tangent of the input value
= 1E+38 for invalid input

Description:

Calculates the Tangent of a real value.

Example

(* FBD Program using "TAN" and "ATAN" Functions *)



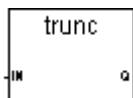
(* ST Equivalence: *)

```
tangent := TAN (angle);  
result := ATAN (tangent); (* result is equal to angle*)
```

(* IL Equivalence: *)

```
LD      angle  
TAN  
ST      tangent  
ATAN  
ST      result
```

TRUNC



Arguments:

IN REAL Any REAL value

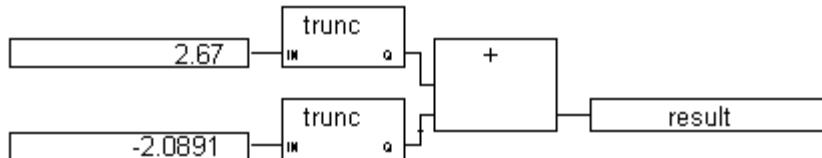
Q REAL If IN>0, biggest integer less or equal to the input
If IN<0, least integer greater or equal to the input

Description:

Truncates a real value to have just the integer part.

Example

(* FBD Program using "TRUNC" Function *)



(* ST Equivalence: *)

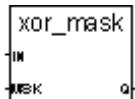
```
result := TRUNC (+2.67) + TRUNC (-2.0891);  
(* means: result := 2.0 + (-2.0) := 0.0; *)
```

(* IL Equivalence: *)

```
LD      2.67  
TRUNC  
ST      temporary    (* temporary result of first TRUNC *)  
LD      -2.0891  
TRUNC
```

```
ADD      temporary
ST       result
```

XOR_MASK



Arguments:

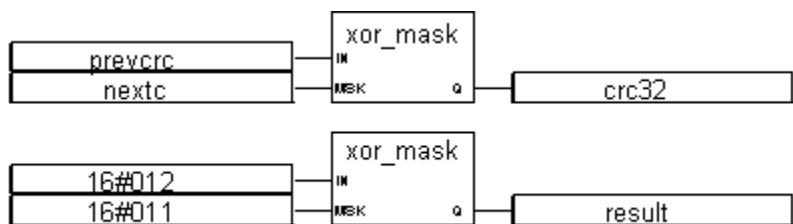
| | | |
|-----|------|---|
| IN | DINT | Must have integer format |
| MSK | DINT | Must have integer format |
| Q | DINT | Bit-to-bit logical Exclusive OR between IN and MSK |

Description:

Integer exclusive OR bit-to-bit mask

Example

(* FBD example with XOR_MASK Operators *)



(* ST Equivalence: *)

```
crc32 := XOR_MASK (prevcrc, nextc);
result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)
```

(* IL equivalence: *)

| | |
|----------|---------|
| LD | prevcrc |
| XOR_MASK | nextc |
| ST | crc32 |
| LD | 16#012 |
| XOR_MASK | 16#011 |
| ST | result |

Standard Function Blocks

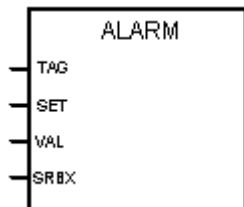
DS800 Development Suite supports the following standard function blocks:

| | | |
|--|------------------|--|
| Alarm and event operations | ALARM | Places a time-stamped alarm in the alarm log of a device. |
| Warning: These function blocks are available only on ROC800-Series devices. | EVENT | Places a time-stamped event in the event log of a device. |
| Boolean operations | SR | Set dominant bistable |
| | RS | Reset dominant bistable |
| | R_TRIGGER | Rising edge detection |
| | F_TRIGGER | Falling edge detection |
| Communications | CONNECT | Connection to a resource |
| | USEND_S | Sending of a message to a resource |
| | URCV_S | Reception of a message from a resource |
| Comparator | CMP | Full comparison Function Block |
| Counters | CTU | Up counter |
| | CTD | Down counter |
| | CTUD | Up-down counter |
| Database commands | DBG_CLR_GET_ERR | Clears error set by TLP_GET_xxx functions |
| Warning: These function blocks are available only on ROC800-Series devices. | DBG_CLR_SET_ERR | Clears error set by TLP_SET_xxx functions |
| | DBG_GET_ERR | Indicates error with TLP_GET_xxx functions |
| | DBG_SET_ERR | Indicates error with TLP_SET_xxx functions |
| | SOFT_POINT_READ | Reads entire contents of a soft point and writes the data to an output value |
| | SOFT_POINT_WRITE | Writes data to a specified soft point |
| | TLP_GET_DINT | Returns TLP data as a DINT |

| | | |
|--|-----------------|---|
| | TLP_GET_REAL | Returns TLP data as a REAL |
| | TLP_GET_SINT | Returns TLP data as an SINT |
| | TLP_GET_STRING | Returns TLP data as a STRING |
| | TLP_GET_TLP | Reads the TLP from a specified TLP |
| | TLP_SET_DINT | Writes a DINT to a TLP |
| | TLP_SET_REAL | Writes a REAL to a TLP |
| | TLP_SET_SINT | Writes a SINT to a TLP |
| | TLP_SET_STRING | Writes a STRING to a TLP |
| Data manipulation | AVERAGE | Running average over N samples |
| License request and operating system priority | REQUEST_LICENSE | Requests a license from the license server. |
| | SET_PRIORITY | Sets the priority of the operating system. |
| Warning: These function blocks are available only on ROC800-Series devices. | | |
| Process control | DERIVATE | Differentiation according to time |
| | HYSTER | Boolean hysteresis on difference of reals |
| | INTEGRAL | Integration over time |
| | STACKINT | Stack of integer |
| Signal generation | BLINK | Blinking Boolean signal |
| | SIG_GEN | Signal generator |
| | TON | On-delay timing |
| | TOF | Off-delay timing |
| | TP | Pulse timing |

Note: When new function blocks are created, these can be called from any language.

ALARM



Warning: This function block is available only on ROC800-Series devices.

Arguments:

| | | |
|------|--------|--|
| TAG | STRING | A brief textual description of the alarm |
| SET | BOOL | TRUE enables alarm, FALSE disables alarm |
| VAL | REAL | Numerical value associated with the alarm |
| SRBX | BOOL | Alarm is a spontaneous report by exception alarm |

Description:

The ALARM function places a time stamped alarm in the alarm log of the device. Setting the SET input to TRUE logs an “alarm set”, whereas, setting this input to false logs an “alarm clear”. The SRBX input allows for the device to contact a host to report an alarm condition. Further setup is required in ROCLINK if this input is set to TRUE.

AVERAGE



Arguments:

| | | |
|-----|------|---------------------------------------|
| RUN | BOOL | TRUE=run / FALSE=reset |
| XIN | REAL | Any real Variable |
| N | DINT | Application defined number of samples |

XOUT REAL Running average of XIN value

Note: When setting or changing the value for N, you need to set RUN to FALSE, then set it back to TRUE.

Description:

Stores a value at each cycle and calculates the average value of all already stored values. Only the N last values are stored.

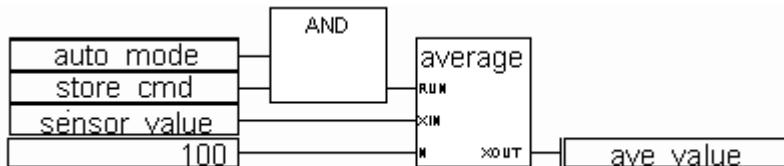
The number of samples N cannot exceed **128**.

If the "RUN" command is **FALSE** (reset mode), the output value is equal to the input value.

When the maximum N of stored values is reached, the first stored value is erased by the last one.

Example

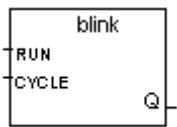
(* FBD Program using "AVERAGE" Block: *)



(* ST Equivalence: AVERAGE1 instance of AVERAGE block *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);  
ave_value := AVERAGE1.XOUT;
```

BLINK



Arguments:

RUN BOOL Mode: TRUE=blinking / FALSE=reset the output to false

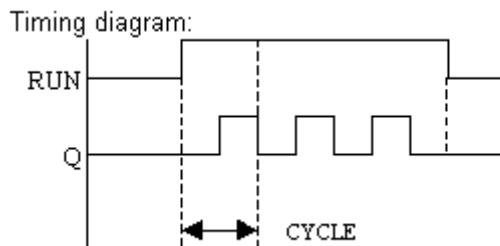
CYCLE TIME Blinking period. Possible values range from 0ms to 23h59m59s999ms.

Q BOOL Output blinking signal

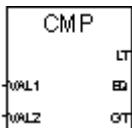
Description:

Generates a blinking signal.

Timing diagram:



CMP



Arguments:

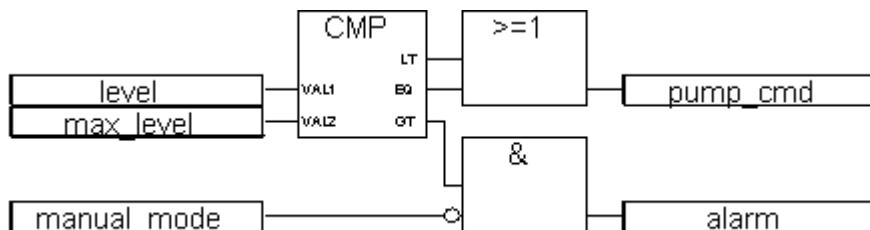
| | | |
|------|------|-----------------------------------|
| VAL1 | DINT | Any signed integer value |
| VAL2 | DINT | Any signed integer value |
| LT | BOOL | TRUE if val1 is Less Than val2 |
| EQ | BOOL | TRUE if val1 is Equal to val2 |
| GT | BOOL | TRUE if val1 is Greater Than val2 |

Description:

Compare two values: tell if they are equal, or if the first is less or greater than the second one.

Example

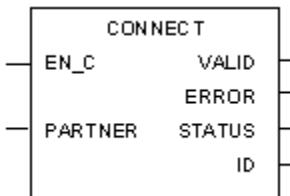
(* FBD Program using "CMP" Block *)



(* ST Equivalence: We suppose CMP1 is an instance of CMP block *)

```
CMP1(level, max_level);  
pump_cmd := CMP1.LT OR CMP1.EQ;  
alarm := CMP1.GT AND NOT(manual_mode);
```

CONNECT



Arguments:

| | | |
|---------|--------|--|
| EN_C | BOOL | Enable connection. |
| PARTNER | STRING | Name of the remote communication partner. |
| VALID | BOOL | If TRUE, connection ID is valid. |
| ERROR | BOOL | If TRUE, new non-zero status received. |
| STATUS | DINT | Last detected status. |
| ID | DINT | Identification of the communication Channel. |

Description:

Creates a connection with a remote or local Resource (of current Project or another Project) and manages the exchanges (for blocks USEND_S and URCV_S). For details on the USEND_S block, see page 613. For details on the URCV_S block, see page 612.

It creates a communication channel identifier (ID).

This identifier is required in all others communication function blocks (URCV_S or USEND_S).

PARTNER parameter is a string with the following format:

'ResourceNumber@Address'

Example

'1@123.45.67.89'

Connection with the ETCP driver to Resource 1 at address 123.45.67.89.

If the Resource is on the same Configuration, its number is enough to identify it (e.g. '1').

On a rising edge of EN_C parameter, the CONNECT Block establishes the communication with the remote partner.

The VALID parameter is set to TRUE until the communication is available.

Every time the status changes, the output parameter ERROR is set to TRUE during one cycle and the new status is set in the STATUS parameter.

STATUS can take following values:

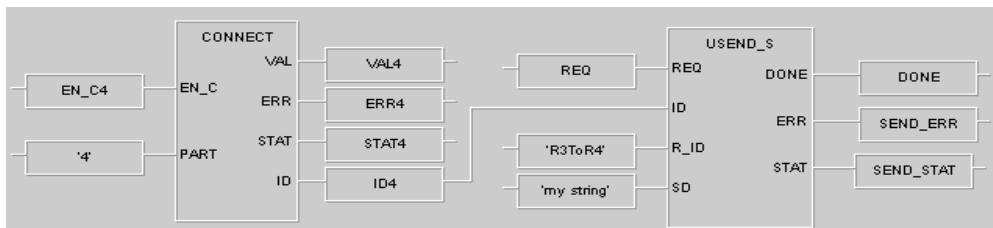
| STATUS | Description |
|--------|------------------------------------|
| 0 | Connection successfully completed. |
| 1 | Waiting for reply |
| 2 | Too many CFB connect |
| 3 | Not ready for a new connection |
| 4 | Connect failed |
| 5 | Bad partner |

- | STATUS | Description |
|--------|------------------------------------|
| 0 | Connection successfully completed. |
| 1 | Waiting for reply |
| 2 | Too many CFB connect |
| 3 | Not ready for a new connection |
| 4 | Connect failed |
| 5 | Bad partner |

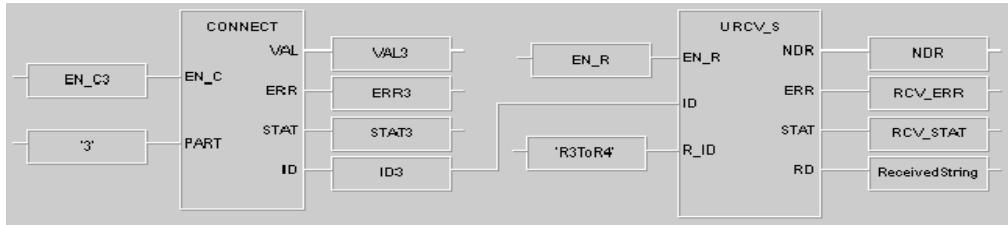
If the connection failed, a new connection is not automatically done, a rising edge must be detected on EN_C parameter.

Example

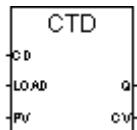
The following is a program of Resource 3 that sends a string to Resource 4 on the same Configuration:



The following is the corresponding program in Resource 4 that receives the string:



CTD



Arguments:

| | | |
|------|------|--|
| CD | BOOL | Counting input (down-counting when CD is TRUE) |
| LOAD | BOOL | Load command (dominant) (CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed initial value |
| Q | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

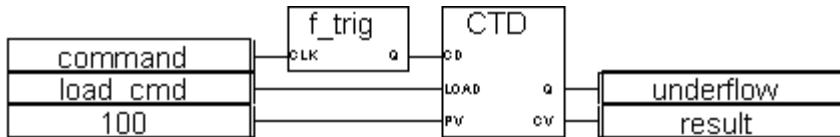
Warning: The CTD Block does not detect the rising or falling edges of the counting input (CD). It must be associated with an "R_TRIG" or "F_TRIG" block to create a pulse counter.

Description:

Count (integer) from a given value down to 0 1 by 1

Example

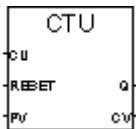
(* FBD Program using "CTD" Block *)



(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block and CTD1 is an instance of CTD block*)

