

# AADvance-Trusted SIS Workstation software User Guide 

Rockwell Automation Publication ICSTT-UMOO2D-EN-D - October 2023
Supersedes Rockwell Automation Publication ICSTT-UMOO2C-EN-D - January 2022

## Rockwell Automation

## Important User Information

Read this document and the documents listed in the additional resources section about installation, configuration, and operation of this equipment before you install, configure, operate, or maintain this product. Users are required to familiarize themselves with installation and wiring instructions in addition to requirements of all applicable codes, laws, and standards.
Activities including installation, adjustments, putting into service, use, assembly, disassembly, and maintenance are required to be carried out by suitably trained personnel in accordance with applicable code of practice.
If this equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.
The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.
Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.


WARNING: Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.


ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

IMPORTANT Identifies information that is critical for successful application and understanding of the product.
Labels may also be on or inside the equipment to provide specific precautions.


SHOCK HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.


ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).
Rockwell Automation recognizes that some of the terms that are currently used in our industry and in this publication are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.
Introducing the AADvance-Trusted SIS Workstationsoftware
System Security ..... 22
Network Firewall ..... 23
Windows Firewall ..... 24
Application View ..... 24
Language Editor ..... 26
Applying best practices ..... 27
Reset Container and Shape Settings in FBD POUs ..... 34
Reset Visual Settings in LD POUs ..... 35
Block Library ..... 36
Properties Window ..... 37
Locked Variables Viewer ..... 38
Toolbox ..... 39
Variable Selector ..... 40
Creating Variables from the Variable Selector ..... 42
Editing Variables from the Variable Selector. ..... 43
Delete Variables from the Variable Selector ..... 44
Block Selector ..... 44
Parameters Display ..... 46
Parameters View ..... 46
Generating Documentation ..... 48
Find and Replace Utility ..... 52
Quick Find ..... 52
Quick Replace. ..... 53
Spy Lists ..... 53
External Tools ..... 55
Working in the Development Environment ..... 56
Zooming the Contents of Active POUs ..... 56
Displaying the Output Window. ..... 57
Use the Error List ..... 58
Arranging and Docking Windows ..... 58
Arrange the windows layout. ..... 60
Searching in the Development Environment. ..... 61
Navigating in the Development Environment ..... 61
Customize Toolbars ..... 62
Create Toolbars ..... 63
Customize Commands ..... 63
Specify the command display options ..... 65
Importing and Exporting Settings ..... 65
Export Selected Environment Settings ..... 65
Import Selected Environment Settings ..... 66
Reset all Settings ..... 67
Operations Summary ..... 68
Development Environment Keyboard Shortcuts ..... 68
Options for the Development Environment ..... 70
Chapter 1
Setting Environment Options ..... 70
Find and replace display settings ..... 71
Import and export settings ..... 72
Shortcut keyboard combinations ..... 73
Startup ..... 74
Tabs and windows options ..... 74
Specify Project Options ..... 75
Build options ..... 76
Online Settings ..... 77
Specify Block Library settings ..... 77
Specify Trusted settings ..... 77
Specify Cross Reference Browser settings ..... 78
Specify Documentation Generator options ..... 78
Setting Grid Options ..... 79
Arrays View ..... 79
Defined Words View ..... 80
Dictionary View ..... 80
Parameters Grid ..... 81
Structures View ..... 81
Variable Groups View ..... 82
Variable Selector Settings ..... 83
Define I/O Wiring Settings ..... 83
Setting IEC Language Options ..... 84
Function Block Diagram Settings ..... 84
Ladder Diagram Settings. ..... 87
Sequential Function Chart Settings ..... 88
Structured Text Settings ..... 90
Setting ISaVIEW Options ..... 92
ISaVIEW Animation Settings ..... 92
ISaVIEW Edition Settings. ..... 92
ISaVIEW Objects Settings ..... 93
Defining Spy List Settings ..... 100
Offline Grid Settings ..... 100
Online Grid Settings ..... 100
Description Window ..... 101
Chapter 2
Working with AADvanceCreating an AADvance Project104
Import an AADvance Project. ..... 105
Import an AADvance Project with Library ..... 106
Upgrade the Series 9000 target definition ..... 106
Import an AADvance Project with Library with outdated Series
9000 Target Definition ..... 107
Controllers ..... 108
Programs ..... 112
Functions ..... 113
Function Blocks ..... 115
Variables ..... 117
AADvance Libraries ..... 118
Opening an AADvance Project ..... 120
Closing a Project ..... 120
Setting Project Access Control ..... 121
Setting Target Access Control ..... 122
Import and Export Variables Data ..... 123
Generating Code ..... 125
Building an AADvance Project. ..... 125
Verifying POU Syntax ..... 126
Cleaning a Project ..... 127
Run an Application Online ..... 127
Download code to controllers ..... 129
Performing Updates ..... 130
Starting and Stopping Controllers ..... 133
Clearing an Application ..... 134
Connect with controllers ..... 134
Modifying Cycle Timing ..... 136
Disconnecting from Controllers ..... 137
Simulate an application ..... 137
Pausing Projects and Controllers ..... 139
Forcing the Values of Variables ..... 139
Monitoring the Values of Variables ..... 140
Accessing the Values of System Variables ..... 141
Name conventions and limitations ..... 142
File System Paths ..... 146
Version Control ..... 147
Define a Repository ..... 148
Check In Elements to Version Control ..... 149
Checking Out Elements from Version Control. ..... 150
Undoing Check Outs of Elements ..... 151
Reviewing Pending Checkins ..... 151
Getting the Latest Versions of Elements ..... 152
Labeling a Version ..... 153
Archive and Restore Repository Projects ..... 153
Accessing Repository Projects ..... 154
To access a repository in a shared folder ..... 155
To transfer a repository to another computer ..... 156
To access a project stored in the local repository of another computer ..... 156
Publishing Libraries ..... 157
Working Offline and Reconnecting to a Repository ..... 158
View the History of Elements ..... 159
Refreshing the Version Control Status of Elements ..... 159
Dictionary ..... 160
Arrays Grid ..... 161
Structures Grid ..... 162
Defined Words Grid ..... 163
Variables Grid ..... 164
Quick Declaration of Variables ..... 167
Cross Reference Browser ..... 168
Defined Word Cross References ..... 169
Function and Function Block Cross References ..... 170
Program Cross References ..... 170
Variable Cross References ..... 171
Variable Dependencies ..... 171
Controller Status ..... 172
Front Panel Status Indicators ..... 173
Communication View ..... 175
Defining Ethernet Ports ..... 177
Using Modbus Communications ..... 178
Defining a Modbus Slave ..... 180
Defining a Modbus Master ..... 181
Specifying the Workbench and OPC Communications ..... 191
Mapping OPC Variables ..... 191
Map SOE Variables ..... 193
Configuring Serial Ports ..... 193
Configuring the Controller for SNTP ..... 194
Configure a Controller as an SNTP Client. ..... 195
Configure a Controller as an SNTP Server ..... 197
Using a Controller as an SNTP Client and Server ..... 199
Configuring Serial Tunneling. ..... 199
Defining Bindings ..... 200
Consume Status Variables for Bindings ..... 203
Binding Process ..... 203
Creating Links between Controllers ..... 205
Mapping Variable Bindings ..... 208
Use CIP over EtherNet/IP ..... 210
Creating the CIP Produce Link. ..... 212
Creating CIP Consume Links ..... 213
Update Ethernet ports of CIP produce and CIP consume links ..... 216
Resolve invalid CIP links ..... 216
Configuring Peer-to-Peer Communications ..... 217
Peer-to-Peer Communication Cycle. ..... 218
Configuring a Peer-to-Peer Network ..... 219
Configuring Peer-to-Peer Data Blocks ..... 223
Defining the Quality of Service ..... 229
TCP Negotiation ..... 231
Equipment View ..... 231
Hardware Redundancy ..... 233
Input Voting ..... 233
Output Voting ..... 234
Configuring the Controller ..... 234
Configuring Processor Variables ..... 236
Status Processor Variables ..... 237
Control Processor Variables ..... 245
RTC Processor Variables and the Real Time Clock ..... 248
Configure the Controller I/O ..... 259
Standard or Mixed System Type ..... 260
Eurocard or Mixed System Type ..... 261
Configuring I/O Modules for AADvance Applications ..... 261
T9K_TA_GROUP_STATUS ..... 263
Series 9000 Structure Data Types ..... 264
Digital Input Modules ..... 265
Analog Input Modules ..... 269
Digital Output Modules ..... 278
Analog Output Modules ..... 286
AADvance Discover ..... 295
AADvance Discover Features ..... 297
Ethernet Forwarding ..... 299
Compiler Verification ..... 300
FBD Language ..... 301
FBD Diagram Main Format ..... 302
Execution Order of FBD Programs ..... 303
Debugging FBD Programs ..... 304
FBD Elements ..... 305
Blocks ..... 306
FBD Variables ..... 307
Vertical Bars ..... 308
Labels ..... 309
Jumps ..... 310
Returns ..... 310
Rungs ..... 311
Left Power Rails ..... 312
Right Power Rails ..... 312
Coils ..... 312
Contacts ..... 316
Regions ..... 320
Comments ..... 320
FBD Keyboard Shortcuts ..... 321
LD Language ..... 322
Debugging LD programs ..... 324
LD Elements ..... 326
LD Rungs ..... 328
Blocks ..... 329
Coils ..... 332
Contacts ..... 338
Jumps ..... 342
Returns ..... 344
Branches ..... 345
Selecting LD Elements ..... 346
LD Keyboard Shortcuts. ..... 347
ST Language ..... 349
ST Main Syntax. ..... 349
Expressions and Parentheses ..... 351
Calling Functions ..... 352
Calling Function Blocks ..... 353
Debugging ST programs ..... 353
ST Basic Elements and Statements ..... 354
Assignments ..... 355
CASE Statement ..... 355
EXIT Statement. ..... 356
FOR Statement ..... 357
IF-THEN-ELSIF-ELSE-END_IF Statement ..... 358
REPEAT Statement ..... 359
RETURN Statement ..... 360
WHILE Statement ..... 361
ST Extensions ..... 362
GSTART Statement in SFC Action ..... 363
GFREEZE Statement in SFC Action ..... 365
GSTATUS Statement in SFC Action ..... 365
GKILL Statement in SFC Action ..... 366
GRST Statement in SFC Action ..... 366
ST Keyboard Shortcuts ..... 368
SFC Language ..... 370
SFC Main Format ..... 370
SFC Execution Behavior ..... 371
SFC Program Hierarchy ..... 372
Child SFC POUs ..... 373
Debugging SFC programs ..... 374
Breakpoint on Step Activation ..... 375
Breakpoint on Step Deactivation ..... 376
Breakpoint on Transition ..... 377
Transition Clearing Forcing ..... 377
SFC Elements ..... 378
Steps. ..... 379
Transitions ..... 381
Sequence Controls ..... 382
Jumps to Steps ..... 385
Coding Action Blocks for Steps ..... 386
Boolean Actions ..... 387
Pulse Actions ..... 388
Non-Stored Actions ..... 389
SFC Actions ..... 390
Coding Conditions for Transitions. ..... 391
Conditions Programmed in ST ..... 392
Conditions Programmed in LD ..... 393
Calling Functions from Transitions ..... 393
Calling Function Blocks from Transitions ..... 394
SFC Keyboard Shortcuts ..... 394
Language Reference. ..... 395
Programs ..... 395
Functions ..... 396
Function Blocks ..... 397
Execution Rules ..... 398
Reserved Keywords ..... 399
Variables ..... 399
Directly Represented Variables ..... 400
Defined Words ..... 401
Data Types ..... 402
Elementary IEC 61131-3 Types ..... 402
Derived Types: Arrays ..... 412
Derived Types: Structures ..... 413
Operators ..... 413
1 GAIN ..... 414
Addition ..... 415
AND ..... 416
ANY_TO_BOOL ..... 417
ANY_TO_BYTE ..... 417
ANY_TO_DATE ..... 418
ANY_TO_DINT ..... 418
ANY_TO_DWORD ..... 419
ANY_TO_LREAL ..... 420
ANY_TO_INT. ..... 420
ANY_TO_LINT ..... 421
ANY_TO_LWORD ..... 422
ANY_TO_REAL ..... 423
ANY_TO_SINT. ..... 423
ANY_TO_STRING ..... 424
ANY_TO_TIME. ..... 425
ANY_TO_UDINT ..... 425
ANY_TO_UINT ..... 426
ANY_TO_ULINT ..... 426
ANY_TO_USINT ..... 427
ANY_TO_WORD ..... 428
Division ..... 428
Equal ..... 429
Greater Than ..... 430
Greater Than or Equal ..... 431
Less Than ..... 432
Less Than or Equal ..... 433
Multiplication ..... 434
NEG ..... 435
NOT ..... 435
Not Equal ..... 436
OR ..... 436
Subtraction ..... 437
XOR ..... 438
Functions ..... 439
ABS ..... 439
ACOS ..... 440
AND_MASK ..... 440
ASCII ..... 441
ASIN ..... 442
ATAN ..... 442
CHAR ..... 443
COS ..... 443
CURRENT_ISA_DATE ..... 444
DELETE ..... 444
EXPT ..... 445
FIND. ..... 446
INSERT ..... 446
LEFT ..... 447
LIMIT ..... 448
LOG ..... 449
MAX ..... 449
MID ..... 450
MIN ..... 451
MLEN ..... 451
MOD ..... 452
MUX4 ..... 453
MUX8 ..... 453
NOT_MASK ..... 454
ODD ..... 455
OR_MASK ..... 455
POW ..... 456
RAND ..... 457
REPLACE ..... 457
RIGHT ..... 458
ROL (Rotation Left) ..... 459
ROR (Rotation Right) ..... 460
SEL ..... 460
SHL (Shift Left Arithmetic or Shift Left Signed) ..... 461
SHR (Shift Right Arithmetic or Shift Right Signed) ..... 462
SIN ..... 462
SQRT ..... 463
SUB_DATE_DATE ..... 463
TAN ..... 464
TRUNC ..... 465
XOR_MASK ..... 465
Function Blocks ..... 466
AVERAGE ..... 466
BLINK ..... 467
CMP ..... 468
CTD ..... 468
CTU ..... 469
CTUD ..... 470
DERIVATE ..... 471
F_TRIG ..... 472
HYSTER ..... 473
INTEGRAL ..... 473
LIM_ALRM ..... 474
R_TRIG ..... 475
RS. ..... 476
SIG_GEN ..... 476
SR. ..... 477
STACKINT ..... 478
TOF ..... 480
TON ..... 480
TP. ..... 481
Series 9000_L Function Blocks ..... 482
KvbConsNetStatus ..... 482
KvbProdNetStatus ..... 483
MathErr ..... 483
ModbusServiceStatus ..... 484
SoeServiceStatus ..... 485
Glossary ..... 486
A - C ..... 486
D - H. ..... 492
I-N ..... 498
O-R ..... 502
S - Z ..... 506
License AADvance applications ..... 512
Obtain an authorized Full license ..... 513
Remove an authorized license ..... 513
AADvance Manuals ..... 514
Chapter 3
Working with Trusted Applications
Creating a Trusted Project ..... 515
To import a Trusted Toolset project ..... 517
Archive a Trusted project ..... 517
Restore a Trusted project ..... 518
Copy a Trusted project ..... 519
To access the properties for a Trusted project ..... 519
Project Security ..... 519
Introduction ..... 519
Permission-based control ..... 519
Intellectual property control ..... 520
Use FactoryTalk Security with AADvance-Trusted SIS Workstation software ..... 520
Security policy considerations ..... 521
AADvance-Trusted SIS Workstation software and FactoryTalk Security ..... 521
Securable Actions supported by FactoryTalk Security. ..... 522
Trusted Devices ..... 523
Enable power flow debugging ..... 524
Programs ..... 525
Export a program ..... 526
Add a program from an exchange file ..... 526
Functions ..... 527
Export a function ..... 528
Add a function from an exchange file ..... 528
Function Blocks ..... 529
Export a function block ..... 530
Add a function block from an exchange file ..... 530
Variables ..... 531
Trusted Libraries ..... 531
Opening a Trusted Project ..... 532
Closing a Project ..... 533
Setting Project Access Control ..... 533
Import and Export Variables Data ..... 534
Use Modbus Communications ..... 535
Configure a Modbus Slave ..... 538
Configure a Modbus Master ..... 541
Define Communications and Status Reporting ..... 543
Broadcast Messages to Slaves ..... 544
Report Statistics Data ..... 546
Configure a Slave for a Modbus Master ..... 547
Validate Changes to Modbus Master Configurations ..... 553
Generating Code ..... 554
Build a Trusted Project ..... 554
Verifying POU Syntax ..... 555
Cleaning a Trusted Project ..... 556
Running an Application Online ..... 556
Download code to Trusted controllers ..... 557
Connect with Trusted controllers ..... 558
Access diagnostic information ..... 559
Start or stop a Trusted controller ..... 560
Disconnect from Trusted controllers ..... 561
Simulate an Application ..... 561
Forcing the Values of Variables ..... 562
Monitoring the Values of Variables ..... 563
Name conventions and limitations ..... 564
File System Paths ..... 567
Online Updates ..... 567
Regular Online Updates ..... 568