```
F_TRIG1(command);  
CTD1(F_TRIG1.Q, load_cmd, 100);  
underflow := CTD1.Q;  
result := CTD1.CV;
```

CTU



Arguments:

| | | |
|-------|------|---|
| CU | BOOL | Counting input (counting when CU is TRUE) |
| RESET | BOOL | Reset command (dominant) |
| PV | DINT | Programmed maximum value |
| Q | BOOL | Overflow: TRUE when CV >= PV |
| CV | DINT | Counter result |

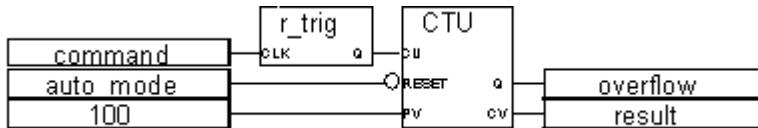
Warning: The CTU Block does not detect the rising or falling edge of the counting input (CU). It must be associated with an "R_TRIG" or "F_TRIG" block to create a pulse counter.

Description:

Count (integer) from 0 up to a given value 1 by 1

Example

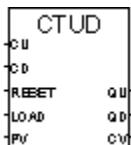
(* FBD Program using "CTU" Block *)



(* ST Equivalence: We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)

```
R_TRIG1(command);  
CTU1(R_TRIG1.Q,NOT(auto_mode),100);  
overflow := CTU1.Q;  
result := CTU1.CV;
```

CTUD



Arguments:

| | | |
|-------|------|---|
| CU | BOOL | Up-counting (when CU is TRUE) |
| CD | BOOL | Down-counting (when CD is TRUE) |
| RESET | BOOL | Reset command (dominant) (CV = 0 when RESET is TRUE) |
| LOAD | BOOL | Load command (CV = PV when LOAD is TRUE) |

| | | |
|----|------|------------------------------|
| PV | DINT | Programmed maximum value |
| QU | BOOL | Overflow: TRUE when CV >= PV |
| QD | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

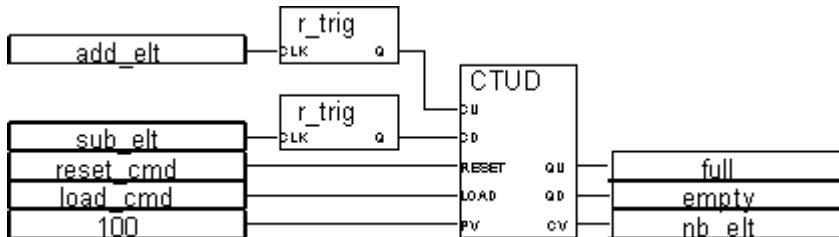
Warning: The CTUD Block does not detect the rising or falling edge of the counting inputs (CU and CD). It must be associated with an "R_TRIG" or "F_TRIG" Block to create a pulse counter.

Description:

Count (integer) from 0 up to a given value 1 by 1 or from a given value down to 0 1 by 1

Example

(* FBD Program using "CTUD" Block *)



(* ST Equivalence: We suppose R_TRIG1 and R_TRIG2 are two instances of R_TRIG Block and CTUD1 is an instance of CTUD block*)

```

R_TRIG1(add_elt);
R_TRIG2(sub_elt);
CTUD1(R_TRIG1.Q, R_TRIG2.Q, reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
  
```

DBG_CLR_GET_ERR

`DBG_CLR_GET_ERR`

Warning: This function block is available only on ROC800-Series devices.

Arguments:

This function has no arguments.

Description:

This function clears the errors that were encountered with any of the TLP_GET_xxx functions.

DBG_CLR_SET_ERR

`DBG_CLR_SET_ERR`

Warning: This function block is available only on ROC800-Series devices.

Arguments:

This function has no arguments.

Description:

This function clears the errors that were encountered with any of the TLP_SET_xxx functions.

DBG_GET_ERR



Warning: This function block is available only on ROC800-Series devices.

Arguments:

status (stat) BOOL TRUE=error/FALSE=no error

Description:

This function is intended as an aid for debugging your DS800 application. Returns TRUE if an error has been encountered with any of the TLP_GET_xxx functions. Errors occur if the TLP does not exist, or if the TLP references a data type that is incompatible with the return type of the function. The error remains set until DBG_CLR_GET_ERR is called.

DBG_SET_ERR



Warning: This function block is available only on ROC800-Series devices.

Arguments:

status (stat) BOOL TRUE=error/FALSE=no error

Description:

This function is intended as an aid for debugging your DS800 application. Returns TRUE if an error has been encountered with any of the TLP_SET_xxx functions. Errors occur if the TLP does not exist, or if the TLP references a data type that is incompatible with the data type of the value it is being set to. The error remains set until DBG_CLR_SET_ERR is called.

DERIVATE



Arguments:

| | | |
|-------|------|--|
| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| XIN | REAL | Input: any real value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

Description:

Differentiation of a real value.

If the "CYCLE" parameter value is less than the cycle timing of the execution of the resource in the target, the sampling period is forced to this cycle timing.

Example

(* FBD Program using "DERIVATE" Block: *)



(* ST Equivalence: DERIVATE1 instance of DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);  
derived_value := DERIVATE1.XOUT;
```

EVENT



Warning: This function block is available only on ROC800-Series devices.

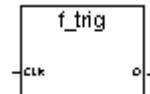
Arguments:

TAG STRING A brief textual description of the event

Description:

The EVENT function places a time-stamped user event in the event log of the device.

F_TRIG



Arguments:

CLK BOOL Any Boolean Variable

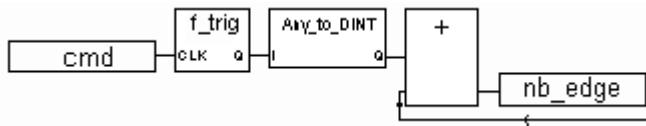
Q BOOL TRUE when CLK changes from TRUE to FALSE
 FALSE if all other cases

Description:

Detects a falling edge of a Boolean Variable

Example

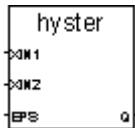
(* FBD Program using "F_TRIG" Block *)



(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block *)

```
F_TRIG1(cmd);  
nb_edge := ANY_TO_DINT(F_TRIG1.Q) + nb_edge;
```

HYSTER



Arguments:

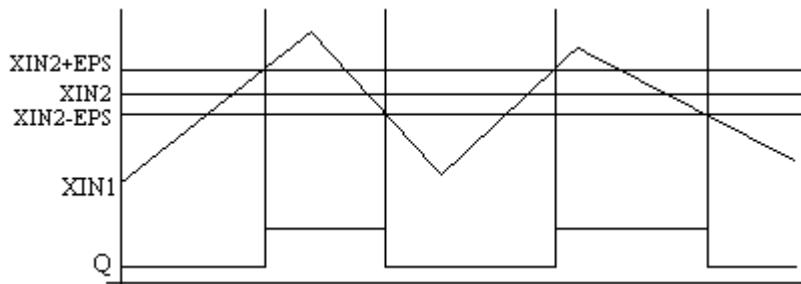
| | | |
|------|------|--|
| XIN1 | REAL | Any real value |
| XIN2 | REAL | To test if XIN1 has overpassed XIN2+EPS |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| Q | BOOL | TRUE if XIN1 has overpassed XIN2+EPS and is not yet below XIN2-EPS |

Description:

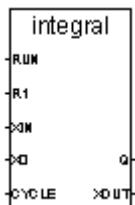
Hysteresis on a real value for a high limit.

Example

Example of a timing diagram:



INTEGRAL



Arguments:

| | | |
|-------|------|--|
| RUN | BOOL | Mode: TRUE=integrate / FALSE=hold |
| R1 | BOOL | Overriding reset |
| XIN | REAL | Input: any real value |
| X0 | REAL | Initial value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| Q | BOOL | Not R1 |
| XOUT | REAL | Integrated output |

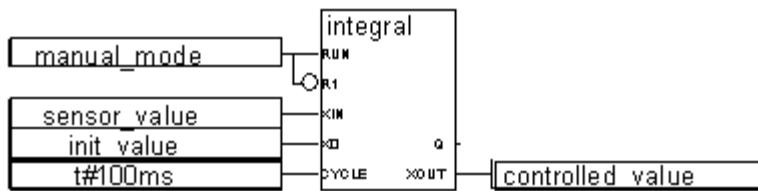
Description:

Integration of a real value.

If the "CYCLE" parameter value is less than the cycle timing of the execution of the resource in the target, the sampling period is forced to this cycle timing.

Example

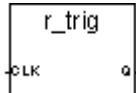
(* FBD Program using "INTEGRAL" Block: *)



(* ST Equivalence: INTEGRAL1 instance of INTEGRAL block *)

```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value,  
t#100ms);  
  
controlled_value := INTEGRAL1.XOUT;
```

R_TRIG



Arguments:

CLK BOOL Any Boolean Variable

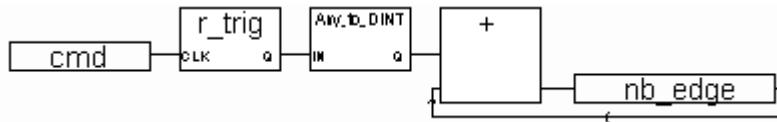
Q BOOL TRUE when CLK rises from FALSE to TRUE
FALSE in all other cases

Description:

Detects a Rising Edge of a Boolean Variable

Example

(* FBD Program using "R_TRIG" Block *)



(* ST Equivalence: We suppose R_TRIG1 is an instance of R_TRIG Block *)

```
R_TRIG1(cmd);  
nb_edge := ANY_TO_DINT(R_TRIG1.Q) + nb_edge;
```

REQUEST_LICENSE



Warning: This function block is available only on ROC800-Series devices.

Arguments:

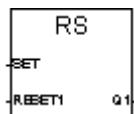
| | | |
|------|--------|--|
| name | STRING | Name of application to request license for |
| vid | DINT | Vendor ID |

| | | |
|------|-------------|------------------------------------|
| cqry | DINT | Query options for application code |
| cd1 | DINT | Application code 1 |
| cd2 | DINT | Application code 2 |
| vqry | DINT | Query options for version |
| ver1 | STRING | Version 1 |
| ver1 | STRING | Version 2 |
| lic | LicenseInfo | Identifies license returned |

Description:

The REQUEST_LICENSE function requests a license from the license server.

RS



Arguments:

| | | |
|--------|------|--|
| SET | BOOL | If TRUE, sets Q1 to TRUE |
| RESET1 | BOOL | If TRUE, resets Q1 to FALSE (dominant) |
| Q1 | BOOL | Boolean memory state |

Description:

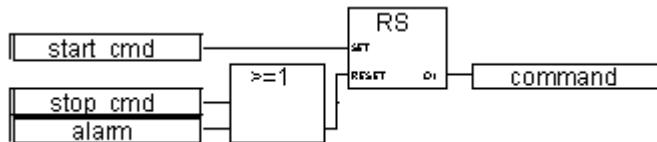
Reset dominant bistable:

| Set | Reset1 | Q1 | Result Q1 |
|-----|--------|----|-----------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Example

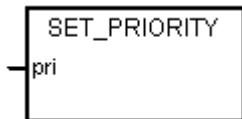
(* FBD Program using "RS" Block *)



(* ST Equivalence: We suppose RS1 is an instance of RS block *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

SET_PRIORITY



Warning: This function block is available only on ROC800-Series devices.

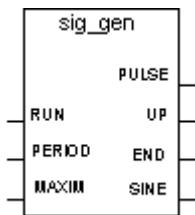
Arguments:

pri DINT Priority of the operating system. Possible values range between 10 and 31.

Description:

Sets the priority of the operating system.

SIG_GEN



Arguments:

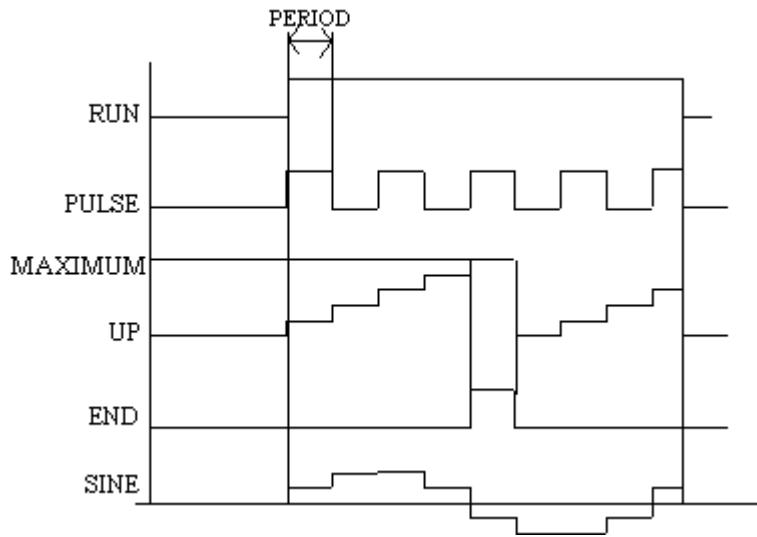
| | | |
|---------|------|---|
| RUN | BOOL | Mode: TRUE=running / FALSE=reset to false |
| PERIOD | TIME | Duration of one sample. Possible values range from 0ms to 23h59m59s999ms. |
| MAXIMUM | DINT | Maximum counting value |
| PULSE | BOOL | Inverted after each sample |
| UP | DINT | Up-counter, increased on each sample |
| END | BOOL | TRUE when up-counting ends |
| SINE | REAL | Sine signal (period = counting duration) |

Description:

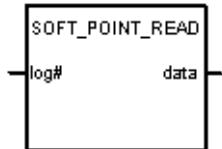
Generates various signal: blink on a boolean, a integer counter-up, and real sine wave.

When counting reaches maximum value, it restarts from 0 (zero). So END keeps the TRUE value only during 1 PERIOD.

Timing diagram:



SOFT_POINT_READ



Warning: This function block is available only on ROC800-Series devices.

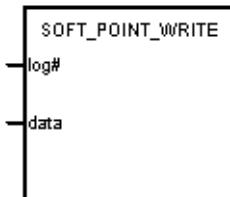
Arguments:

| | | |
|------------------|-----------|---|
| log_num (log#) | DINT | Identifies the logical number of a soft point to read from. |
| data | SoftPoint | Stores contents of specified soft point |

Description:

Reads entire contents of a soft point and writes the data to an output value.

SOFT_POINT_WRITE



Warning: This function block is available only on ROC800-Series devices.

Arguments:

| | | |
|------------------|-----------|--|
| log_num (log#) | DINT | Identifies the logical number of a soft point to write to. |
| data | SoftPoint | Contains data to be written to soft point. |

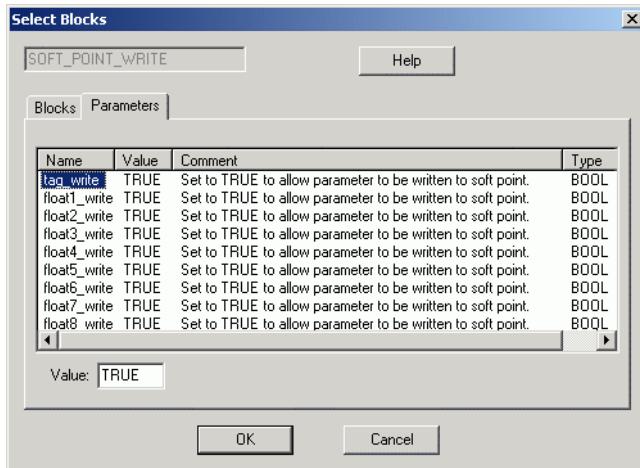
Description:

Writes data to a specified soft point.