Intelligent Online Updates ..... 569
Intelligent Update Manager ..... 571
Intelligent Update Options ..... 572
Manual Function Block Matching ..... 574
Intelligent Update Manager Report ..... 576
System Configuration Tool ..... 576
Managing Templates ..... 578
Threshold Templates ..... 580
LED Templates ..... 582
Force Templates ..... 586
System Templates ..... 587
Flags Templates ..... 591
Filter Templates ..... 592
Additional CLI Templates ..... 593
Speed Monitor Templates ..... 594
Shutdown Templates ..... 595
Channel Type Templates ..... 596
De-energized Short Circuit Templates ..... 597
Configuring the SYSTEM.INI File ..... 598
Configuring the Trusted TMR Processor ..... 599
Assigning I/O Modules to the Trusted System ..... 600
Configuring I/O Modules for Trusted Applications ..... 601
Configuring the 8151 Communications Interface Module ..... 602
Generating the SYSTEM.INI file ..... 603
Example SYSTEM.INI file ..... 604
Setting the Communications Port ..... 609
Serial Communications ..... 609
Ethernet Communications ..... 610
Download the SYSTEM.INI file to the TMR Processor ..... 610
Upload the SYSTEM.INI file from the TMR Processor ..... 611
Dictionary ..... 611
Access a Dictionary grid instance ..... 612
Defined Words Grid ..... 612
Variables Grid ..... 615
Variable extended attributes ..... 621
Cross Reference Browser ..... 623
Update the cross reference data ..... 625
View cross references of a variable ..... 625
Change the view of variable cross references ..... 626
View cross references of a block ..... 627
Change the view of block cross references ..... 628
List view for block cross references ..... 628
Tree view for block cross references ..... 628
View cross references of a defined word ..... 629
Change the view of defined word cross references ..... 630
List view for defined word cross references ..... 630
Tree view for defined word cross references ..... 630
Validators ..... 631
Cross Reference Checker ..... 632
TIC Dependency Checker ..... 633
TIC Difference Checker ..... 634
TIC Version Checker ..... 636
Errors and Warnings ..... 637
I/O Wiring ..... 638
Add I/O boards and complex equipment to a rack list ..... 640
Toggle the real/virtual attribute of an I/O device ..... 641
Access simple devices of complex equipment. ..... 642
Display I/O device information ..... 642
Delete an I/O device ..... 642
Wire the channels of an I/O device ..... 643
Unwire channels of an I/O device ..... 643
Lock and unlock an I/O variable ..... 644
Force the value of an I/O variable ..... 644
Create a conversion table ..... 645
Edit the points of an existing conversion table ..... 646
Delete a conversion table ..... 646
I/O wiring keyboard shortcuts ..... 647
FBD Language ..... 647
FBD Diagram Main Format ..... 647
Debug FBD Programs ..... 649
FBD Elements ..... 649
Blocks in FBD editors. ..... 650
FBD Variables ..... 650
Vertical Bars ..... 652
Labels ..... 652
Jumps ..... 653
Returns. ..... 654
Rungs ..... 655
Right Power Rails ..... 656
Left Power Rails. ..... 656
Coils ..... 657
Contacts ..... 660
Regions. ..... 664
Comments ..... 664
FBD Keyboard Shortcuts ..... 665
LD Language ..... 666
Debug LD Programs ..... 668
LD Elements ..... 669
LD Rungs. ..... 670
LD Blocks ..... 672
Coils ..... 675
Contacts ..... 681
Jumps ..... 685
Returns ..... 687
Branches ..... 688
Selecting LD Elements ..... 689
LD Keyboard Shortcuts ..... 690
ST Language ..... 692
ST Main Syntax. ..... 692
Expressions and Parentheses ..... 694
Calling Functions ..... 695
Calling Function Blocks ..... 696
Debug ST Programs ..... 696
ST Basic Elements and Statements ..... 697
Assignments ..... 697
CASE Statement ..... 698
EXIT Statement ..... 698
FOR Statement ..... 699
IF-THEN-ELSIF-ELSE-END_IF Statement ..... 700
REPEAT Statement ..... 701
RETURN Statement ..... 702
WHILE Statement ..... 703
ST Extensions ..... 704
TSTART Statement ..... 704
TSTOP Statement ..... 705
ST Operators ..... 705
FEDGE ..... 706
REDGE ..... 706
ST Keyboard Shortcuts ..... 707
ISaVIEW ..... 709
Create an ISaVIEW screen from a blank document ..... 710
Create an ISaVIEW screen from a template ..... 710
Export an ISaVIEW screen as a template ..... 711
Insert Objects ..... 711
Arc ..... 712
Arrow ..... 712
Ellipse ..... 713
Rectangle ..... 713
Rounded Rectangle ..... 714
Triangle ..... 714
Image ..... 715
Web Container ..... 715
Button ..... 716
Edit Box ..... 716
Gauge ..... 716
Slider ..... 718
Line ..... 719
Bar Meter ..... 719
Polygon. ..... 720
Edit Objects ..... 721
Select Objects ..... 722
Edit the Properties of Objects ..... 722
Cut, Copy, and Paste Objects ..... 722
Delete Objects ..... 723
Move Objects ..... 723
Resize Objects ..... 723
Group Objects ..... 724
Align Objects ..... 724
Move Objects to the Front and Back ..... 725
Define Animation Effects for Objects ..... 725
Action ..... 726
Color ..... 727
Displacement ..... 728
Rotation ..... 729
Size ..... 730
Text ..... 731
Visibility ..... 732
Preview ISaVIEW Screens ..... 732
Preview Selections ..... 732
Preview Animation Effects (Editable) ..... 733
Language Reference. ..... 734
Programs ..... 735
Functions ..... 735
Function Blocks ..... 736
Execution Rules ..... 737
Reserved Keywords ..... 738
Variables ..... 738
Directly Represented Variables ..... 739
Defined Words ..... 740
Data Types ..... 741
Boolean Data Type ..... 741
Double Integer Data Type ..... 742
Real Data Type ..... 743
Time Data Type ..... 743
Message Data Type ..... 744
Operators ..... 745
1 GAIN ..... 746
Addition ..... 747
ANA ..... 748
AND ..... 748
BOO ..... 749
CAT ..... 750
Division ..... 751
Equal ..... 752
Greater Than ..... 752
Greater Than or Equal ..... 753
Less Than ..... 754
Less Than or Equal ..... 755
MSG ..... 756
Multiplication ..... 756
NEG ..... 757
Not Equal ..... 758
OPERATE ..... 759
OR. ..... 759
REAL ..... 760
Subtraction. ..... 760
SYSTEM ..... 761
TMR ..... 764
XOR ..... 764
Functions ..... 765
ABS ..... 766
ACOS ..... 767
AND_MASK ..... 767
ARCREATE ..... 768
ARREAD ..... 769
ARWRITE ..... 770
ASCII ..... 771
ASIN ..... 771
ATAN ..... 772
CHAR ..... 773
COS ..... 773
DAY_TIME ..... 774
DELETE ..... 775
EXPT ..... 776
F_CLOSE ..... 776
F_EOF ..... 777
F_ROPEN ..... 778
F_WOPEN ..... 778
FA_READ ..... 779
FA_WRITE ..... 780
FIND ..... 781
FM READ ..... 782
FM_WRITE ..... 783
INSERT ..... 784
LEFT ..... 785
LIMIT ..... 786
LOG ..... 787
MAX ..... 787
MID ..... 788
MIN ..... 789
MLEN ..... 790
MOD ..... 791
MUX4 ..... 791
MUX8 ..... 792
NOT_MASK ..... 794
ODD ..... 794
OR_MASK ..... 795
POW ..... 796
RAND ..... 796
REPLACE ..... 797
RIGHT ..... 798
ROL (Rotation Left) ..... 799
ROR (Rotation Right) ..... 800
SEL ..... 801
SHL (Shift Left Arithmetic or Shift Left Signed) ..... 801
SHR (Shift Right Arithmetic or Shift Right Signed) ..... 802
SIN ..... 803
SQRT ..... 804
TAN ..... 804
TRUNC ..... 805
XOR_MASK ..... 806
Function Blocks ..... 806
AVERAGE ..... 807
BLINK ..... 808
CMP ..... 808
CTD ..... 809
CTU ..... 810
CTUD ..... 811
DERIVATE ..... 812
F_TRIG ..... 813
HYSTER ..... 814
INTEGRAL ..... 814
LIM_ALRM ..... 815
R_TRIG ..... 816
RS. ..... 817
SEMA ..... 818
SIG_GEN ..... 818
SR. ..... 819
STACKINT ..... 820
TOF ..... 821
TON ..... 822
TP. ..... 823
Global Library Functions and Function Blocks ..... 823
AVG ..... 825
BCD Translation (BCD) ..... 826
CLAMP ..... 827
DEV_AL ..... 828
Error Statistics (ERRSTAT) ..... 829
FL2REGH ..... 831
FL2REGL. ..... 831
High Value Select (HVS) ..... 832
IPID ..... 833
Low Value Select (LVS) ..... 836
MANTRK ..... 837
MASSFLOW ..... 838
Median Value Select (MVS) ..... 839
PC DRUM ..... 840
PID_II ..... 840
R_SEL ..... 843
RATE ..... 844
REG2FL ..... 845
SCALE ..... 846
SLEW ..... 847
Square Root Extraction (SQRTX) ..... 848
T AVG ..... 849
Activity Logger ..... 852
Log Entries ..... 852
Legacy Log Entries ..... 855
Search the Activity Logger ..... 857
Copy and Paste Activity Logger Entries ..... 858
Diary ..... 858
License Trusted applications ..... 859
Obtain an authorized Full license ..... 859
Remove an authorized license ..... 860
Trusted Product Descriptions and Manuals ..... 860
Legal Notices ..... 862

# Introducing the AADvance-Trusted SIS Workstation software 

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software is a complete suite for building process control projects for use with Trusted ${ }^{\circledR}$ Series 8000 controllers or AADvance Series 9000 controllers. The environment includes development tools and other technologies simplifying design, development, and deployment of applications. Develop projects on a Windows ${ }^{\circledR}$ development platform, using the Integrated Development Environment (IDE) and language editors.

The AADvance-Trusted SIS Workstation software graphically represents and organizes controllers and networks. For AADvance applications, the development process consists of creating projects made up of one or more controllers, representing individual targets. For Trusted applications, the development process consists of creating projects made up of one device, representing the Trusted controller.

The development environment consists of the following windows and tools:

- Application View: Displays a graphical view of projects in a logically organized tree view.
- Block Library: Displays a graphical view of all operators, functions, and function blocks available for the program organization units (POUs) of a project.
- Block Selector: Enables selecting operators, functions, and function blocks for use in block elements defined in programs.
- Cross Reference Browser: Provides an overview of the variables, programs, functions, function blocks, and defined words existing in a project including information like names, various properties, location of usage, and comments.
- Description Window: Enables adding descriptions to projects, POUs, and AADvance controllers.
- Dictionary: Enables managing variables and defined words. Create, edit, or delete variables and defined words from the dictionary instances.
- Generate Documentation: Enables generating documentation for projects, controllers or devices, POUs, variables, and library elements.
- Error List: Displays errors, warnings, and messages produced when editing POUs and performing build operations.
- Find and Replace Utility: Enables finding and replacing strings or expressions in files.
- Language Editors: Provide the environment for developing the contents of POUs using IEC 61131-3 programming languages.
- Locked Variables viewer: Enables unlocking locked variables while an application is connected or simulating.
- Output window: Displays generated status messages.
- Parameters display: Provides a graphic representation of the parameters for a POU selected in the Block Selector. When selecting a POU from the block list, displays the local, input, and output parameters.
- Parameters view: Enables managing parameter and local variables for user-defined (POUs). This view provides a graphic display of the parameters and local variables.
- Properties window: Enables viewing and editing the properties of items selected within language containers, the Application View, and other various views. Also use the Properties window to view and edit file and project properties.
- Toolbox: Displays the programming language elements available to add to programs. When shifting focus to a different program, the selection in the Toolbox shifts to the tab for the corresponding programming language.
- Spy List: Enables monitoring changes in values, for variables, arrays (if supported by the application), structures (if supported by the application), and instances of function blocks. Add items to spy lists directly, from the Dictionary, or from the language editors.
- Variable Selector: Enables selecting variables for use in POUs or mapping variables for the various communication protocols, like SOE and Modbus.

The AADvance-Trusted SIS Workstation software also contains windows and tools that are specific to an AADvance or Trusted applications; find additional information on the available windows and tools in those sections.

## System Security

## See also

Application View on page 24
Language Editor on page 26
Working in the Development Environment on page 56
Options for the Development Environment on page 70
Description Window on page 101
The AADvance or Trusted ${ }^{\circledR}$ system uses Ethernet networks, potentially exposing the system to accidental or malicious infection or attack. To protect the system, you must configure the following:

- Network Firewall
- Windows Firewall

To ensure the system is secure, perform the following general security steps:

- The AADvance or Trusted system must not be connected to a network with open unsecured access to the internet.
- A router firewall must be active on the workstation, preventing access to the unused Ethernet ports on each communication interface.
- An up-to-date anti-virus software must be installed on the workstation.

Important: You may need to temporarily disable firewalls when using the AADvance Discover tool

- The workstation must be password protected and locked when unattended.
- The AADvance application must be password protected if the program enable key is not used on the system.
- The Trusted application must be password protected if the maintenance enable key switch is not in the Run position.
Tip: Since serial networks are closed, local, and have limited protocol functionality, they
are immune to external attacks (apart from deliberate local sabotage).


## See also

## Network Firewall on page 23

Windows Firewall on page 24
Network Firewall
When the network used by the AADvance or Trusted ${ }^{\circledR}$ system is connected to another network, the connections must pass through the firewall. These communication ports are supported by the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software:

| Protocol | Port Number | Availability | Purpose |
| :--- | :--- | :--- | :--- |
| TCP | 502 | When configured | Modbus® slave |
| TCP | 1132 | Always available | AADvance, application downloads, debug, SOE, etc. |
| TCP | 6000 | Always available | Trusted, application downloads, debug, etc. |
| TCP | $10001-10006$ | When configured (and the <br> application is stopped) | Transparent Communications Interface (serial tunneling) |
| TCP | 44818 | Always available | CIPr" produce and consume |
| TCP | 55555 | Always available | Diagnostic interface |
| UDP | 123 | When configured | SNTP |
| UDP | 1123,1124 | Always available | SNCP bindings |
| UDP | 2010 | Always available | Discovery and configuration protocol (DCP, Rockwell Automation ${ }^{\circ}$ ) |
| UDP | 2222 | When configured | CIP produce and consume I/Os |
| UDP | 5000 | When at least one P2P <br> subnet is active on a <br> Controller | Peer-to-Peer |
| UDP | 44818 | Always available | CIP produce and consume |

When the Always available ports are not configured or unused, they are open to unauthorized access. Perform the following actions to protect network ports:

- If the network used by the system is connected to another network, the connection must pass through the firewall to protect the system from potential threats from other networks.
- Configure the firewall to block all communication ports. If required, enable a specific port to a device needing to communicate with other devices on other networks.
- The SNCP port must only be allowed to pass through the firewall if the Windows ${ }^{\circledR}$ PCs running the AADvance-Trusted SIS Workstation software are on a separate network.
- The variable bindings port must only be allowed to pass through the firewall if the AADvance controller is communicating with another AADvance controller on a separate network.
- The other communication ports (such as Modbus and SNTP), must only be allowed to pass through the firewall if the AADvance controller, Trusted controller, or Windows PC communicates with other devices on other networks.


## See also

Windows Firewall on page 24

## Windows Firewall

Activate and configure the Microsoft ${ }^{\circledR}$ Windows firewall on a workstation to allow connections through the ports used to communicate with the AADvance or Trusted ${ }^{\circledR}$ controllers.

Configure other ports for the following scenarios:

- Allow the workstations to access the Windows Updates, time synchronization, and anti-virus updates.
- The Windows workstation may run additional software (such as HMI or data logging), therefore requiring the ports to pass through the firewall.
Tip: Refer to the manufacturer's manual for instructions about firewall settings.