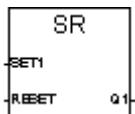
Hidden Parameters:

This function block has hidden parameters that correspond to each parameter in a soft point. These hidden parameters can be used to control which parameters in the input data actually get written to the soft point. The hidden parameters are of type BOOL. The default value for each parameter is TRUE, changing this value to FALSE prevents the corresponding soft point parameter from getting over-written with the input data.

Double-clicking the function block in the Function Block Diagram opens up a dialog box from which you access the hidden parameters. Click on the "Parameters" tab to display the hidden parameters.



SR



Arguments:

SET1 BOOL If TRUE, sets Q1 to TRUE (dominant)

RESET BOOL If TRUE, resets Q1 to FALSE

Q1 BOOL Boolean memory state

Description:

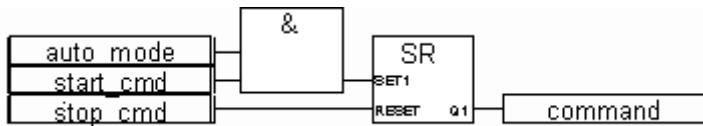
Set dominant bistable:

| Set1 | Reset | Q1 | Result Q1 |
|-------------|--------------|-----------|------------------|
| 0 | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Example

(* FBD Program using "SR" Block *)

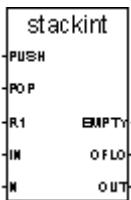


(* ST Equivalence: We suppose SR1 is an instance of SR block *)

```

SR1((auto_mode & start_cmd) , stop_cmd);
command := SR1.Q1;
  
```

STACKINT



Arguments:

| | | |
|-------|------|---|
| PUSH | BOOL | Push command (on Rising Edge only) add the IN value on the top of the stack |
| POP | BOOL | Pop command (on rising edge only) delete in the stack the last value pushed (top of the stack) |
| R1 | BOOL | Resets the stack to its empty state |
| IN | DINT | Pushed value |
| N | DINT | Application defined stack size |
| EMPTY | BOOL | TRUE if the stack is empty |
| OFLO | BOOL | Overflow: TRUE if the stack is full |
| OUT | DINT | Value at the top of the stack |

Description:

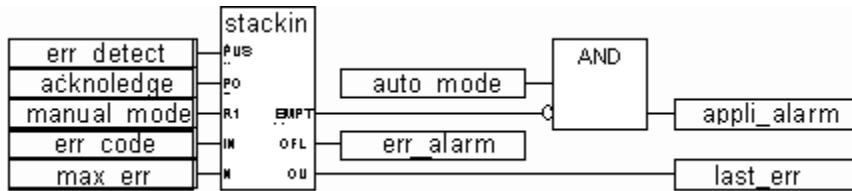
Manage a stack of integer values.

The "STACKINT" Function Block includes a rising edge detection for both PUSH and POP commands. The maximum size of the stack is **128**. The application defined stack size N cannot be less than 1 or greater than 128.

Note: OFLO value is valid only after a reset (R1 has been set to TRUE at least once and back to FALSE).

Example

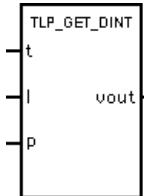
(* FBD Program using "STACKINT" Block: error management *)



(* ST Equivalence: We suppose STACKINT1 is an instance of STACKINT Block *)

```
STACKINT1(err_detect, acknoledges, manual_mode, err_code, max_err);  
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);  
err_alarm := STACKINT1.OFLO;  
last_error := STACKINT1.OUT;
```

TLP_GET_DINT



Warning: This function block is available only on ROC800-Series devices.

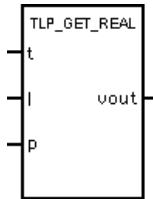
Arguments:

| | | |
|------------------|------|---|
| point_type (t) | DINT | Refers to point type that contains the parameter to be read |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to get |
| vout | DINT | Value of specified TLP |

Description:

Gets parameter referenced by a TLP. The specified TLP may be any numerical data type. Parameter is converted to a DINT before being written to the output value.

TLP_GET_REAL



Warning: This function block is available only on ROC800-Series devices.

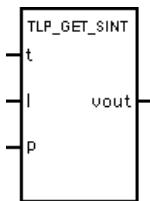
Arguments:

| | | |
|------------------|------|---|
| point_type (t) | DINT | Refers to point type that contains the parameter to be read |
| logical (1) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to get |
| vout | REAL | Value of specified TLP |

Description:

Gets parameter referenced by a TLP. The specified TLP may be any numerical data type. Parameter is converted to a REAL before being written to the output value.

TLP_GET_SINT



Warning: This function block is available only on ROC800-Series devices.

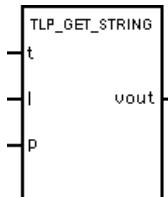
Arguments:

| | | |
|------------------|------|---|
| point_type (t) | DINT | Refers to point type that contains the parameter to be read |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to get |
| vout | SINT | Value of specified TLP |

Description:

Gets parameter referenced by a TLP. The specified TLP may be any numerical data type. Parameter is converted to a SINT before being written to the output value.

TLP_GET_STRING



Warning: This function block is available only on ROC800-Series devices.

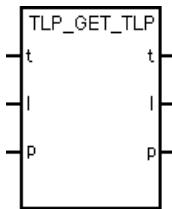
Arguments:

| | | |
|------------------|--------|---|
| point_type (t) | DINT | Refers to point type that contains the parameter to be read |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to get |
| vout | STRING | Value of specified TLP |

Description:

Gets parameter referenced by a TLP. The specified TLP must be a string. The string is truncated if it is too large to fit in the return value.

TLP_GET_TLP



Warning: This function block is available only on ROC800-Series devices.

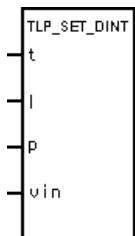
Arguments:

| | | |
|------------------|------|---|
| point_type (t) | DINT | Refers to point type that contains the parameter to be read |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to get |
| rpoint_type (t) | DINT | Specified point type |
| rlogical (l) | DINT | Specified instance of point type |
| rparameter (p) | DINT | Specified parameter |

Description:

Reads the TLP from a specified TLP.

TLP_SET_DINT



Warning: This function block is available only on ROC800-Series devices.

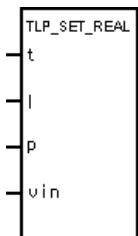
Arguments:

| | | |
|------------------|------|--|
| point_type (t) | DINT | Refers to point type that contains the parameter to be set |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to set |
| vin | DINT | New value for specified parameter |

Description:

Sets parameter referenced by a TLP. Any numerical data type can be set with this function. Vin is converted to the data type of specified TLP before parameter is set.

TLP_SET_REAL



Warning: This function block is available only on ROC800-Series devices.

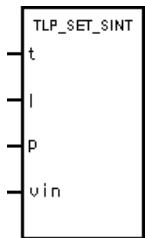
Arguments:

| | | |
|------------------|------|--|
| point_type (t) | DINT | Refers to point type that contains the parameter to be set |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to set |
| vin | REAL | New value for specified parameter |

Description:

Sets parameter referenced by a TLP. Any numerical data type can be set with this function. Vin is converted to the data type of specified TLP before parameter is set.

TLP_SET_SINT



Warning: This function block is available only on ROC800-Series devices.

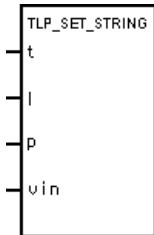
Arguments:

| | | |
|------------------|------|--|
| point_type (t) | DINT | Refers to point type that contains the parameter to be set |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to set |
| vin | SINT | New value for specified parameter |

Description:

Sets parameter referenced by a TLP. Any numerical data type can be set with this function. Vin is converted to the data type of specified TLP before parameter is set.

TLP_SET_STRING



Warning: This function block is available only on ROC800-Series devices.

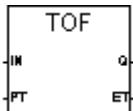
Arguments:

| | | |
|------------------|--------|--|
| point_type (t) | DINT | Refers to point type that contains the parameter to be set |
| logical (l) | DINT | Instance of specified point type |
| parameter (p) | DINT | Parameter to set |
| vin | STRING | New value for specified parameter |

Description:

Sets parameter referenced by a TLP. Any string type can be set with this function. String will be padded with spaces if vin is shorter than string being set. String will be truncated if vin is longer than string being set.

TOF



Arguments:

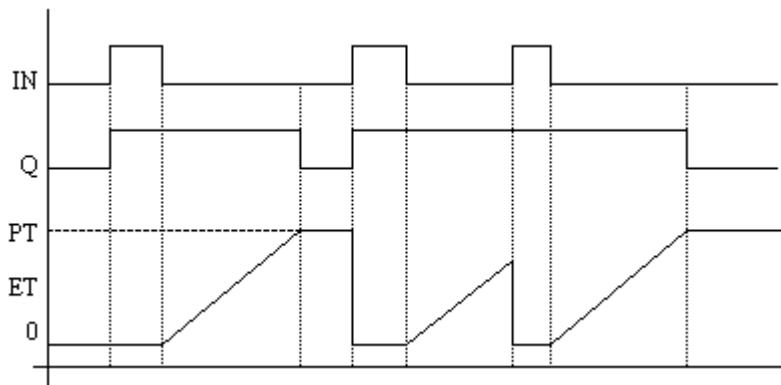
| | | |
|----|------|--|
| IN | BOOL | If Falling Edge, starts increasing internal timer If Rising Edge, stops and resets internal timer |
| PT | TIME | Maximum programmed time |

Q BOOL If TRUE: total time is not elapsed
ET TIME Current elapsed time

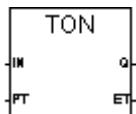
Description:

Increase an internal timer up to a given value.

Timing diagram:



TON



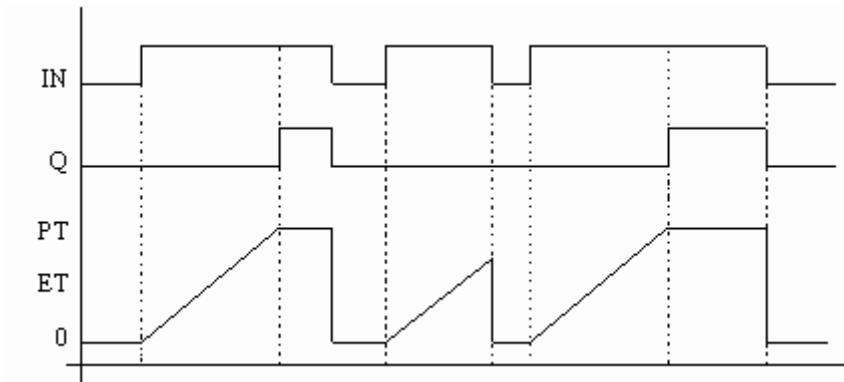
Arguments:

IN BOOL If Rising Edge, starts increasing internal timer
 If Falling Edge, stops and resets internal timer
PT TIME Maximum programmed time
Q BOOL If TRUE, programmed time is elapsed
ET TIME Current elapsed time. Possible values range from 0ms to 23h59m59s999ms.

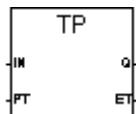
Description:

Increase an internal timer up to a given value.

Timing diagram:



TP



Arguments:

IN BOOL If Rising Edge, starts increasing internal timer (if not already increasing)
 If FALSE and only if timer is elapsed, resets the internal timer
 Any change on IN during counting has no effect.

PT TIME Maximum programmed time

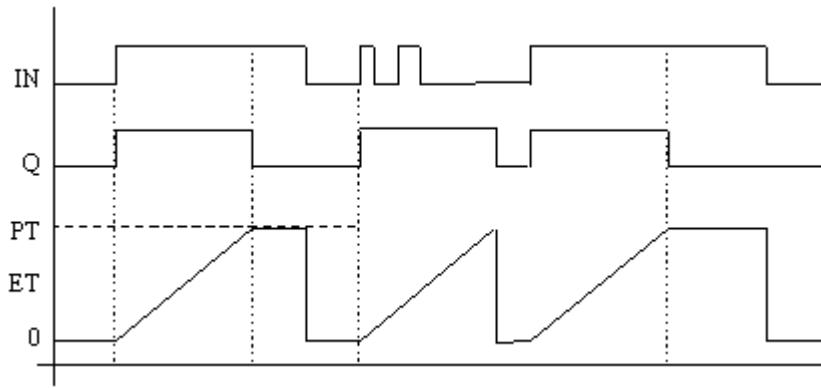
Q BOOL If TRUE: timer is counting

ET TIME Current elapsed time. Possible values range from 0ms to 23h59m59s999ms.

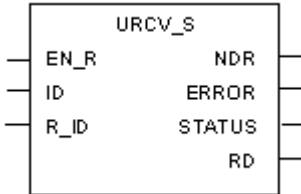
Description:

Increase an internal timer up to a given value.

Timing diagram:



URCV_S



Arguments:

| | | |
|--------|--------|---|
| EN_R | BOOL | Enable to receive data |
| ID | DINT | Identification of the communication Channel |
| R_ID | STRING | Identification of the remote SFB inside the Channel |
| NDR | BOOL | If TRUE, new string received in RD |
| ERROR | BOOL | If TRUE, new non-zero STATUS received |
| STATUS | DINT | Last detected status |
| RD | STRING | Received string |

Description:

Receive a string from a remote or local resource (of current Project or another Project).

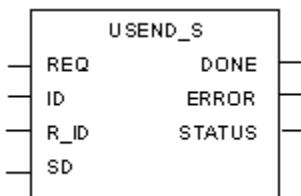
Warning: Connect block must have been called in current cycle before the URCV_S call. This CFB receives a string from one USEND_S instance. Previously received string is overwritten. If string is successfully received then NDR is set to TRUE during one cycle. If an error occurs, the ERROR output parameter is set to TRUE and the status is set in the STATUS parameter.

STATUS can have the following values:

| STATUS | Description |
|--------|--------------------------------|
| 0 | Receive successfully completed |
| 1 | Waiting for message |
| 2 | Invalid identifier |
| 3 | Not ready for receive |
| 6 | Waiting for message |
| 7 | Dialog has failed |

See example in the description of the CONNECT Block.

USEND_S



Arguments:

| | | |
|------|--------|---|
| REQ | BOOL | Send request on rising edge |
| ID | DINT | Identification of the communication channel |
| R_ID | STRING | Identification of the remote CFB inside the channel |

| | | |
|--------|--------|--|
| SD | STRING | String to send |
| DONE | BOOL | If TRUE, function performed successfully |
| ERROR | BOOL | If TRUE, new non-zero STATUS received |
| STATUS | DINT | Last detected status |

Description:

Send a string to a remote or local Resource (of current Project or another Project).

Warning: Connect block must have been called in current cycle before the USEND_S call. This CFB sends a string to one URCV_S instance on rising edge of REQ. If string is successfully sent then DONE is set. If an error occurs, the output parameter ERROR is set to TRUE and the status is set in the STATUS parameter.

STATUS can have the following values:

| STATUS | Description |
|--------|-----------------------------|
| 0 | Send successfully completed |
| 1 | Send in progress |
| 2 | Invalid identifier |
| 3 | Not ready to send |
| 6 | Dialog has failed |
| 7 | Send has failed |

If the send failed, a new send is not automatically done, a rising edge must be detected on REQ parameter.

See example in the description of the CONNECT block.

Glossary

The Glossary contains terms used in the Workbench and their definitions.

| | |
|---|--|
| Access Control | The use of password-protection to control access to projects, resources, POU's, and targets. For projects, resources, and POU's, access control can also limit access to read-only mode. |
| Access Method | Methods to access the Virtual Machine database from a client application (programmed in C): SMA, MIB, SID. |
| Action | In SFC: an action can be on a Boolean variable or a child SFC, or a collection of operations (written in ST, IL, LD) to perform with an associated SFC step. The action is executed when the Step (SFC) is active. In FC: an action is a collection of operations (written in ST, IL, LD) to perform. |
| Activity of a Step | Attribute of a Step (SFC) which is activated by an SFC token. |
| Address | Optional hexadecimal address freely defined for each variable. This address can be used by an external application to access the value of the variable when the resource is executed by the Target. |
| Alias | The property of a variable indicating a short name for a variable. For FBD and LD diagrams, aliases indicate the parameters in functions and function blocks. |
| Array | Set of elements of the same type referenced by one or more indexes enclosed in square brackets and separated by commas. The index is an integer. Examples: tabi[2] or tabij[2,4]. |
| Attribute | The property of a variable indicating whether a variable is read-only, write-only, or free (read and write). |
| Automatic Instance (of a function block) | A function block having no assigned instances. Automatic instances of function blocks cannot be added to a POU during online changes. See also Declared Instance (of a function block) |

| | |
|-------------------------------|--|
| Binding | Bindings are directional links, i.e., access paths, between variables located in different resources. The Workbench enables two types of bindings: internal bindings and external bindings. Internal bindings are between resources within the same project. External bindings are between resources belonging to different projects. |
| Binding Error Variable | Variables enabling the management of binding errors at the consumer resource level. |
| Boolean (Bool) | Basic type that can be used to define a variable, a Parameter (POU) or a device. A Boolean can be TRUE (1) or FALSE (0). |
| Boolean Action | SFC Action: a Boolean variable is assigned with the activity of a Step (SFC). |
| Breakpoint | <p>SFC POU: Mark placed by the user at debug time, on an SFC Step (SFC) or Transition. The Target system stops when an SFC token is moved on a breakpoint.</p> <p>Step-by-step mode: For ST and IL POUs, you set breakpoints to specific lines of code. For LD POUs, you set breakpoints to rungs. When running an application in Debug mode, the application stops when it encounters a breakpoint.</p> |
| C Function | Function written with the "C" language, called from POUs, in a synchronous manner. |
| C Language | High level literal language used to access particularities of the target system. C language can be used to program C functions, function blocks and conversion functions. |
| Call Stack | Information which tracks stepping between POUs and called functions. Debug information includes call stack. You can only generate debug information for resources producing TIC code. |
| Cell | Elementary area of the graphic matrix for graphic languages such as SFC, FBD or LD or for the Dictionary Grid View. |
| CFB | Indicates a C function block |
| CFU | Indicates a C function |

| | |
|------------------------------|---|
| Channel | A channel of a device represents a hardware I/O point. It can be an input or an output. A variable is generally connected to a channel in order to be used in POUs. Directly represented variables can also be used in POUs. |
| Check In | Sending the contents of Workbench elements including projects, configurations, resources, and POUs for storage in a version source control database. Checked-in elements can be recovered at a later time. |
| Child | For SFC and FC, program which is activated by its father. The child has only one father. Only its father can start or kill it. A father can have more than one child. |
| Clearing a Transition | The forcing of the clearing of a transition whether the latter is valid or not (i.e all previous steps are active or not). Tokens are moved and actions are executed as for a usual transition clearing. All tokens existing in the preceding steps are removed. A token is created in each of the following steps. |
| CMG | Short name for the configuration manager |
| Coil | Graphic component of an LD Program representing the assignment of an output or an internal variable. |
| Common Scope | Scope of a declaration applying to all POUs within a Project. (Only defined words and types can have common scope). |
| Complex Equipment | See I/O Complex Device. |
| Condition | A Boolean expression attached to an SFC Transition or an FC test. In case of an SFC transition, the transition cannot be cleared when its condition is false. |
| Configuration | A software object made up of one or more resources. A configuration becomes a RAS device target when it is downloaded onto a target. |
| Configuration Manager | (ConfigurationManager.exe) The executable file providing communication services between the Workbench and target. Responsible for launching, killing, and giving the status of running virtual machines. |
| Connection | The link between networks and configurations, displayed in the hardware architecture view. |

| | |
|-----------------------------------|--|
| Constant Expression | Literal expression used to describe a constant value. |
| Consumer Group | A group holding external producer variables having bindings with consumer variables defined in the project. |
| Consumption Error Behavior | Indication of the value to use when an error occurs for an internal binding. Possible values are either the last value issued from the binding or a specified default value. |
| Contact | Graphic component of an FBD or LD diagram. Depending on the type of contact, it represents the value or function of an input or an internal variable. |
| Contextual Menu | Menu that is displayed under the mouse cursor by right-clicking the mouse. |
| Convergence | Multiple connection link from multiple SFC symbols (steps or transitions) to a single symbol. Convergences can be single or double. A single convergence (OR) is a multiple link from multiple transitions to the same step. A double convergence (AND) is a multiple link from multiple steps to the same transition. |
| Conversion | Filter attached to an input or output variable. The conversion is automatically applied each time the input variable is read or the output variable is refreshed. |
| Conversion Function | "C" written Function which describes a conversion. Such a conversion can be attached to any input or output, integer or real variable. |
| CRC | Cyclic redundancy checking |
| Cross References Browser | A tool that finds all references to variables, i.e., cross references, defined in the POU's of a project. The browser provides a total view of the declared variables in the programs of the project and where these are used. |
| CSV File Format | (Comma Separated Values) A delimited data format having each piece of information separated by commas and each line ending with a carriage return. The CSV file format can be used for importing or exporting variables data. |
| Current Result (IL) | Result of an instruction in an IL POU. The current result can be modified by an instruction, or used to set a variable. |

| | |
|--|--|
| Cycle | The Virtual Machine executes the programs of a resource as a cycle. All programs of the resource are executed following the order defined by the user, from the first program to the last and again and again. Before the execution of the first program, inputs are read. After the execution of the last program, the outputs are refreshed. |
| Cycle Time | The time between two input scans on the target. It represents the time to execute one cycle. The cycle time can differ at each cycle if none is programmed. When the cycle time is shorter, the Virtual Machine waits until this time has elapsed. When the cycle time is longer, the Virtual Machine immediately scans the inputs but signals with the "overflow" that the programmed time has been exceeded. When the cycle time is 0, the Virtual Machine does not wait to start a new cycle. |
| Cycle-to-cycle Mode | Execution mode: In this mode, cycles are executed one by one, according to the orders given by the user of the debugger. |
| Database | The collection of definitions making up a Workbench project. The version source control feature stores checked-in information in a separate database. |
| Data Link | A directional link between resources across which variable bindings data is conveyed. |
| Debug Information | For use when debugging using the step-by-step mode with ST, IL, and LD POU (programs, functions, and function blocks). Debug information includes call stack information which tracks stepping between POU and called functions. You can only generate debug information for resources producing TIC code. |
| Declared Instance (of a function block) | A function block having assigned instances, i.e., declared in the dictionary. Declared instances of function blocks can be added to a POU during online changes. |
| | See also Automatic Instance (of a function block) |
| Defined Word | Word that is an expression. This word can be used in POU. At compiling time the word is replaced by the expression. A defined word can not use a defined word. |

| | |
|--------------------------------------|---|
| Delayed Operation (IL) | Operation of an IL Program, executed when the ")" instruction occurs, later in the Program. |
| Dependency (on a library) | The state where a project uses, i.e., depends, on functions or function blocks defined in a library. |
| Device | See I/O device. |
| Dictionary | The view displaying the variables, function and function block parameters, types, and defined words used in the programs of a Project. |
| Dimension | The size (number of elements) of an array. For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements. |
| Direction | Variables and devices have a direction. For the property of a variable, direction indicates whether a variable is an input, output, or internal. The direction of a device can be input or output. |
| Directly Represented Variable | A variable is generally declared before its use in one POU. Inputs and outputs can be used without any declaration respecting a defined syntax. It corresponds to direct represented variables. Example: %QX1.6, %ID8.2 |
| Divergence | Multiple connection link from a single SFC symbol (steps or transitions) to multiple SFC symbols. Divergences can be single or double. A single divergence (OR) is a multiple link from one step to many transitions. A double divergence (AND) is a multiple link from one transition to many steps. |
| Double Integer (DINT) | Signed double integer 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| Driver | See IO driver, Network Driver. |
| Edge | See Falling Edge, Rising Edge. |
| ETCP | (ETCP.exe) DS800 Development Suite network driver that uses the TCP / IP stack. |
| Events Logger | A logger that receives events from DS800 Development Suite targets. You view these events using the Events Viewer. Events are stored in a log file, in Unicode format. A new log file is automatically created each day at 00:00:00 hours |

| | |
|------------------------------|---|
| Events Viewer | A viewer that displays run-time system events logged with the Events Logger. |
| Execution Mode | The mode in which a resource is executed: real-time, cycle-to-cycle, and step-by-step. |
| External Binding List | The list of consumer groups, holding external producer variables having bindings with consumer variables defined in the project, and producer groups, holding outgoing producer variables for consumption in external bindings defined in another project. |
| Expression | Set of operators and identifiers. |
| Falling Edge | A falling edge of a Boolean variable corresponds to a change from TRUE (1) to FALSE (0). |
| Father Program | For SFC and FC, program which controls other programs, called its children. See Child. |
| FBD | Function Block Diagram. Programming language. |
| FC | Flow Chart. Programming language. |
| Function | POU which has input parameters and one output parameter. A function can be called by a program, a function or a function block. A function has no instance. It means that local data are not stored, and are generally lost from one call to the other. A function can be written in ST, IL, LD, FBD and "C". |
| Function Block | POU which has input and output parameters and works on internal data (parameters). A program can call an instance of a function block. A function block instance can not be called by a function (no internal data for a function). A function block can call another function block (instantiation mechanism is extended to the function blocks called). A function block can be written in ST, IL, LD, FBD and "C". |
| Global Scope | Scope of a declaration applying to all POUs of one resource. |
| Global Variable | A variable whose scope is global. |
| Hardware Architecture | The view graphically displaying the configurations of a project and the network links between them. |

| | |
|---------------------------------------|---|
| Hidden Parameter | Input parameters of a function block that are not displayed in FBD diagrams. Hidden parameters are set in the Parameters tab of the Select Block dialog. |
| Hierarchy | Architecture of a Project, divided into several POUs. The hierarchy tree represents the links between father programs and children programs. See Father Program, Parent Program. |
| Identifier | Unique word used to represent a variable or a constant expression in the programming. |
| IFB | Indicates an IEC 61131 function block |
| IFU | Indicates an IEC 61131 function |
| IL | Instruction List. Programming language. |
| Initial Situation | Set of the initial steps of an SFC Program, which represents the context of the program when it is started. |
| Initial Step | Special Step (SFC) of an SFC Program, which is activated when the program starts. |
| Initial Value | Value which has a variable when the Virtual Machine starts the execution of the resource. The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the Virtual Machine has stopped. |
| Input | Direction of a variable or a Device. An input variable is connected to an input channel of an input Device. |
| Input Parameter | Input argument of a function or a function block. These parameters can only be read by function or function block. A parameter is characterized by a type. |
| Instance (of a Function Block) | Copy of the internal data of a function block which persists from one call to the other. This word is used, by extension, to say that a program calls a function block instance and not the function block itself. |
| Instruction | Elementary operation of an IL program, entered on one line of text. |
| Internal | Attribute of a variable, which is not linked to an input or output device. Such a variable is called an internal variable. |

| | |
|------------------------------|--|
| Internal Binding List | The view displaying the resource links and internal variable bindings defined for a project. |
| I/O Binding | A virtual connection between two software elements. |
| I/O Channel | See Channel. |
| I/O Complex Device | Element grouping several "simple devices". This provides the means for manufacturers to mix types and directions. The implementation of the I/O Driver of a complex device corresponds to the implementation of the drivers of all the devices composing it. Parameters are also attached to a complex device, OEM parameters. |
| I/O Simple Device | Element grouping several channels of the same type and same direction (INPUT, OUTPUT). An Array can be connected to a device if all elements are connected to contiguous channels, the type of the array must be the type of the Device. Variables of the same type can also be connected to channels of a device. A device corresponds to a hardware device and an I/O Driver in (or linked to) the Virtual Machine. Parameters are also attached to a device: the OEM parameters. I/O devices are defined by the integrator. |
| I/O Driver | "C" code which makes the interface between a Virtual Machine and the hardware devices. The driver can be statically linked to the Virtual Machine or in a separate DLL (such as for the Windows NT target). Two types of drivers are available for use in the Workbench: generic and advanced. |
| IO Variable | Variable connected to an input or output device. An IO variable must be connected on a channel of an IO device. |
| IO Wiring | Definition of the links between the variables of the Project and the channels of the devices existing on the Target system. |
| ISaRSI | (IsaRSI.exe) Enhanced serial port driver. The network driver that provides communication with the workbench on a serial port. Similar to ETCP. |
| ITA | Indicates an array |
| ITS | Indicates a structure |

| | |
|---------------------------|---|
| IXLSma Server | (IxLSmaServer.exe) Provides service for performing IXL read operations, using the HSD driver with the SMA method. This method is independent from the virtual machine cycle and is thus faster. |
| Jump to a Step | SFC graphic component representing a link from a Transition to a Step (SFC). The graphic symbol of a jump is an arrow, identified with the reference of the destination step. |
| Keyword | Reserved identifier of the language. |
| Label | For FBD, IL, or LD, identifier identifying an instruction. Labels can also be used for jump operations. |
| LD | Ladder Diagram. Programming language. |
| Level 1 of the FC | Main description of an FC program. Level 1 groups the chart (actions and tests), and the attached comments. |
| Level 1 of the SFC | Main description of an SFC program. Level 1 groups the chart (steps and transitions), and the attached comments. |
| Level 2 of the FC | Detailed description of an FC program. It is the description of the actions and tests. Level 2 programming for FC elements can be developed with ST or LD. |
| Level 2 of the SFC | Detailed description of an SFC program. It is the description of the actions within the steps, and the Boolean conditions attached to the transitions. Level 2 programming for SFC elements can be developed with ST or LD or call an SFC child. |
| Library | Special projects made up of configurations and resources in which you define functions and function blocks for reuse throughout DS800 Development Suite projects. Libraries also enable you to modularize projects and to isolate functions and function blocks so that these can be validated separately. |
| Link | For FBD, SFC, or LD diagrams, a graphic component connecting elements in a diagram. |
| Link Architecture | The view graphically displaying the resources of a project and the resource data links, used for internal bindings, between them. This is the default view of the Workbench providing a main entry point to all editors. |
| Literal | A lexical unit that directly represents a value. |