## See also

Network Firewall on page 23

## Application View

## To access the Application View

## From the View menu, click Application View.

Use the Application View to display a graphical view of projects in a logically organized tree view. For Trusted ${ }^{\circledR}$ applications, projects include one device, programs, functions, function blocks, global variables, defined words, and libraries. For AADvance applications, projects include controllers, programs, functions, function blocks, controller variables, global data types, and libraries. The physical locations of project files can differ from the representation in the tree view structure.

| AADvance Tree View | Trusted Tree View |
| :---: | :---: |
|  | Application View <br> TRUSTEDproj <br> Device1 <br> Programs <br> ProgFBD <br> ProgLD <br> Local Variables <br> Local Defined Words <br> ProgLD－Diary <br> ProgST <br> 4 描 Functions $\square$ FCT1 <br> 4 退 Function Blocks FB1 <br> Global Variables Defined Words <br> Libraries Global <br> 青 Functions Function Blocks |

From the Application View，perform many tasks for AADvance applications：
－Adding controllers，programs，functions，and function blocks
－Adding new libraries，published libraries，and existing libraries
－Setting properties of projects，controllers，programs，functions，and function blocks
－Setting passwords for projects，controllers，targets，and programs
－Building projects，controllers，programs，functions，and function blocks
－Verifying syntax for programs，functions，and function blocks
－Cleaning projects and controllers
－Downloading and updating projects and controllers
－Connecting controllers
－Copying，pasting，deleting，and renaming controllers
－Cutting，copying，pasting，deleting，and renaming programs，functions， and function blocks
－Opening，defining，importing，and exporting variables
－Opening and defining global data types
－Importing target definitions
－Accessing version control operations for projects，controllers， programs，functions，and function blocks
－Creating variable groups
Important：Avoid using names with more than 32 characters in the Application View．

From the Application View，perform many tasks for Trusted applications：
－Adding programs，functions，and function blocks

- Importing libraries
- Setting properties of projects, devices, programs, functions, and function blocks
- Setting passwords for projects
- Building projects, devices, programs, functions, and function blocks
- Verifying syntax for programs, functions, and function blocks
- Cleaning projects, devices, programs, functions, and function blocks
- Downloading and updating projects and devices
- Connecting devices
- Renaming devices
- Cutting, copying, pasting, deleting, and renaming programs, functions and function blocks
- Opening, defining, importing, and exporting variables
- Opening and defining defined words
- Opening and modifying the Diary text file for programs, functions, and function blocks

| Task | Procedure |
| :--- | :--- |
| Drag and drop elements to change <br> position in tree structure | In the Application View, drag the required element to the new position within the same section. |
| Refresh the content displayed in the <br> Application view | Select the Application View, then from the View menu, click Refresh. |

> Tip: $\quad$ For Sequential Function Chart (SFC) program organization units (POUs), move an SFC main program, to a different position in the tree structure. When moving an SFC POU with children, children follow their parent. Also move a child SFC POU to a different main SFC POU or to a different hierarchy level. You can only drag child SFC POUs to hierarchical levels already containing an SFC child.

## See also

Working in the Development Environment on page 56
Options for the Development Environment on page 70

## Language Editor

Use the language editor is the environment to develop the contents of program organization units (POUs). Develop these POUs using language containers. Language editors hold elements of a given IEC 61131-3 programming language. A POU can only have one language editor. Description windows hold non-semantic information. When building projects, the compiler excludes information from description windows and HMI editors.

From the language editor, edit multiple POUs simultaneously. Individual POUs open in separate workspaces each with a tab indicating the POU name. The tabs enable moving from one POU to another.

When working in the language editor, choose to expand the workspace to a full-screen view.

Edit the contents of POUs in the workspace.

| Task | Procedure |
| :--- | :--- |
| Expand the workspace to full-screen <br> view | $\bullet$ From the View menu, click Full Screen. |


| Task | Procedure |
| :---: | :---: |
| Select an element | Selected elements are displayed with colored handles. When selecting multiple elements, the handles of the first element are green and subsequent elements are turquoise. <br> When aligning multiple elements, the reference point depends on the programming language. <br> 1. To select one element, click the element in the workspace. <br> 2. To select multiple elements, do one of these: <br> - Starting from an empty workspace, drag the pointer over the elements. <br> - While pressing Shift, use the pointer to select elements individually. |
| Insert an element | - From the Toolbox, drag the element into the language editor. |
| Insert an identifier | When inserting identifiers, insert a constant or variable automatically via the Variable Selector. <br> 1. From the Toolbox, drag the variable element into the workspace. <br> 2. From the Variable Selector, perform one of these, and then click OK. <br> - In the Name field, type a literal value. <br> When inserting literal values that begin with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc' <br> - Select the required variable from the lists of variables. |
| Insert a block | Insert blocks into language editors of graphical programs from the Toolbox. Following insertion, specify the type of block, such as operator, function, or function block from the Block Selector. <br> For programs, the available items are operators (OPE), standard functions (SFU), standard function blocks (SFB), user IEC 61131-3 Functions (IFU), user IEC 61131-3 Function Blocks (IFB) and all Functions (CFU) and Function Blocks (CFB) supported by the current target. <br> 1. From the Toolbox, drag the block element into the workspace. <br> 2. In the Block Selector, in the list of blocks, locate the required block. <br> 3. To limit the blocks displayed, sort the block list and filter the block list. <br> 4. From the list of available blocks, select a block, and then click OK. |
| Move an element | 1. In the workspace, select one or more elements. <br> 2. Drag the elements to another position. |
| Resize an element | Change the dimensions of specific individual elements. <br> 1. In the language editor, select an element. <br> 2. Place the cursor over the element handles, then click and drag to the required size. |
| Delete an element | - In the language editor, select one or more elements, then press Delete. |
|  | Graphical POUs display an error symbol below elements with errors in the programming logic. Pausing on this symbol displays a description of the error. |

## See also

Applying best practices on page 27
Reset Container and Shape Settings in FBD POUs on page 34
Reset Visual Settings in LD POUs on page 35

## Applying best practices

Use these best practices for keeping program organization unit (POU) code simple and clear.




| Best Practice | Example |
| :--- | :--- |
| Avoid assigning multiple levels of |  |
| variables in a network. |  |
| Flatten variables on a single level to <br> avoid CVT mismatches due to the <br> execution order. |  |



| Best Practice |
| :--- |
| Avoid using power rails with negated <br> connections. <br> Use variables providing the required <br> input value to avoid CVT errors. Using a power rail with negated connections: |



## See also

Language Editor on page 26
Compiler Verification on page 300
Parameters View on page 46
Block Selector on page 44

Reset Container and Shape Settings in FBD POUs

Reset the container or shape settings to modify the visual properties for the language containers or elements in Function Block Diagram (FBD) Program Organization Units (POUs).

After resetting the Container Settings in the Properties window, subsequent elements added to the language editor use the new values.

## Tip: $\quad$ Changes to the Display Grid property made from the Properties window immediately appear in the language editor.

## To reset container and shape settings in FBD POUs

- In an open FBD Program Organization Unit (POU), right-click in the language editor, and then select one of these commands:
- Reset Container Settings - Resets the values defined for the Container Settings in the Properties window to the values set in Tools > Options > IEC Languages > Function Block Diagram (FBD). Elements in the open POU are not modified.
- Reset Shapes Settings - Resets the visual properties of all elements in the open POU to the properties defined for the Container Settings in the Properties window.
- Reset All Settings - Resets the values defined for the Container Settings in the Properties window and the visual properties of all elements in the open POU to the values set in Tools > Options > IEC Languages > Function Block Diagram (FBD)


## To reset shape settings for selected elements in FBD POUs

- In an open FBD POU, select the elements to reset, right-click and then select Reset Shape Settings.

The visual properties of the selected elements reset to the values defined for the Container Settings in the Properties window.

## See also

## Language Editor on page 26

Function Block Diagram Settings on page 84
Properties Window on page 37

# Reset Visual Settings in LD POUs 

Reset the visual settings to modify the visual properties for the language containers and elements in Ladder Diagram (LD) Program Organization Units (POUs).

After resetting the LD Container in the Properties window, subsequent elements added to the language editor use the new values.

## To reset visual settings in LD POUs

- In an open LD POU, right-click in the rung, and then select one of these commands:
- Reset All Visual Settings - Resets the visual properties of all elements in the open POU to the properties defined for the LD Container in the Properties window.
- Reset Container Visual Settings - Resets the values defined for the LD Container in the Properties window to the values set in Tools > Options > IEC Languages > Ladder Diagram (LD). Elements in the open POU are not modified.

[^0]
## To reset visual settings for selected elements in LD POUs

- In an open LD POU, select the elements to reset, right-click and then select Reset Element Visual Settings.

The visual properties of the selected elements reset to the values defined for the LD Container in the Properties window.