| | |
|--------------------------|---|
| Local scope | Scope of a declaration applying to only one POU. |
| Locked I/O | Input or output variable, disconnected logically from the corresponding I/O device, by a "Lock" command sent by the user from the debugger. |
| Maximum time | Time of the longest cycle since the Virtual Machine has started the execution of the programs of a resource. |
| Memory for Retain | Run-time setting for a resource indicating the location where retained values are stored (the required syntax depends on the implementation). |
| Message | See STRING. |
| Method | See Access Method. |
| Modifier (IL) | Single character put at the end of an IL operation keyword, which modifies the meaning of the operation. |
| Network | The means of communication between configurations and their clients. |
| Network Driver | "C" code which makes the interface between the Target network layer and the network. |
| Non-stored Action | SFC Action: it is a list of statements, executed at each Target cycle, when the corresponding Step (SFC) is active. |
| OEM | Original Equipment Manufacturer |
| OEM Parameter | Parameters attached to an IO device or an I/O Complex Device. A parameter is characterized by a type. An OEM parameter is defined by the designer of the Device. It can be a constant, or a variable parameter entered by the user during the I/O connection. |
| Operand (IL) | Variable or constant expression processed by an elementary IL instruction. |
| Operation (IL) | Basic instruction of the IL language. An operation (or operator) is generally associated to an operand in an instruction. |
| Operator | Basic logical operation such as arithmetic, boolean, comparator, and data conversion. |
| Output | Direction of a variable or a device. An output variable is connected to an output channel of an output Device. |

| | |
|-------------------------|--|
| Output Parameter | Output argument of a function or function block. These parameters can only be written by a function or function block. A function has only one output parameter. A parameter is characterized by a type. |
| Overflow | Integer value which corresponds to the number of times the cycle time has been exceeded. Always 0, if cycle time is 0. |
| Parameter (POU) | See Input Parameter, Output Parameter, OEM Parameter, and Hidden Parameter |
| Parent Program | It can be a Father Program or an FC program that call an FC Sub-program. |
| PLC | Programmable Logic Controller |
| POU | Program Organization Unit: set of instructions written in one of the following languages: SFC, FC, IL, ST, FBD, LD. A POU can be a program, a function or function block. |
| Power Rail | Main left and right vertical rails at the extremities of a ladder diagram. |
| Producer Group | A group holding outgoing producer variables for consumption in external bindings defined in another project. |
| Program | See POU. A program belongs to a resource. It is executed by the Virtual Machine, depending on its location (order) in the resource. |
| Project | Set of configurations and links between their resources. |
| Project Updater | A program allowing to convert projects developed using previous versions for use within the latest version. Each time you upgrade to a newer version, you need to update projects. |
| PROPI | PROPI is an interface enabling you to send commands directly to the Workbench via a custom application. For instance, you could use the PROPI interface when using the Workbench in the background. |
| Pulse Action | SFC Action: it is a list of statements executed only once when the corresponding Step (SFC) is activated. |
| Qualifier | Determines the way the action of a step is executed. The qualifier can be N, S, R, P0 or P1. |

| | |
|-----------------------------|--|
| Real | Type of a variable, stored in a floating IEEE single precision 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| Real Device | I/O Device physically connected to an I/O device on the target machine. See Virtual Device. |
| Real Time Mode | Run time normal execution mode: the Target cycles are triggered by the programmed cycle timing. |
| Reference Name (SFC) | Name which identifies an SFC Step (SFC) or Transition in an SFC program. |
| Register (IL) | Current result of an IL sequence. |
| Resource | The POU's and definitions making up a Virtual Machine. |
| Resource Name | The unique identifier of a resource within a configuration. |
| Retain | Attribute of a variable. The value of a retain variable is saved by the Virtual Machine at each cycle. The value stored is restored if the Virtual Machine stops and restarts. |
| Return | Graphic component of an LD program representing the conditional end of a program. |
| Return Parameter | See Output Parameter. |
| Rising Edge | A rising edge of a Boolean variable corresponds to a change from FALSE (0) to TRUE (1). |
| Rung | Graphic component of an LD program representing a group of circuit elements leading to the activation of a coil in an LD diagram. |
| Run-time Error | Application error detected by the Target system at run time. |
| Scope | See Global Scope, Common Scope, Local scope. |
| Section | Program, function and function block sections are where are localized POU of a resource. POU's located in the Program section are executed by the Virtual Machine. |
| Security State | The indication of the level of access control that is applied to a resource, a POU, or a target. |

| | |
|---------------------------------|---|
| Selection List | Also known as a 'combo-box'. When a Selection List is provided for a particular cell, clicking on its right part (down arrow), displays the available choices. To make a selection, perform one of the following operations: - click on the item (use the scroll bar first if the required choice is not visible) - move in the list using the cursor keys and press Enter - type the first letter (if more than one item starts with this letter, press the letter again to select the next occurrence). |
| Separator | Special character (or group of characters) used to separate the identifiers in a literal language. |
| Sequential | Attribute of a program. A sequential program gives an order to operations of a process and conditions between operations. Generally, it is programmed with SFC or FC. |
| Server | Part of the target that receives requests from IXL to retrieve information about the resource run by the Virtual Machine. |
| SFB | Indicates a standard function block |
| SFC | Sequential Function Chart. Programming language. |
| SFU | Indicates a standard function |
| Short Integer (SINT) | Signed integer 8-bit format. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| Single Resource Mode | The project editing mode limiting access for an individual user to one resource and its POUs. Other users can access other resources of the same project. |
| SIT | Indicates a Standard IEC 61131 type. |
| ST | Structured Text. Programming language. |
| Standard IEC 61131 Types | Boolean (Bool), Short Integer (SINT), Double Integer (DINT), Real, Timer (TIME), STRING. See Type. |
| Statement | Basic ST complete operation. |

| | |
|---------------------------------------|--|
| Step (SFC) | Basic graphic component of the SFC language. A step represents a steady situation of the process, and is drawn as a square. A step is referenced by a name. The activity of a step is used to control the execution of the corresponding actions. |
| Step (FC) | The word step may be used for Flow Chart actions. |
| See Action. | |
| Step-by-step Mode | A mode used while debugging ST, IL, and LD POU's where you set breakpoints at specific lines of code or rungs causing the application to stop when reached. |
| STRING | Character string. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| Structure | Corresponds to a type which has previously been specified to be a data structure, i.e. a type consisting of a collection of named elements (or fields). Each field can be a basic type, a basic structured type, a structure or an array. A field of a variable with a structure type can be accessed using the following syntax: VarName.a, VarName.b[3], VarName.c.d |
| Sub-program | Programs written in SFC or FC language and called by a father program. A sub-program is also called a child program. To call sub-programs written in another language, use a function. A function can be called by any POU. |
| Symbol Table | The file corresponding to the variables and function blocks defined for a resource. This file is downloaded onto the target. The symbol table is set to one of two formats: complete table or reduced table. The complete table contains all defined variables, whereas, the reduced symbol table only contains the names of variables having a defined Address cell. |
| Symbols Monitoring Information | When debugging or simulating, code required to enable graphically displaying the output values of functions and operators in FBD and LD diagrams. |
| System Events | Events occurring on the development platform. Such events can be logged using the Events Logger and viewed using the Events Viewer. |

| | |
|----------------------------------|--|
| System Variable | System variables hold the current values of all system variables for a resource. You can read from or write to system variables. These variables are defined in the dsys0def.h file. For example, the current cycle time is a system variable that can only be read by a program. |
| Target | The hardware platform on which Virtual Machines run resources of a project. You download configurations (Configs), onto a RAS device target. |
| Target Definition Builder | The Target Definition Builder enables the description of targets (main definition and options of the embedded software), complex data types (such as defined in IEC languages), "C" functions, function blocks and conversion functions, I/O devices or network drivers for IXL communication and/or data binding. |
| TIC Code | Target Independent Code produced by the DS800 Development Suite compiler for execution on virtual machines. |
| Timer (TIME) | Unit of a timer is the millisecond. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| Token (SFC) | Graphical marker used to show the active steps of an SFC program. |
| Top Level Program | Program put at the top of the hierarchy tree. A top level program is activated by the system. See also Parent Program, Father Program. |
| Transition | Basic graphic SFC component. A transition represents the condition between different SFC steps. A transition is referenced by a name. A Boolean Condition is attached to each transition. |
| Type | <p>Data types are defined for many items in DS800 Development Suite projects:</p> <ul style="list-style-type: none"> - variables -function or function block parameters - devices <p>See Standard IEC 61131 Types, Basic Structured Types, User Types.</p> |

| | |
|---------------------------------|--|
| User Data | User Data are any data of any format (file, list of values) which have to be merged with the generated code of the resource in order to download them into the RAS device target. Such data are not directly operated by the Virtual Machine and is commonly dedicated to other software installed on the target PLC. |
| User Types | Types that the user can define using basic types or other user types. User types can be arrays or structures. |
| Validity of a Transition | Attribute of a Transition. A transition is validated (or enabled) when all the preceding steps are active. |
| Variable | Unique identifier of elementary data which is used in the programs of a Project. |
| Variable Binding | See Binding |
| Variable Group | Grouping of variables enabling managing and logically sorting these within a resource. Variable groups are displayed in the dictionary's variables tree. |
| Variable Name | A unique identifier, defined in the Workbench, for a storage location containing information used in exchanges between resources. |
| Version Information | The information indicating the compilation version number, the compilation date, and the CRC of the data the resource works on for three sources of resource code: <ul style="list-style-type: none"> - the compiled code for the resource in the Workbench project - the code for the resource running on the target - the code for the resource stored on the target |
| Version Source Control | A tool that manages the changing versions of Workbench elements including projects, configurations, resources, and POU's by saving them to a version source control database. Saving these elements to a control database enables you to retrieve older versions of the elements at a later time. |
| Virtual Device | I/O Device which is not physically connected to an I/O device of the Target machine. See Real Device. |
| Virtual Machine | (IsaVM.exe) The instantiation of a resource on a Target. |

| | |
|-------------------|--|
| Wiring | The property of a variable indicating the I/O channel to which the variable is wired. |
| Zip Source | An exchange file (.PXF) holding all data from Workbench elements. From the compilation options for a resource, you can choose to embed a zip source file for resources, configurations, or projects onto the target. This source file can be uploaded from the target at a later time. |

Index

Symbols

-) operator for IL 482
- * operator 488
- + operator 489
- operator 491
- / operator 492
- < operator 511
- <= operator 510
- <> operator 515
- = operator 505
- > operator 508
- >= operator 507
- __SYSVA_KVBCERR, consumption error
 - variables 68
- __SYSVA_KVPERR, production error
 - variables 68

Numerics

- 1 gain operator 494

A

- ABS function 522
- access control
 - for configurations 118
 - for POU's 104
 - for projects 41
 - for resources 64
- accessing
 - configuration properties 114
 - contextual menus 25
 - diagnostic information 307
 - events viewer, run-time system events 176
 - history details, previous versions of
 - Workbench elements 366
 - internal binding list, the 71
 - resource properties 54
 - the cross references browser 355
 - the Dictionary view 125
 - the external binding list 81
- ACOS function 523
- action blocks
 - adding in SFC charts 229
 - attaching to SFC steps 229
 - calling functions and function blocks from 409

deleting in SFC charts 233
moving in execution order 232

actions, Flow Chart
described 420
inserting 240

actions within steps
boolean 404
described 404
list of instructions for 408
non-stored 406
pulse 405
SFC 407

adding
action blocks in SFC charts 229
descriptions for configurations 119
descriptions for POU's 108
descriptions for resources 66
FC sub-programs 102
I/O devices for I/O wiring 169
POU's in resources 99
rows to the Dictionary grid 134
SFC child programs 102
variables to spy list 316

addition operator 489

addresses, renumbering in the Dictionary grid 140

adjusting zoom, workspace 23

alarm and event operations
ALARM function block 575
EVENT function block 588

ALARM function block 575

aligning coils on rungs (LD elements) 264

AND operator 495

AND_MASK function 524

ANY_TO_BOOL operator 496

ANY_TO_DINT operator 499

ANY_TO_REAL operator 501

ANY_TO_SINT operator 498

ANY_TO_STRING operator 504

ANY_TO_TIME operator 502

appearance
of I/O wiring view 156
of language editors 182

of simulator 322
of the Dictionary view 126

arithmetic operations
1gain operator 494
ABS function 522
ACOS function 523
addition operator 489
ASIN function 526
ATAN function 527
COS function 530
division operator 492
EXPT function 533
LOG function 540
MOD function 546
multiplication operator 488
NEG operator 512
POW function 554
RAND function 555
SIN function 565
SQRT function 566
subtraction operator 491
TAN function 569
TRUNC function 570

arrays
basic or user types, described 379
initializing elements in 150

ascending order, sorting for Dictionary grid 138

ASCII function 525

ASIN function 526

assignment, ST basic statement 459

ATAN function 527

attaching action blocks, SFC steps 229

attributes, variables 389

auto input of names, variables or blocks 265

automatic instances of function blocks, debugging
334

available programming languages
for function blocks 99
for functions 98
for programs 96

AVERAGE function block 575

B

background colors, customizing for views and editors 26

begin, Flow Chart component 418

binary operations

- NOT_MASK function 550
- OR_MASK function 553
- ROL function 559
- ROR function 560
- SHL function 563
- SHR function 564
- XOR_MASK function 571

bindings

- between variables, described 67
- error variables for 68
- external, between projects 81
- external, defining 88
- internal, defining 78
- internal, within a project 71

BLINK function block 577

blocks (functions and function blocks) in LD

- on the left, inserting 262
- on the right, inserting 262
- usage of 451

boolean

- actions within steps 404
- constant expressions 381
- negations in FBD, described 433
- variables 390

boolean operations

- AND operator 495
- AND_MASK function 524
- F_TRIG function block 588
- NOT operator 514
- ODD function 551
- OR operator 516
- R_TRIG function block 591
- RS function block 593
- SR function block 598
- XOR operator 518

BREAK resource state 291

breakpoints

- on step activation (SFC) 312
- on step deactivation (SFC) 313
- on transition (SFC) 314
- removing, step-by-step mode 301
- setting, step-by-step mode 301
- setting/removing for steps and transitions (SFC) 311
- viewing (step-by-step mode) 307

browser

- for cross references 355
- manipulating in simulator 326

browsing POU s of a project 357

building code

- for POU s 346
- for projects 345
- for resources/projects 347

builds, stopping (projects, resources, and POU s) 348

C

C source code, implications of generating 351

CAL operator for IL 485

calculating cross references 357

calling

- function blocks from IL (CAL operator) 485
- function blocks from transitions 413
- function blocks in FBD 433
- functions from IL 483
- functions from transitions 412
- functions in FBD 433

CASE, OF, ELSE, END_CASE, ST basic statements 462

cell-level validation 153

changing, coils and contacts types (LD elements) 263

channels in I/O devices
 freeing 174
 mapping 172
 wiring 172

CHAR function 528

checking in Workbench elements, version source
 control 363

child SFC POU, described 370

cleaning
 code stored on targets 336
 projects and resources 348

clearing
 the contents of output window 24
 transitions (SFC) 311
 transitions, forcing of 315

clearing VSC status 359

closing projects 36

CMP function block 578

code
 building/rebuilding for projects 345
 cleaning from targets 336
 downloading to targets for resources 293
 generating for resources 55
 sequences of, particular cases for online
 changes 327
 stopping builds of 348

coils (LD elements)
 aligning on rungs 264
 changing types of 263
 inserting in FBD POU 271
 inserting in Ladder diagrams 262

collapsing grid components, Dictionary grid 135

command lines
 opening projects 36
 starting events logger 175

comments
 displaying or hiding for variables 279
 in FC charts, described 424
 inserting in FBD 272
 inserting in FC charts 245
 inserting in literal languages 392

communications
 CONNECT function block 579
 URCV_S function block 612
 USEND_S function block 613

comparison operations
 CMP function block 578
 equal operator 505
 greater than operator 508
 greater than or equal operator 507
 less than operator 511
 less than or equal operator 510
 not equal operator 515

compilation
 options for resources 55
 stopping in progress 348

compiling
 POUs 346
 projects 345
 resources/projects 347

complex structures (examples of), Flow Chart 425

computer allocated hidden variables, effect on
 online changes 330

conditions
 attached to transitions 410
 for downloading resource code 293
 in FC charts, described 420

configurations
 accessing details for previous versions of
 366
 accessing the properties window for 114
 checking in 363
 comparing versions of 365
 controlling development access for 118
 controlling target access for 118
 creating history reports for 366
 creating in project 110
 defining target properties for 117
 deleting from a project 111
 editing descriptions of 119
 general properties for 115
 getting previous versions of 365
 identification of 115

inserting resources in 112
managing 110
moving in hardware architecture view 112
moving resources between 113
viewing the history of 364

CONNECT function block 579

connection lines in Ladder diagrams 436

connections (configurations to networks)
 creating 123
 deleting 124
 described 123

connectors (Flow Chart)
 described 424
 linking elements 244

constant expressions
 boolean 381
 double integer 382
 real 382
 short integer 381
 string 383
 timer 383

consuming variables, viewing for internal
 bindings 77

consumption error variables 68

contacts (LD elements)
 changing types of 263
 inserting in FBD POU 270
 on the left, inserting 261
 on the right, inserting 261

contextual menus, accessing 25

convergences (SFC elements)
 deleting branches from 220
 double, described 402
 inserting new branches in 219
 linking and placing in chart 217
 single, described 400

conversions, deleting from I/O wiring 171

copying
 POUs 100
 resources 51
 variables (Dictionary grid elements) 136

corners, inserting (FBD elements) 267

COS function 530

counters
 CTD function block 581
 CTU function block 582
 CTUD function block 583
 DBG_CLR_GET_ERR function block 585
 DBG_CLR_SET_ERR function block 585
 DBG_GET_ERR function block 586
 DBG_SET_ERR function block 586

cover page, adding as printing option 340

CRC (Cyclic Redundancy Checking), viewing for
 resources 307

creating
 configurations 110
 connections between configurations and
 networks 123
 data links 74
 FC sub-programs 102
 history reports, version source control 366
 libraries 281
 networks 120
 POUs in resources 99
 projects 34
 resources 50
 SFC child programs 102
 structures in the types tree 129
 variable groups 91

cross references
 browser for 355
 browsing POU of a project 357
 calculating 357
 defining search options for finding 358
 csv files, importing variables data using 93

CTD function block 581

CTU function block 582

CTUD function block 583

current step, locating for step-by-step mode 302

CURRENT_ISA_DATE function 531

cursor coordinates, displaying in FBD and LD
 editors 254

custom parameters, resources 63

customizing, colors/fonts of views and editors 26

cutting
 POUs 100
 variables (Dictionary grid elements) 136
cycle time
 setting for resources, debug mode 303
 setting for resources, edition mode 59
cycle-to-cycle execution mode, resources 299
cyclic and sequential operations 369

D

data conversion
 ANY_TO_BOOL operator 496
 ANY_TO_DINT operator 499
 ANY_TO_REAL operator 501
 ANY_TO_SINT operator 498
 ANY_TO_STRING operator 504
 ANY_TO_TIME operator 502
 TMR operator 517
data links (internal bindings)
 creating 74
 deleting 76
 hiding and showing 77
data manipulation
 AVERAGE function block 575
 LIMIT function 539
 MAX function 541
 MIN function 543
 MUX4 function 547
 MUX8 function 549
 SEL function 562
database commands
 DBG_CLR_GET_ERR 585
 DBG_CLR_SET_ERR 585
 DBG_GET_ERR 586
 DBG_SET_ERR 586
 SOFT_POINT_READ 596
 SOFT_POINT_WRITE 597
 TLP_GET_DINT 601
 TLP_GET_REAL 602
 TLP_GET_SINT 603

TLP_GET_STRING 604
TLP_GET_TLP 605
TLP_SET_DINT 606
TLP_SET_REAL 607
TLP_SET_SINT 608
 TLP_SET_STRING 609
database-level validation 154
DBG_CLR_GET_ERR function block 585
DBG_CLR_SET_ERR function block 585
DBG_GET_ERR function block 586
DBG_SET_ERR function block 586
debug
 information, generating at program level 107
 information, generating at resource level 55
 mode, starting for the project 295
 toolbar in language editors 187
 toolbar in main environment 17
debugging
 instances of function blocks 334
 modes for a project 289
declared
 instances of function blocks, debugging 334
 variables, modifying in online changes 329
defined words
 described 392
 grid for, Dictionary 145
 parameters component for 90
defining
 external bindings 88
 internal bindings 78
 printing options for project items 340
 producer groups, external bindings 83
 search options for finding cross references 358
 TLP variables 146
delayed operations
 described (IL elements) 475
DELETE function 532
deleting
 action blocks from SFC charts 233
 configurations from a project 111

connections between configurations and networks 124
data links, internal bindings 76
external bindings 89
I/O devices and conversions from I/O wiring 171
internal bindings 80
POUs 100
producer groups, external bindings 85
resources 53
structures 130
variables (Dictionary grid elements) 136
demoting SFC child programs 103
dependencies, projects on libraries 282
DERIVATE function block 587
descending order, sorting for Dictionary grid 138
description languages, programs 375
descriptions
 adding to projects 40
 adding to resources 66
 editing for configurations 119
diagnostic information, accessing 307
diagram format, FBD 429
Dictionary
 accessing the 125
 adding and inserting rows 134
 appearance of the 126
 cutting, copying, and deleting elements (variables) in the 136
 defined words grid, described 145
 described 125
 duplicating rows in the 139
 editing contents of cells and rows in the 133
 expanding/collapsing grid components in the 135
 finding and replacing elements in the 137
 moving rows in the 135
 parameters grid, described 143
 parameters tree, described 128
 pasting elements (variables) in the 138
 printing the grid of the 141
 renumbering addresses in the 140
resizing columns and rows in the 132
selecting rows and elements in the 132
sorting the grid of the 138
types grid, described 144
types tree, described 129
variables grid, described 142
variables tree, described 127
working with the grids of the 131
direct
 coils in Ladder diagrams, described 443
 contacts in Ladder diagrams, described 440
direction, variables 389
directly represented variables 387
directory structure, installation 29
displaying
 errors and information, output window 24
 I/O device window headers 325
 the status bar, the 25
 tooltips for function blocks 267
 tooltips for variables 266
 variable comments 279
divergences (SFC elements)
 deleting branches from 220
 double, described 402
 inserting new branches in 219
 linking and placing 217
 single, described 400
division operator 492
DO-WHILE structures, inserting (FC elements) 242
docking toolbars 14
double
 convergences, described 402
 divergences, described 402
 integer constant expressions 382
 integer variables 390
downloading resources code onto targets 293
duplicating rows, Dictionary grid 139
dynamic behavior
 for Flow Chart diagrams 426
 setting SFC limits 59
 SFC charts, described 414

E

editing

- descriptions for configurations 119
- descriptions for POU's 108
- descriptions for resources 66
- external bindings 89
- internal bindings 80
- level 2 programming, SFC elements 228
- links between resources, external bindings 87
- producer groups, external bindings 85
- resource properties 54
- the contents of cells and rows, Dictionary grid 133
- transition code, SFC elements 231

editing modes

- for projects, normal and single-resource 32
- for the Dictionary, grid and line 131

elements

- moving in FBD POU's 275
- resizing in FBD POU's 274
- Workbench, uploading from targets 46
- end, Flow Chart component 418
- equal operator 505
- error detection 59
- ERROR resource state 291
- ETCP network parameter 62
- EVENT function block 588
- events logger, starting 175
- execution
 - order, moving action blocks in (SFC POU's) 232
 - order, showing for FBD programs 265
 - rules for resource cycles 376
 - starting and stopping for resources 297
- execution modes, resources
 - cycle-to-cycle 299
 - real-time 298
 - step-by-step 299
- EXIT, ST basic statement 466

expanding grid components, Dictionary 135

exporting

- variables data 93
 - Workbench elements between projects 43
- expressions in ST programs 455
- EXPT function 533
- extended properties, resources 63
- extensions, ST 467
- external bindings
 - accessing the list of 81
 - defining 88
 - defining producer groups for 83
 - deleting 89
 - deleting producer groups for 85
 - editing 89
 - editing links between resources for 87
 - editing producer groups for 85
 - linking resources for 86
 - overview of 81

F

F_TRIGGER function block 588

falling edge detection

- contacts, described 442
 - negative coils, described 448
- FBD (Function Block Diagram)
- boolean negation, described 433
 - displaying cursor coordinates 254
 - inserting comments 272
 - inserting corners 267
 - inserting function blocks 267
 - inserting jumps to labels 268
 - inserting labels 268
 - inserting links, connection 267
 - inserting returns 269
 - inserting variables 266
 - jumps and labels, described 431
 - main diagram format 429
 - managing guideline areas 254
 - monitoring output values 295

return statement, described 431
showing execution order 265
toolbar, language editor 195

FC (Flow Chart)
actions, described 420
begin component, described 418
comments, described 424
complex structures, examples of 425
conditions, described 420
connectors, described 424
creating sub-programs 102
dynamic behavior of 426
end component, described 418
execution of sub-programs 371
flow link, described 419
I/O specific actions, described 423
inserting actions 240
inserting comments 245
inserting connector links 244
inserting DO-WHILE structures 242
inserting flow links 243
inserting I/O specific actions 244
inserting IF-THEN-ELSE structures 241
inserting sub-programs 245
inserting tests 240
inserting WHILE-DO structures 242
language editor, menu bar options for 236
programs, hierarchy restrictions for 96
renumbering in charts 249
sub-programs, vertical structures of 422
syntax verification rules, main 426
toolbar, language editor 191
using goto symbols 249
viewing level 2 windows for 250
working with charts 239

filenames, projects 36

FIND function 534

finding
matching coils (LD POU)s 278
matching names (LD POU)s 278
variables (elements) in POUs 207
variables (elements) in the Dictionary 137

flow links
described 419
inserting 243

fonts
changing for printing options 340
customizing for views and editors 26

FOR, TO, BY, DO, END_FOR, ST basic
statement 465

forcing
transition clearing 315
values of variables in a spy list 319
values of variables in the Dictionary 304

foreground colors, customizing for views and
editors 26

freeing channels, I/O wiring 174

function blocks
ALARM 575
AVERAGE 575
BLINK 577
calling from action blocks 409
calling from IL (CAL operator) 485
calling from ST programs 457
calling from transitions 413
CMP 578
CONNECT 579
creating in resources 99
CTD 581
CTU 582
CTUD 583
DBG_CLR_GET_ERR 585
DBG_CLR_SET_ERR 585
DBG_GET_ERR 586
DBG_SET_ERR 586
debugging instances of 334
defining access control for 104
DERIVATE 587
described 373
displaying tooltips for 267
EVENT 588
F_TRIG 588
HYSTER 589
inserting in FBD diagrams 267

| | |
|---|-----|
| inserting in POU | 203 |
| INTEGRAL | 590 |
| manipulating in resources | 100 |
| modifying instances of, in online changes | |
| 330 | |
| R_TRIG | 591 |
| REQUEST_LICENSE | 592 |
| reusing through libraries | 281 |
| RS | 593 |
| SET_PRIORITY | 594 |
| SIG_GEN | 595 |
| SOFT_POINT_READ | 596 |
| SOFT_POINT_WRITE | 597 |
| SR | 598 |
| STACKINT | 600 |
| summary of standard | 573 |
| TLP_GET_DINT | 601 |
| TLP_GET_REAL | 602 |
| TLP_GET_SINT | 603 |
| TLP_GET_STRING | 604 |
| TLP_GET_TLP | 605 |
| TLP_SET_DINT | 606 |
| TLP_SET_REAL | 607 |
| TLP_SET_SINT | 608 |
| TLP_SET_STRING | 609 |
| TOF | 609 |
| TON | 610 |
| TP | 611 |
| URCV_S | 612 |
| USEND_S | 613 |
| working with | 99 |
| functions | |
| ABS | 522 |
| ACOS | 523 |
| AND_MASK | 524 |
| ASCII | 525 |
| ASIN | 526 |
| ATAN | 527 |
| calling from action blocks | 409 |
| calling from IL | 483 |
| calling from ST programs | 456 |
| calling from transitions | 412 |
| CHAR | 528 |
| COS | 530 |
| creating in resources | 99 |
| CURRENT_ISA_DATE | 531 |
| defining access control for | 104 |
| DELETE | 532 |
| described | 371 |
| EXPT | 533 |
| FIND | 534 |
| INSERT | 536 |
| inserting in POU | 203 |
| LEFT | 537 |
| LIMIT | 539 |
| LOG | 540 |
| manipulating in resources | 100 |
| MAX | 541 |
| MID | 542 |
| MIN | 543 |
| MLEN | 544 |
| MOD | 546 |
| MUX4 | 547 |
| MUX8 | 549 |
| NOT_MASK | 550 |
| ODD | 551 |
| OR_MASK | 553 |
| POW | 554 |
| RAND | 555 |
| REPLACE | 556 |
| reusing through libraries | 281 |
| RIGHT | 558 |
| ROL | 559 |
| ROR | 560 |
| SEL | 562 |
| SHL | 563 |
| SHR | 564 |
| SIN | 565 |
| SQRT | 566 |
| standard, summary of | 521 |
| SUB_DATE_DATE | 567 |
| TAN | 569 |

TRUNC 570
working with 98
XOR_MASK 571

G

general properties
 for configurations 115
 for resources 55
generating
 C source code, implications of 351
 debug information, program level 107
 debug information, resource level 55
 symbols monitoring information 107
 TIC code 55
getting previous versions of Workbench elements 365
GFREEZE statement 470
GKILL statement 469
go to line, ST and IL POU's 279
goto
 steps or transitions, SFC elements 227
 symbols, FC elements 249
greater than operator 508
greater than or equal operator 507
grid
 displaying for language editors 197
 editing mode for the Dictionary 131
 view for I/O Wiring 159
groups
 creating for variables 91
 managing for variables 91
 of variables, opening 91
GRST statement 471
GSTART statement 468
GSTATUS statement 472
guideline areas
 managing in FBD editor 254