## See also

Language Editor on page 26
Ladder Diagram Settings on page 87
Properties Window on page 37

## Block Library

## To access the block library

From the View menu, click Block Library.
The Block Library provides a graphical view of all operators, functions, and function blocks available for the program organization units (POUs) of a project. When developing POUs:

- Drag and drop blocks from the library to the language container
- Sort blocks that display in the library according to alphabetical order, categories, scope, or limit a search based on names. Blocks display in either tile or list views.


Types of blocks that may be available from the block library are:

- Standard operators
- Standard functions
- Standard function blocks
- User-defined IEC 61131 functions
- User-defined IEC 61131 function blocks
- User-defined functions and function blocks from a library

Blocks are sorted by scope:

- Standard blocks
- Library blocks (a scope for each library dependency)
- Target-specific C blocks

| Task | Procedure |
| :--- | :--- |
| Insert a block in a POU | In the Block Library, locate the required block, then click and hold the mouse on the block while dragging to the destination in <br> the POU container. <br> Tip: If deleting a library function block instance from a POU by mistake, do not use the Undo command. Insert the <br> library function block again. |
| Sort blocks in the library | Right-click in the Block Library, and then for each sort group click one of these: <br> $\bullet$ Category or Scope <br> $\bullet$ List View or Tree View <br> $\bullet$ Expander display mode or Tabs display mode |
| Search blocks in the library | Perform searches for blocks by entering any part a block name. Type text in the library search field to display only the blocks <br> containing these characters. <br> $\bullet$ In the search field in the Block Library, type the required text. |

## See also

Block Selector on page 44
Spy Lists on page 53
Blocks on page 306

## To access the Properties window

From the View menu, click Properties Window. The F4 or Alt+Enter keyboard shortcuts are also available.

Use the Properties window to view and edit the properties of items selected within language containers, the Application View, and other various views. Also use the Properties window to view and edit file and project properties. View the common properties for multiple objects and elements. When selecting multiple objects, the Properties window displays only the properties that are common to all the objects and elements.

In the Properties window, properties are organized alphabetically or into categories displayed alphabetically. Expand the categories to view the property information including property names and values. Properties that display in gray are read-only.

Edit property values using the plain text fields and drop-down combo boxes provided. Where required, links to custom editors or dialog boxes displayed in the property value fields.

The Properties window toolbar contains:

| Icon | Description |
| :---: | :---: |
| * | Displays the name of the item or group of items selected. |
| ${ }_{\square}^{\text {® }}$ | Displays the property names and values organized into categories. |
| 会 $\downarrow$ | Displays the properties sorted in alphabetical order |
| 廻 | Not applicable |

## See also

## Application View on page 24

Parameters View on page 46

## Locked Variables Viewer

## To access the list of locked variables

From the Online or Simulation menu, point to Windows, and then click Locked Variables. The Locked Variables window is only accessible while an application is connected or simulating.

Use the Locked Variables window to enable unlocking locked variables while an application is connected or simulating. This window lists all locked variables and their source throughout an application. When viewing locked variables, the variable identification indicates its source like the project name, and ends with the variable name.


Tip: $\quad$ Only $1 / 0$ variables can be locked.

From the Locked Variables window, perform these tasks:

| Task | Procedure |
| :---: | :---: |
| Find variables from the list of locked variables | Perform searches based on any part of the variable identification displayed in the Locked Variables window such as the complete or partial variable, controller, or project name. <br> In the search field, enter text contained in the identification of the required variable, then do one of these: <br> - To find the first instance of the variable, click . <br> - To find the next instance of the variable, click $\stackrel{\text { 皿 }}{ }$. |
| Unlock variables from the list of locked variables | - In the list of locked variables, select the variables to unlock or click $\qquad$ to select all the variables in the list, then click to unlock them. |

## Toolbox

## To display the toolbox

- From the View menu, click Toolbox (or press Ctrl+Alt+X).

Expand the multiple segments or tabs of the Toolbox. Also scroll through the entire tree within the Toolbox. To expand Toolbox tabs, click the blank rightpointing arrow next to the tab name. To collapse expanded tabs, click the darkened down-pointing arrow next to the tab name.

The Toolbox displays icons for elements to add to programs. When shifting focus to a different program, the current selection in the Toolbox shifts to the tab for the corresponding programming language. Ways to manipulate the Toolbox are:

| Task | Procedure |
| :--- | :--- |
| Display the Toolbox | • From the View menu, click Toolbox (or press Ctrl+Alt+X). |
| Search elements in the Toolbox | • In the search field of the Toolbox, enter text contained in the element to locate. <br> The Toolbox displays only the elements containing the specified text. |
| Restore all default tabs and <br> elements to the Toolbox | • Right-click the Toolbox, and then click Reset Toolbox. |

Customize the Toolbox by rearranging elements within a tab or adding custom tabs and elements. Ways to manipulate Toolbox tabs are:

| Task | Procedure |
| :--- | :--- |
| Expand a Toolbox tab | • Click the blank right-pointing arrow next to the name of the collapsed Toolbox tab. |
| Collapse a Toolbox tab | • Click the darkened down-pointing arrow next to the name of the expanded Toolbox tab. |
| Move a Toolbox tab | Perform on of these to move Toolbox tabs within the Toolbox: <br> - Right-click on the name of the tab, and then click Move Down or Move Up. <br> - Drag the tab to the required position in the Toolbox. |
| Rename a Toolbox tab | 1. From the Toolbox, right-click the required tab, and then click Rename Tab. <br> 2. In the space provided, type a name for the tab, then press Enter. |
| Add a custom Toolbox tab | Added tabs appear at the bottom of the Toolbox. Reposition and add elements to tabs. <br> 1. From the Toolbox, right-click any tab, and then click Add Tab. <br> 2. On the blank tab, in the space provided, type a name for the tab, then press Enter. |
| Remove a custom Toolbox tab | Move the elements to retain to other tabs before deleting the custom tabs. <br> 1. From the Toolbox, right-click the tab to remove, then click Delete Tab. <br> When elements remain on the tab, a message box informs you that those elements will be deleted. |
| 2isplay all available Toolbox tabs | • Right-click the Toolbox, and then click Show All. |

Insert elements in language containers that display in the integrated development environment (IDE). This action adds the fundamental code to create an instance of the Toolbox element in the active program file. Ways to manipulate toolbox elements are:

| Task | Procedure |
| :---: | :---: |
| Insert a workspace element | From the Toolbox, drag the required element into the workspace. |
| Rename a Toolbox element | 1. In the Toolbox, right-click the required element, and then click Rename Item. <br> 2. In the space provided, type a name for the element, and then press Enter. |
| Sort the elements alphabetically | - In the Toolbox, right-click the required tab, then click Sort Items Alphabetically. |
| Hide element names | - In the Toolbox, right-click the required tab, then click List View. |
| Rearrange elements | Reposition elements displayed on Toolbox tabs. <br> - In the Toolbox, select the required element and perform one of the following: <br> - Right-click the element, and then click Move Down or Move Up. <br> - Drag the element to the required position. |
| Move an element between tabs | - In the Toolbox, select the required element and perform one of the following: <br> - Drag the required element onto another tab. <br> - Right-click the element and click Cut or Copy, then right-click the required tab and click Paste. |
| Remove an element | Certain elements like the pointer element cannot be removed. <br> - In the Toolbox, right-click the required element, and then click Delete. |

## See also

## Variable Selector on page 40

Block Selector on page 44

## Variable Selector

## To access the Variable Selector

For graphical POUs (FBD or LD), drag an element requiring the selection of a variable, like a variable, coil, or contact. For textual POUs (ST), right-click in the language container, and then click Variable Selector (Ctrl+I). When mapping variables for communication protocols, click Add Variable or

Tip: The Variable Selector can only be accessed while in design mode.
Use the Variable Selector to select variables for use in program organization units (POUs) or mapping variables for the various communication protocols, like SOE and Modbus. The available scopes depending on the access of the Variable Selector. The Variable Selector displays variables in alphanumerical order. Sort or filter the listed variables.

| Important: | Read this section before using the Variable Selector. The main purpose of <br> the selector is to select variables for use in POUs or mapping variables for <br> the communication protocols. The preferred method to create variables is <br> from dictionary instances. |
| :--- | :--- |

When working with the Variable Selector, perform many tasks:


- Grid filtering fields, limits the variables displayed in the variables grid by typing alphanumeric characters in the Name field.
- Scope tabs, indicates the scope for the variables to display in the variables grid. For AADvance POUs, the available scopes are local variables, controller variables (for example, global), system variables, directly represented variables, and defined words. For Trusted ${ }^{\circledR}$ POUs, the available scopes are local variables, global variables, directly represented variables, and defined words. When mapping variables for communication protocols, the available scopes are either local and global, or only global.
- Grid column headers, sorts the contents of the grid in ascending or descending alphanumerical order according to the individual properties.
- Column filtering cells, filters the contents of the grid by typing specific alphanumerical characters contained in the variables to display or clicking the down arrow to display the available grouping options from a drop-down list. Also perform filtering other than the default "starts with" operation by clicking the rightmost icon in the filtering cell.
- Variables grid, contains the variables defined for the selected scope.
- Available definition row, enables the definition of the properties for a new variable.