H

hardware architecture view 109
headers/footers, including as printing option 340
hidden variables, computer allocated, effect on
 online changes 330
hiding
 resource links, internal bindings 77
 the status bar, Workbench 25
 toolbars, Workbench 14
 variable comments in language editors 279
hierarchy
 changing for SFC child programs 103
 restrictions for SFC and FC programs 96
history reports, creating 366
HSD network parameter 62
HYSTER function block 589

I

I/O devices
 adding for I/O wiring 169
 deleting conversions 171
 deleting from I/O wiring 171
 freeing channels of 174
 mapping channels of 172
 modifying with online changes 331
 opening for I/O wiring 170
 setting the real or virtual attribute for 171
 wiring channels of 172
I/O specific actions
 described 423
 inserting in charts 244
I/O variable comments, displaying or hiding 279
I/O wiring
 adding I/O devices to 169
 appearance of 156
 deleting I/O devices and conversions from
 171
 freeing channels in devices 174

grid view of 159
mapping channels of devices 172
opening I/O devices in 170
overview of 155
parameters component for 90
setting the real or virtual attribute, devices 171
tool, working with the 160
toolbar, main environment 20
tree view, described 157
wiring channels of devices 172

I/Os, simulating a panel of 320

identification
 configuration properties for 115
 defining for resources 55

identifiers
 inserting in POU s 201
 using defined words as 392

IF, THEN, ELSE, ELSIF, END_IF, ST basic statements 461

IF-THEN-ELSE structures, inserting (FC elements) 241

IL (Instruction List)
) operator for 482
 calling function blocks from (CAL operator) 485
 calling functions from 483
 delayed operations, described 475
 go to line for POU s 279
 JMP operator for 480
 labels, described 474
 LD operator for 477
 operator modifiers, described 474
 R operator for 479
 RET operator for 481
 S operator for 478
 ST operator for 478
 summary of operators 476
 syntax of programs in 473
 toolbar, language editor 193
 working with POU s, multi-language editor 260

importing
 variables data 93
 Workbench elements between projects 43

initial
 steps (SFC elements), described 396
 steps (SFC elements), inserting 215
 values for variables 150

initializing
 array elements 150
 structure fields 150

INSERT function 536

inserting
 actions (FC elements) 240
 blocks on the left (LD elements) 262
 blocks on the right (LD elements) 262
 coils (LD elements for FBD POU s) 271
 coils (LD elements) 262
 comments (FBD elements) 272
 comments (FC elements) 245
 connector links (FC elements) 244
 contact on the left (LD elements) 261
 contact on the right (LD elements) 261
 contacts (LD elements for FBD POU s) 270
 corners (FBD elements) 267
 DO-WHILE structures (FC elements) 242
 flow links (FC elements) 243
 function blocks (FBD elements) 267
 I/O specific actions (FC elements) 244
 identifiers in POU s 201
 IF-THEN-ELSE structures (FC elements) 241
 initial steps (SFC elements) 215
 jumps (SFC elements) 222
 jumps to labels (FBD elements) 268
 jumps to labels (LD elements) 262
 labels (FBD elements) 268
 labels (LD elements) 263
 LD vertical connections (FBD POU s) 270
 left power bars (LD elements for FBD POU s) 270
 links (LD elements) 264
 links (SFC elements) 221

links, connection (FBD elements) 267
networks 120
operators, functions, and function blocks in
 POUs 203
parallel blocks (LD elements) 262
parallel contacts (LD elements) 262
resources in configurations 112
resources in the link architecture view 50
returns (FBD elements) 269
returns (LD elements) 263
right power bars (LD elements for FBD
 POUs) 271
rows in the Dictionary grid 134
rungs (LD elements) 264
steps (SFC elements) 216
sub-programs (FC elements) 245
tests (FC elements) 240
transitions (SFC elements) 216
variables (FBD elements) 266
WHILE-DO structures (FC elements) 242
installation, directory structure of 29
instance symbols extra bytes 107
INTEGRAL function block 590
internal bindings
 accessing the list of 71
 defining 78
 deleting 80
 deleting resource links for 76
 described 71
 editing the contents of 80
 viewing for a resource 77
internal variable comments, displaying or hiding
 279
inverted
 coils, described 444
 contacts, described 440

J

JMP operator for IL 480
jumps
 described (FBD elements) 431
 described (LD elements) 450
 described (SFC elements) 398
 inserting (FBD elements) 268
 inserting (LD elements) 262
 inserting (SFC elements) 222

K

keywords, list of reserved 385

L

labels
 described (FBD elements) 431
 described (IL elements) 474
 described (LD elements) 450
 inserting (FBD elements) 268
 inserting (LD elements) 263
language editors
 appearance of 182
 debug toolbar in 187
 FBD toolbar in 195
 Flow Chart toolbar in 191
 for SFC, described 209
 IL toolbar in 193
 LD toolbar in 194
 managing the workspace of 197
 multi-language, described 253
 opening POU's from 206
 options toolbar in 186
 SFC breakpoints toolbar in 189
 SFC toolbar in 189

ST toolbar in 192
standard toolbar in 185
toolbars, summary of available 184
layers toolbar, main environment 19
LD (Ladder Diagram)
 aligning coils on rungs 264
 applying paste special in POUss 277
 changing coils and contacts types 263
 connection lines, described 436
 direct coils, described 443
 direct contacts, described 440
 displaying cursor coordinates 254
 falling edge detection (negative) coils,
 described 448
 falling edge detection (negative) contacts,
 described 442
 finding matching coils in POUss 278
 finding matching names in POUss 278
 inserting block on the left 262
 inserting block on the right 262
 inserting coils 262
 inserting contact on the left 261
 inserting contact on the right 261
 inserting jumps to labels 262
 inserting labels 263
 inserting links 264
 inserting parallel blocks 262
 inserting parallel contacts 262
 inserting returns 263
 inserting rungs 264
 inverted coils, described 444
 inverted contacts, described 440
 jumps, described 450
 labels, described 450
 monitoring output values in POUss 295
 multiple connections, described 437
 power rails, described 436
 reset coils, described 446
 return statements, described 449
 rising edge detection (positive) coils,
 described 447
 rising edge detection (positive) contacts,
 described 441
 set coils, described 445
 toolbar, language editor 194
 usage of blocks (functions and function
 blocks), described 451
 working with POUss, multi-language editor
 261
LD elements (for FBD POUss)
 inserting coils 271
 inserting contacts 270
 inserting LD vertical connections 270
 inserting left power bar 270
 inserting right power bar 271
LD operator for IL 477
LEFT function 537
left power bars (LD elements for FBD POUss),
 inserting 270
less than operator 511
less than or equal operator 510
level 2 windows
 editing, SFC elements 228
 viewing, FC elements 250
levels of programming, SFC editor 214
libraries
 creating 281
 described 281
 licensing third-party 282
 using in projects 282
license request, REQUEST_LICENSE function
 block 592
licensing third-party libraries 282
LIMIT function 539
line editing mode, Dictionary 131
link architecture view 48
linking
 configurations and networks 123
 resources for external bindings 86
 resources for internal bindings 74

links
 inserting, FBD elements 267
 inserting, LD elements 264
 inserting, SFC elements 221

list
 of external bindings 81
 of instructions for actions within steps 408
 of internal bindings 71

locating current step, step-by-step mode 302

locked variables
 locking 304
 unlocking 307

LOG function 540

logging system events
 opening log file 175
 starting 175
 viewing 176

M

magnification factor, adjusting in workspace 23

main format
 of SFC programs 395

managing
 configurations 110
 external bindings 81
 I/O wiring 160
 internal bindings 71
 POUs (Program Organization Unit) 96
 projects 32
 resources 48
 the hardware architecture view of a project 109
 the link architecture view of a project 48
 the workspace for language editors 197

manipulating POU's in resources 100

manual input of names, variables or blocks 265

mapping channels, I/O wiring 172

margins, including as printing option 340

MAX function 541

memory
 defining size for on-line changes 59
 requirements for online changes 331

menu bar options
 for FC editor 236
 for main environment 5
 for multi-language editor 256
 for SFC editor 211
 for simulator 323

MID function 542

MIN function 543

MLEN function 544

MOD function 546

modes
 for debugging a project 289
 for editing a project 32
 for edition the Dictionary 131
 for resources execution 298

monitoring
 information, generating for symbols 107
 output values of FBD/LD POU's 295

moving
 action blocks in execution order (SFC elements) 232
 configurations 112
 elements in FBD POU's 275
 networks 121
 POU's between sections and resources 100
 resources between configurations 113
 rows, Dictionary grid 135
 toolbars 14

multi-language editor
 described 253
 menu bar options for 256
 programming languages used with the 260
 selecting elements in the 273
 working with FBD POU's 265
 working with LD POU's 261
 working with ST/IL POU's 260

multiple connections in Ladder diagrams 437

multiplication operator 488

MUX4 function 547
MUX8 function 549

N

naming conventions
for defined words 392
for directly represented variables 387
for variables 385
NEG operator 512
networks
 creating 120
 described 120
 moving 121
 properties for resources 62
non-stored actions, within steps 406
not equal operator 515
NOT operator 514
NOT_MASK function 550

O

ODD function 551
OEM specific options 63
online changes
 declared variables, options for modifying using 329
 function block instances, options for modifying using 330
 I/O devices, options for modifying using 331
 memory requirements for 331
 modifying running resources using 332
 particular cases for 327
 performing 327
 types, bindings, and resource properties, options for modifying using 331
 variables, options for modifying using 329
online mode for debugging 289

opening
 I/O devices in I/O wiring 170
 level 2 windows (FC elements) 250
 POUs in language editors 206
 projects 36
 spy lists 319
 the I/O wiring tool 155
 variable groups 91
operating system priority, SET_PRIORITY
 function block 594
operative states of resources 48
operator modifiers (IL elements), described 474
operators
) for IL programs 482
 1 gain 494
 addition 489
 AND 495
 ANY_TO_BOOL 496
 ANY_TO_DINT 499
 ANY_TO_REAL 501
 ANY_TO_SINT 498
 ANY_TO_STRING 504
 ANY_TO_TIME 502
 CAL for IL 485
 division 492
 equal 505
 greater than 508
 greater than or equal 507
 inserting in POUs 203
 JMP for IL programs 480
 LD for IL programs 477
 less than 511
 less than or equal 510
 multiplication 488
 NEG 512
 NOT 514
 not equal 515
 OR 516
 R for IL programs 479
 RET for IL programs 481
 S for IL programs 478
 ST for IL programs 478

- subtraction 491
summary of IL 476
summary of standard 487
TMR 517
XOR 518
- options
for printing of project items 340
toolbar in language editors 186
toolbar in main environment 20
- OR operator 516
- OR_MASK function 553
- oriented links, description of 398
- output values, monitoring (FBD/LD POU)s 295
- output window
clearing the contents of the 24
displaying errors and build information 24
-
- P**
- page numbering, specifying in printing options 340
- panel of I/Os, simulating a 320
- parallel blocks, inserting (LD elements) 262
- parallel contacts, inserting (LD elements) 262
- parameters
grid, described 143
I/O wiring and defined words components of 90
network, for resources 62
tree, described 128
- parentheses in ST programs 455
- password protection
for configuration access control 118
for POU access control 104
for project access control 41
for resource access control 64
- paste special, applying in LD POU's 277
- pasting
elements (variables) in the Dictionary grid 138
POU's 100
resources 52
- performing online changes 327
- popup menus, accessing 25
- POU's (Program Organization Unit)
accessing details for previous versions of 366
building/rebuilding code for 346
checking in 363
cleaning 348
comparing versions of 365
creating history reports for 366
creating in resources 99
defining access control for 104
editing descriptions of 108
finding and replacing elements in 207
getting previous versions of 365
inserting identifiers in 201
inserting operators, functions, and function blocks in 203
managing 96
manipulating in resources 100
opening from language editors 206
stepping in 302
stopping builds 348
unlocking 104
viewing the history of 364
- POW function 554
- power rails for Ladder diagrams 436
- preferences, setting for opening and exiting 26
- previewing project printing 342
- previous versions, accessing history details for 366
- printing
defining options for 340
previewing project document 342
project items 337
projects 40
selecting project items for 338

specifying document range for 342
the Dictionary grid 141

process control

- DERIVATE function block 587
- HYSTER function block 589
- INTEGRAL function block 590
- STACKINT function block 600

producer groups, external bindings

- defining 83
- deleting 85
- editing 85

producing variables, viewing for internal bindings 77

production error variables 68

programming languages

- for use with function blocks 99
- for use with functions 98
- for use with programs 96
- used with the multi-language editor 260

programming levels, SFC editor 214

programs

- changing hierarchy level for SFC child 103
- creating in resources 99
- defining access control for 104
- described 368
- hierarchy in the SFC language 415
- inserting comments in literal language 392
- manipulating in resources 100
- working with 96

project architecture

- child SFC POUs 370
- cyclic and sequential operations 369
- description languages for programs 375
- execution rules for cycles 376
- FC sub-programs 371
- function blocks 373
- functions 371
- overview of 368
- programs 368

project tree view 353

projects

- accessing details for previous versions of 366
- adding descriptions to 40
- browsing POUs of 357
- building/rebuilding code for 345
- checking in 363
- cleaning 348
- closing 36
- comparing versions of 365
- controlling access for 41
- creating 34
- creating history reports for 366
- defining dependencies on libraries for 282
- editing modes for 32
- filenames for 36
- getting previous versions of 365
- hardware architecture view of 109
- link architecture view of 48
- managing 32
- modes for testing 295
- opening 36
- opening with a command line 36
- previewing printing documents for 342
- printing 40
- printing items in 337
- renaming 39
- saving changes to 39
- security state of resources within 36
- selecting items for printing 338
- stopping builds 348
- storage location of 29
- templates for 34
- using libraries in 282
- viewing the history of 364

promoting SFC child programs 103

properties

- for configuration identification 115
- for resource custom parameters 63
- for resource identification 55

for target access, configurations 118
for target definition, configurations 117
pulse actions, within steps 405

R

R operator for IL 479
R_TRIGGER function block 591
RAND function 555
real
 attribute, setting for I/O devices 171
 constant expressions 382
 variables 390
real-time execution mode, resources 298
rearranging variables in spy list 318
rebuilding
 code for projects 345
 POUs 346
refresh rate, setting for resources 290
refreshing status of resources 290
reloading of last project when starting, setting 26
removing
 breakpoints for steps and transitions (SFC) 311
 breakpoints, step-by-step mode 301
 code stored on targets 336
 variables from spy list 318
renaming
 projects 39
 resources 51
 SFC elements 224
 structures 130
renumbering
 addresses in the Dictionary grid 140
 elements in FC charts 249
 elements in SFC charts 233
REPEAT, UNTIL, END_REPEAT, ST basic statements 464
REPLACE function 556
replacing elements, Dictionary grid 137
repository path for version source control 359

REQUEST_LICENSE function block 592
reserved keywords, list of 385
reset coils, described 446
resizing
 columns and rows in the Dictionary 132
 elements in FBD POU 274
resource links (internal bindings)
 deleting 76
 hiding and showing 77
resources
 accessing details for previous versions of 366
 adding descriptions to 66
 appearance of 48
 building code for 347
 checking in 363
 cleaning 348
 comparing versions of 365
 compilation options for 55
 copying 51
 creating 50
 creating history reports for 366
 cycle-to-cycle execution mode for 299
 defining access control for 64
 defining custom parameters for 63
 defining the identification (general) properties of 55
 defining the network parameters for 62
 deleting 53
 deleting data links between (internal bindings) 76
 downloading code to targets for 293
 editing links for external bindings 87
 editing the properties of 54
 executing in step-by-step mode 299
 execution modes for 298
 getting previous versions of 365
 inserting in configurations 112
 inserting in the link architecture view 50
 linking for external bindings 86
 linking for internal bindings 74
 managing 48

moving between configurations 113
operative states of 48
pasting 52
properties, options for modifying using online changes 331
real-time execution mode for 298
renaming 51
run-time settings for 59
running, modifying using online changes 332
setting cycle time, debug mode 303
setting cycle time, edition mode 59
starting and stopping execution of resources 297
status information for 290
stopping builds 348
unlocking 64
viewing the history of 364
window workspace of, described 49

RET operator for IL 481

return statements

- for FBD 431
- for LD 449
- for ST 460

return symbols

- inserting, FBD elements 269
- inserting, LD elements 263

reusing functions and function blocks 281

RIGHT function 558

right power bars (LD elements for FBD POU), inserting 271

rising edge detection (positive)

- coils, described 447
- contacts, described 441

ROL function 559

ROR function 560

row-level validation 153

rows, duplicating in the Dictionary grid 139

RS function block 593

rules for variables 385

RUN resource state 291

run-time

- logging of system events 175
- settings for resources 59
- viewing of system events 176

rungs, inserting (LD elements) 264

S

S operator for IL 478

saving

- before exiting, setting to prompt 26
- changes to projects 39
- changes to spy lists 318

search options, defining for finding cross references 358

security

- for configurations 118
- for POU 104
- for projects 41
- for resources 64
- state of resources within projects 36

SEL function 562

selecting

- elements in the multi-language editor 273
- project items for printing 338
- rows and elements in the Dictionary 132
- variables in a spy list 317

semaphore manipulation, SOFT_POINT_READ function block 596

semaphore manipulation, SOFT_POINT_WRITE function block 597

sequential operations 369

set coils, Ladder diagrams 445

SET_PRIORITY function block 594

setting

- access control for resources 64
- breakpoints for steps and transitions (SFC) 311
- breakpoints, step-by-step mode 301
- cycle time of resources, debug mode 303
- prompting to save before exiting 26

real or virtual attributes for I/O devices 171
refresh rate for resources 290
reloading of last project when starting 26
SFC (Sequential Function Chart)
 actions within steps, described 407
 adding action blocks to level 2 programming 229
 breakpoints on step activation 312
 breakpoints on step deactivation 313
 breakpoints on transition 314
 breakpoints toolbar in language editors 189
 changing hierarchy level of child programs 103
 creating child programs 102
 deleting action blocks from level 2 programming 233
 deleting branches from convergences/divergences 220
 dynamic behavior, described 414
 editing code for transitions 231
 editing level 2 programming 228
 editor, described 209
 GFREEZE statement in actions 470
 GKILL statement in actions 469
 goto steps or transitions 227
 GRST statement in actions 471
 GSTART statement in actions 468
 GSTATUS statement in actions 472
 hierarchy of programs 415
 hierarchy restrictions for programs 96
 inserting initial steps 215
 inserting jumps 222
 inserting links 221
 inserting new branches in convergences/divergences 219
 inserting steps 216
 inserting transitions 216
 linking and placing convergences/divergences 217
 main format of programs 395
 menu bar options for editor 211
 programming levels of editor 214
renaming elements 224
renumbering elements in charts 233
setting/removing breakpoints for steps and transitions 311
toolbar for language editors 189
SHL function 563
short integer
 constant expressions 381
 variables 390
showing
 resource links, internal bindings 77
 toolbars in the main environment 14
SHR function 564
SIG_GEN function block 595
signal generation
 BLINK function block 577
 SIG_GEN function block 595
simulating a panel of I/Os 320
simulation mode
 for debugging a project 289
 starting for a project 295
simulator
 appearance of 322
 displaying I/O device window headers in the 325
 manipulating the browser of the 326
 menu bar options for the 323
 toolbar options for the 324
SIN function 565
single convergences, described 400
single divergences, described 400
single-resource editing mode 32
SOFT_POINT_READ function block 596
SOFT_POINT_WRITE function block 597
sorting
 run-time system events 176
 the Dictionary grid 138
source control, for versions of Workbench elements 359
splitting workspace of language editors 197
spreadsheets, importing variables data using 93

| | |
|--|--|
| spy lists | CTD 581 |
| accessing variables list for 316 | CTU 582 |
| adding variables to 316 | CTUD 583 |
| forcing values of variables in 319 | DBG_CLR_GET_ERR 585 |
| opening 319 | DBG_CLR_SET_ERR 585 |
| rearranging variables in 318 | DBG_GET_ERR 586 |
| removing variables from 318 | DBG_SET_ERR 586 |
| saving 318 | DERIVATE 587 |
| selecting variables in 317 | EVENT 588 |
| SQRT function 566 | F_TRIG 588 |
| SR function block 598 | HYSTER 589 |
| ST (Structured Text) | INTEGRAL 590 |
| assignment basic statements for 459 | R_TRIG 591 |
| calling function blocks from 457 | REQUEST_LICENSE 592 |
| calling functions from 456 | RS 593 |
| CASE, OF, ELSE, END_CASE basic statements for 462 | SET_PRIORITY 594 |
| EXIT basic statements for 466 | SIG_GEN 595 |
| expressions and parentheses in 455 | SOFT_POINT_READ 596 |
| extensions for SFC child execution 467 | SOFT_POINT_WRITE 597 |
| FOR, TO, BY, DO, END_FOR basic statements for 465 | SR 598 |
| go to line for POU's 279 | STACKINT 600 |
| IF, THEN, ELSE, ELSIF, END_IF basic statements for 461 | TLP_GET_DINT 601 |
| main syntax of programs in 453 | TLP_GET_REAL 602 |
| REPEAT, UNTIL, END_REPEAT basic statements for 464 | TLP_GET_SINT 603 |
| return basic statements for 460 | TLP_GET_STRING 604 |
| toolbar in the language editors 192 | TLP_GET_TLP 605 |
| WHILE, DO, END WHILE basic statements for 463 | TLP_SET_DINT 606 |
| working with POU's in the multi-language editor 260 | TLP_SET_REAL 607 |
| ST operator for IL 478 | TLP_SET_SINT 608 |
| STACKINT function block 600 | TLP_SET_STRING 609 |
| standard function blocks | TOF 609 |
| ALARM 575 | TON 610 |
| AVERAGE 575 | TP 611 |
| BLINK 577 | URCV_S 612 |
| CMP 578 | USEND_S 613 |
| CONNECT 579 | standard function blocks, summary of 573 |
| standard functions | standard functions |
| ABS 522 | ABS 522 |
| ACOS 523 | ACOS 523 |
| AND_MASK 524 | AND_MASK 524 |
| ASCII 525 | ASCII 525 |
| ASIN 526 | ASIN 526 |

ATAN 527
 CHAR 528
 COS 530
 CURRENT_ISA_DATE 531
 DELETE 532
 EXPT 533
 FIND 534
 INSERT 536
 LEFT 537
 LIMIT 539
 LOG 540
 MAX 541
 MID 542
 MIN 543
 MLEN 544
 MOD 546
 MUX4 547
 MUX8 549
 NOT_MASK 550
 ODD 551
 OR_MASK 553
 POW 554
 RAND 555
 REPLACE 556
 RIGHT 558
 ROL 559
 ROR 560
 SEL 562
 SHL 563
 SHR 564
 SIN 565
 SQRT 566
 SUB_DATE_DATE 567
 summary of 521
 TAN 569
 TRUNC 570
 XOR_MASK 571
 standard IEC 61131 types, available for
 programming 377
 standard operators
 1 gain 494
 addition 489
 AND 495
 ANY_TO_BOOL 496
 ANY_TO_DINT 499
 ANY_TO_REAL 501
 ANY_TO_SINT 498
 ANY_TO_STRING 504
 ANY_TO_TIME 502
 division 492
 equal 505
 greater than 508
 greater than or equal 507
 less than 511
 less than or equal 510
 multiplication 488
 NEG 512
 NOT 514
 not equal 515
 OR 516
 subtraction 491
 summary of 487
 TMR 517
 XOR 518
 standard toolbar
 in language editors 185
 in main environment 15
 starting
 events logger for run-time system events 175
 execution of resources 297
 status
 bar, displaying and hiding 25
 information, displaying for resources 290
 of elements for version control 359
 step-by-step mode
 executing resources in 299
 locating current step of 302
 removing breakpoints for 301
 setting breakpoints for 301
 stepping in POU's for 302
 STEPPING resource state 291
 STEPPING_ERROR resource state 291

steps (SFC elements)
actions within, described 404
attaching action blocks to 229
described 396
inserting 216
STOP resource state 291
stopping
builds of projects, resources, and POU's 348
execution of resources 297
storage location of projects 29
string
constant expressions 383
variables 391
string manipulation
ASCII function 525
CHAR function 528
DELETE function 532
FIND function 534
INSERT function 536
LEFT function 537
MID function 542
MLEN function 544
REPLACE function 556
RIGHT function 558
structures
basic or user types, described 380
initializing fields of 150
sub-programs (Flow Chart)
described 422
inserting 245
SUB_DATE_DATE function 567
subtraction operator 491
switching
to the Dictionary view 125
to the hardware architecture view 109
to the link architecture view 48
symbols
downloading complete or reduced table of 55
generating monitoring information for 107

syntax
of IL programs 473
of ST programs 453
verification rules for Flow Chart 426

system
accessing variables for 307
events, logging of run-time 175
events, viewing of run-time 176

T

TAN function 569
targets
cleaning code stored on 336
defining compilation options for 55
defining control access for 118
specifying for configurations 117
templates, for libraries 281
templates, specifying for projects 34
testing projects 295
tests, inserting (FC elements) 240
third-party libraries, licensing 282
TIC code, generating 55
time operations
CURRENT_ISA_DATE function 531
SUB_DATE_DATE function 567
TOF function block 609
TON function block 610
TP function block 611
timer
constant expressions 383
variables 391
timing information, accessing 307
title bar of main environment 4
TLP variables
defining 146
TLP_GET_DINT function block 601
TLP_GET_REAL function block 602
TLP_GET_SINT function block 603
TLP_GET_STRING function block 604
TLP_GET_TLP function block 605

- TLP_SET_DINT function block 606
TLP_SET_REAL function block 607
TLP_SET_SINT function block 608
TLP_SET_STRING function block 609
TMR operator 517
TOF function block 609
TON function block 610
toolbars
 available in language editors 184
 debugging, main environment 17
 docking, moving, and showing 14
 I/O wiring, main environment 20
 layers view, main environment 19
 options, main environment 20
 simulator 324
 standard, main environment 15
 version source control, main environment 20
 window buttons, main environment 19
tooltips
 displaying for function blocks 267
 displaying for variables 266
TP function block 611
transitions
 clearing in SFC 311
 conditions attached to (ST or LD), described 410
 editing code for (SFC elements) 231
 forcing clearing of in SFC 315
 in SFC, described 397
 inserting in SFC 216
 programming for conditions in IL 411
 programming for conditions in LD 411
 programming for conditions in ST 410
tree view
 for a project 353
 for I/O wiring 157
TRUNC function 570
types
 arrays, described 379
 available standard IEC 61131 types 377
 grid in the Dictionary 144
 modifying for online changes 331
structures, described 380
tree, creating structures in 129
tree, deleting structures from 130
tree, described 129
tree, renaming structures in 130
-
- U**
- unlocking
 POUs with access control 104
 resources with access control 64
 variables 304
unwiring channels in I/O devices 174
uploading Workbench elements from targets 46
URCV_S function block 612
USEND_S function block 613
user types
 arrays, described 379
 structures, described 380
using libraries in projects 282
-
- V**
- validation
 at cell level 153
 at database level 154
 at row level 153
values, forcing for variables 304
variable bindings
 defining for external bindings 88
 defining for internal bindings 78
 deleting for external bindings 89
 deleting for internal bindings 80
 described 67
 editing for external bindings 89
 editing for internal bindings 80
 external, described 81
 internal, described 71
 linking resources for internal bindings 74
 modifying for online changes 331

variable groups
 creating 91
 managing 91
 opening 91
 producing for external bindings 83

variables
 accessing spy list for 316
 adding to spy list for 316
 attributes and directions for 389
 Boolean 390
 computer allocated hidden, effect on online changes 330
 declared, modifying in online changes 329
 directly represented 387
 displaying comments for 279
 displaying tooltips for 266
 double integer 390
 for binding errors 68
 forcing the values of 304
 forcing values of, spy list 319
 grid, described 142
 importing and exporting 93
 initial values for 150
 inserting, FBD elements 266
 inserting in POU's 201
 locking and unlocking 304
 modification of during online changes 329
 opening a spy list with 319
 real 390
 rearranging in spy list 318
 removing from spy list 318
 rules for 385
 saving spy list with 318
 selecting in spy list 317
 short integer 390
 string 391
 system, accessing 307
 timer 391
 tree, described 127

variables, defining TLP 146

version information, viewing 307

version source control
 accessing history details for previous versions 366
 checking in Workbench elements for 363
 clearing the status of 359
 creating history reports 366
 described 359
 getting previous versions of Workbench elements 365
 repository path for 359
 toolbar in main environment 20
 viewing history of Workbench elements 364

view
 for variable bindings 67
 of the Dictionary 125
 of the hardware architecture 109
 of the I/O wiring 155
 of the link architecture 48

viewing
 breakpoints (step-by-step mode) 307
 history of Workbench elements from version source control 364
 internal bindings 77
 level 2 windows of FC chart elements 250
 run-time system events 176
 the lock status of variables 304
 the project tree 353
 version information 307
 virtual attribute, setting for I/O devices 171

W

WHILE, DO, END WHILE, ST basic statements 463

WHILE-DO structures, inserting (FC charts) 242

window
 buttons toolbar in main environment 19
 headers, displaying for I/O devices 325

wiring
 channels in I/O wiring 172
 tool, opening 155

Workbench elements
 exporting between projects 43
 importing between projects 43
 uploading from targets 46
Workbench, overview of 1
working with
 function blocks 99
 functions 98
 programs 96
workspace
 adjusting zoom in 23
 managing for language editors 197
 of a resource window 49

X

X-Y ratio, setting for language editors 197
XOR operator 518
XOR_MASK function 571

Copyright

Information in these pages is subject to change without notice and does not represent a commitment on the part of Emerson Process Management. No part of these pages may be reproduced in any form or by any means, electronic or mechanical, for any purpose without the express written permission of Emerson Process Management.

© 2002-2008 Remote Automation Solutions, division of Emerson Process Management. All rights reserved.

Product or company names included in these pages are trademarks or registered trademarks of their respective holders.

All logos and links used in this guide are, to the best of our knowledge, included with the permission of the owner - if this is not the case, please let us know immediately.

Any changes made to documentation issued by Emerson Process Management without prior permission of Emerson Process Management (in writing) will void any responsibilities and liabilities normally associated with its contents.

21080227ENGF70WWP70HC13