In the Variable Selector, navigate using keyboard and mouse controls.

| Keyboard/Mouse Control | Description |
| :--- | :--- |
| Arrow keys | Move up, down, left, and right among the cells of the list of variables. <br> Also move left and right across the tabs. |
| Tab key | Move from left to right between the fields, tab, and list of variables. <br> Within the list, move left to right between cells of a row. After exiting <br> the list of variables, move across the command buttons and back to <br> the fields. |
| Esc key | Close the Variable Selector without selecting a variable. |
| Enter key | Close the Variable Selector and display the selected variable in the <br> workspace. |


| Task | Procedure |
| :--- | :--- |
| Select a variable | 1. Access the Variable Selector. <br> 2. To sort the variables listed in the variables grid, click the column heading of the property for which to toggle between <br> ascending and descending alphanumeric order. <br> 3. To clear the filter to see all variables, and not limit the display to those filtered by the variable's data type, hover over a <br> Data Type filter cell and select the Clear Filter icon. <br> 4. To reduce the number of variables listed in the variables grid, in the grid filtering cell directly below the column header of <br> the variable property from which to filter, do one of the following: |
|  | - Type specific characters (alphanumerical) contained in the variables to display. When typing characters, also perform <br> filtering other than the default starts with operation by clicking the icon at the rightmost in the grid filtering cell. <br> Available filtering functions include ends with, equal to, like, match, and other Boolean operations. <br> • Click the down arrow to display the available grouping options from a drop-down list. |
|  | 5. Locate the required variable by scrolling down the list of variables displayed in the grid. |
| 6. Select the variable by clicking anywhere in the row, and then click OK. |  |

## See also

Create Variables from the Variable Selector on page 42
Edit Variables from the Variable Selector on page 43
Delete Variables from the Variable Selector on page 44
Block Selector on page 44
Toolbox on page 39

## Creating Variables from the Variable Selector

When creating variables from the Variable Selector, define the variables in the variables grid to specify values for all variable properties. In the grid, create a variable by providing the required information in the cells of the available definition row (with an asterisk). Quick Declaration enables creating multiple variables with the same prefix, suffix (optional), numbering range, data type, direction, and string length (string or message data types only).

| Important: | The preferred method to create variables is from dictionary <br> instances to avoid potential mistakes like inserting a variable in an <br> unintended scope. |
| :--- | :--- |

## To create a variable from the Variable Selector

1. Access the Variable Selector.
2. In the variables grid, locate the available definition row (with the asterisk), define the required properties, and then press Enter.
For graphical POUs, the variable is defined for the element. For textual POUs, the variable is placed in the editor.
3. (optional) Create multiple variables with the same prefix, suffix, data type, and continuous numbering definition:
a. In the grid area, locate the available definition row (with the asterisk), right-click in the Name cell, and then click Quick Declaration.
b. In the Quick Declaration dialog box, specify the numbering range, a prefix, a suffix (optional), the data type, the direction, and the string length (if applicable), and then click OK.

The numbering range indicates the number of variables to create. For example, a range of 0 to 9 creates ten variables. When using the quick declaration method for creating variables, the consecutive variables shall have the same prefix followed by the starting range number, and optionally finishing with the same suffix.
4. (optional) To insert literal values, in the Name field, type the literal value, and then click OK.
Tip: When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'

## See also

Edit Variables from the Variable Selector on page 43
Delete Variables from the Variable Selector on page 44
Variable Selector on page 40

## Editing Variables from the Variable Selector

To edit a variable, create variables from dictionary instances. This prevents potential mistakes such as inserting a variable in an unintended scope.

## To edit a variable

1. Access the Variable Selector using the required method for the element:

- For graphical program organization units (POUs) Function Block Diagram (FBD) or Ladder Diagram (LD), drag an element requiring a variable or double-click an element in the language container. The Variable Selector opens.
Tip: To open the Variable Selector, select Automatically invoke the Variable/Block Selector.
- For textual POUs (Structured Text (ST)), right-click in the language container and select Variable Selector.
- When mapping variables for communication protocols, select Add Variables.

In the variables grid, locate the variable to edit, changed the required properties for the variable, and then press Enter.

## See also

Create Variables from the Variable Selector on page 42
Delete Variables from the Variable Selector on page 44
Variable Selector on page 40

## Delete Variables from the Variable Selector

Deleting variables from the Variable Selector also removes these variables from the dictionary instance for the program, controller, or device.

| Important: | The preferred method to create variables is from dictionary <br> instances to avoid potential mistakes like inserting a variable in an <br> unintended scope. |
| :--- | :--- |

## To delete a variable

- Access the Variable Selector by doing one of the following:
- For graphical POUs (FBD or LD), drag an element requiring the selection of a variable, like a variable, coil, or contact. The Variable Selector displays (when the Automatically invoke the Variable/Block Selector option is set to True). Also double-click an element in the language container to invoke the Variable Selector.
- For textual POUs (ST), right-click in the language container, and then click Variable Selector (Ctrl+1).
- When mapping variables for communication protocols, click Add

Variable or
In the variables grid, right-click the variable to delete, and then click Delete.

## See also

Create Variables from the Variable Selector on page 42
Edit Variables from the Variable Selector on page 43
Variable Selector on page 40

## Block Selector

## To access the Block Selector

From the Toolbox, drag the block element into the language container.
Use the Block Selector to select operators, functions, and function blocks for use in block elements defined in programs. For Function Block Diagram (FBD) and Ladder Diagram (LD) programs, enter blocks and declared instances.

The Block Selector lists the available operators, functions, and function blocks for IEC 61131-3 programs. The available items are operators (OPE), standard functions (SFU), standard function blocks (SFB), user IEC 61131-3 Functions
(IFU), user IEC 61131-3 Function Blocks (IFB) and all C Functions (CFU) and C Function Blocks (CFB) supported by the current target.

| - A Block Selector (Prog 3) |  |  |  | $\mathrm{x}^{2}$ |
| :---: | :---: | :---: | :---: | :---: |
| AVERAGE (Project29) |  |  |  | , |
| Search |  |  |  | Show Parameters |
|  |  |  |  |  |
| ANY_TO_ULINT |  | Data Conversion | Conversion of any variable to an unsigned long integer variable. | Standard |
| ANY_TO_USINT | OPF | Data Conversion | Conversion of any variable to an unsigned short integer variable. | Standard |
| ANY_TO_WORD | cops | Data Conversion | Conversion of any variable to a word variable. | Standard |
| ASCII | sfo | String manipulation | Character -> ASCII code | Standard |
| ASIN | sfo | Arithmetic | Arc sine | Standard |
| ATAN | sfou | Arithmetic | Arc tangent | Standard |
| AVERAGE | sfe, | Data Manipulation | Running average over N samples | Standard |
| BLINK | sfes. | Signal generation | Blinking boolean signal | Standard |
| CHAR | sfo | String manipulation | ASCII code -> Character | Standard |
| CMP | sfe: | Comparators | Full comparison function block | Standard |
| $\cos$ | sfo | Arithmetic | Cosine | Standard |
| CTD | Sfes: | Counter | Down counter | Standard |
| ctu | sfe: | Counter | Up counter | Standard |
| Instance: | AVERAGE_1 |  |  | - (New) |
|  |  |  | OK | Cancel |

For the block list, the properties are:

| Property | Description |
| :--- | :--- |
| Name | Name of the function, function block, or operator. |
| Type | Type of function, function block, or operator. Possible types are "C" <br> function (CFU), "C" function block (CFB), IEC 61131-3 function (IFU), IEC <br> 61131-3 function block (IFB), operator (OPE), standard function block (SFB), <br> and standard function (SFU). |
| Category | Category of function, function block, or operator. Possible categories vary <br> depending on the target definition. |
| Comment | Comment for the function, function block, or operator. Free-format text. |
| Scope | Indicates where the POU is defined. |

When selecting operators like addition, multiplication, and AND, specify the number of inputs.

Use the Block Selector to refine the list of available blocks by sorting the block list and limiting searches. Also choose to display the parameters while viewing the blocks.

In the Block Selector, navigate with keyboard and mouse controls.

| Arrow keys | Move up, down, left, and right within the cells of the blocks list. |
| :--- | :--- |
| Tab key | Move left and right within the cells of each row in the blocks list. After exiting the blocks list, move from left to right between the <br> fields, option, command buttons and back to the blocks list. |
| Esc key | Move from the blocks list to the fields. |
| Space bar | Open the parameters display when the Show Parameters option is selected. |
| Enter key | Close the Block Selector and display the selected block in the workspace. |

From the Block Selector, access help for the displayed operators, functions, and function blocks.

| Task | Procedure |
| :--- | :--- |
| Create a declared instance of a <br> function block | Declared instances are function blocks with assigned instances. For graphical and non-graphical programs, declare |
| instances in the Block Selector. These instances are considered as variables. |  |
|  | 1. From the list of available blocks, select the function block type. <br> 2. In the Instance field, type a name for the instance, and then click OK. |


| Task | Procedure |
| :--- | :--- |
| Sort the block list | Sort the columns of the block list by setting these in ascending or descending order. <br> $\bullet$ Click the required column header to toggle the sort order between ascending and descending. |
| Limit searches | Type text in the Search field to list only the blocks containing these characters. <br> $\bullet$ In the Search field of the Block Selector, type the required text. |

## See also

## Parameters Display on page 46

Variable Selector on page 40
Toolbox on page 39

## Parameters Display

The parameters display graphically presents the parameters for a program organization unit (POU) selected in the Block Selector. When selecting a POU from the block list, the parameters display automatically shows the local, input, and output parameters. Expand all or collapse all parameters for POUs.


## To expand or collapse all input and output parameters

1. In the block list, select the required block for which to display the existing parameters.
2. To expand all parameters, right-click in the parameters display, then click Expand All.

## To collapse all parameters, right-click in the parameters display, then click Collapse All.

Block Selector on page 44

## Parameters View

## To access the parameters view for a user-defined function or function block

From the Application View, create a user-defined function or function block in the respective section. Right-click the function or function block, and then click Parameters.

Use the Parameters view to manage parameter and local variables for userdefined functions and user-defined function blocks. This view provides a graphic display of the parameters and local variables.


In the Parameters view, define the properties of parameters and local variables:

| Property | Description |
| :--- | :--- |
| Name | Name of the parameter |
| Alias | When supported by the application, the short name used in the graphical language <br> editors for display only. Limited to four characters. |
| Data Type | Data type of the parameter |
| Dimension | When supported by the application, the dimension of the function block. The dimension <br> is defined as a positive double integer (DINT) value. |
| Attribute | Property of a parameter indicating its read and write access rights. Possible values are <br> Read or Read/Write. |
| Comment | Comment for the parameter. Free-format text. For Trusted <br> single-byte variables, limited to 60 <br> Initial Value |
| When supported by the application, the value held by a variable when the virtual machine <br> starts the execution of the application code. |  |

Modify the parameters for functions and function blocks. User-defined functions are limited to one output parameter with a modifiable data type.

From the Parameters view, perform these tasks:

| Task | Procedure |
| :--- | :--- |
| Create parameters and local <br> variables | Create parameters for a currently opened user-defined function or function block. Functions can only have one output. <br> 1. In the Application View, right-click the required function or function block, and then click Parameters. |
|  | 2. To add an input parameter, click New Input, then define the properties for the parameter. <br> 3. To add an output parameter to a function block, click New Output, then define the properties for the parameter. <br> 4. To add a local variable, click New Variable, then define the properties for the variable. |
| Edit parameters and local variables | 1. In the Application View, right-click the required function or function block, and then click Parameters. <br> The function or function block parameters are displayed. |
|  | 2. To edit a parameter, select the parameter, and then modify its properties. <br> 3. To edit a local variable, select the variable, and then modify its properties. |
| Delete parameters and local | 1. In the Application View, right-click the required function or function block, and then click Parameters. <br> variables |
| 2. Right-click the parameter or local variable to remove, and then click Delete. |  |
| Copy, and paste parameters and | 1. In the Parameters view for a user-defined function or function block, cut or copy the required parameter or local variable: |
| - To remove the parameter or local variable, select the item, right-click and then click Cut. |  |
| - To copy the parameter or local variable, select the item, right-click and then click Copy. |  |
| 2. To paste a copied parameter or local variable, right-click in the Parameters view and then click Paste. |  |
| Duplicated parameters or local variables are automatically placed in their respective area, such as input, output, or variable. |  |
| Tip: |  |

Tip: Update each instance of a function or function block after adding, removing, reordering, renaming, changing data type, or changing direction of parameters in all POUs using the instance. Identify these POUs with the Cross Reference Browser, then open each POU and reselect the modified instance using the Block Selector.

## Generating Documentation

While in design mode, generate documentation for projects, controllers or devices, POUs, variables, and library elements. The output format of the documentation is Microsoft ${ }^{\circledR}$ Word ${ }^{\circledR} 2010$ (*.docx). Use Generate Documentation to view the project information for a specific time, and search and edit the generated documentation.

Install Microsoft Word 2010 (or more recent) or another *.docx application to properly view, search, and edit the generated documentation.

Tip: Microsoft Word 2013 has a file size limitation that affects generated documents. Documents generated from projects with large POUs or many variables are unusable or incomplete. For more information on the Microsoft Word 2013 file size limitation, see Operating parameter limitations and specifications in Word.

The Generate Documentation dialog box is separated into three panes: Document Options, Sections, and TOC Preview. Selections in a pane affect what displays in the next pane (from left to right). Changes in the Document

Options pane affect what displays in the Sections pane and changes in the Sections pane affect the TOC Preview pane.


## Options for the Document Options pane are:

| Option | Description | Possible Values |
| :---: | :---: | :---: |
| Sections Template | Template in XML format defining the Sections to be generated in the documentation and their hierarchy. The selected Sections Template affects the items that display in the Sections and TOC Preview panes. <br> The templates are located in the following directory: \%PUBLIC\% IPublic DocumentsISIS Workstation <version>\Templates\DocumentGenerator | DefaultTemplate, NoISaview Template, or a user-defined *.xml template. The default value is DefaultTemplate. |
| Orientation | Orientation of the page | Portrait or Landscape. The default value is Landscape. |
| Page size | Size of the page | Letter, Legal, Statement, Executive, A3, A4, A5, B4 (JIS), B5 (JIS), 11x17, Envelope \#10, Envelope DL, Envelope C5, Envelope B5, Envelope Monarch, Japanese Postcard, A6, Double Japan Postcard Rotated, Executive (JIS), Oficio 8.5x13, 12x18, 8k 273x394 mm, 16k 197x273 mm, or Custom. The default value is Legal. |
| Margins | Left, right, top, and bottom margins for the page | Narrow, Normal, Moderate, or Custom. The custom margins range from 0 inches to the maximum size of the page. The default value is Normal. |
| Microsoft Word Template | Microsoft Word template in *.dotx format used to define the layout for the title page, table of contents, and tables. The templates are located in the following directory: \%PUBLIC\% IPublic DocumentsISIS Workstation <version>\Templates\DocumentGenerator | IsagrafFooter.dotx, Isagraf_NoHeader.dotx, or a user-defined *.dotx template. The default template is IsagrafFooter.dotx. |
| Diagram Scaling | Scaling for all diagrams that display in the generated documentation. | $25 \%, 50 \%, 75 \%, 100 \%, 125 \%, 150 \%, 175 \%, 200 \%, 300 \%, 400 \%$, $500 \%$, Fit to Page, or Custom. When selecting the Custom scaling, a spin box appears enabling the user to select the scaling value. The default value is $100 \%$. |
| Link Type | Type of links in the documentation. | None, Only Bookmarked, Cross Reference, or Hyperlink. The default value is None. |


| Option | Description | Possible Values |
| :--- | :--- | :--- |
| Comment Style | How comments display in the documentation. This option <br> does not affect how comments are displayed in graphical <br> POU diagrams. | $/ /$ comment, $/ /^{*}$ comment $* /$, or $(*$ comment $*)$. The default value is $~^{*}$ <br> comment $*)$. |

Selecting a Sections Template in the Document Options pane modifies the items listed in the Sections pane. The Title Page and Table of contents items are always available for selection in the Sections pane. The $\ldots$ button displays the Variable Settings dialog box and is used to specify how to sort the variables in the generated documentation.

- For AADvance projects, sort variables by Name, Comment, Alias, Data Type, Wiring, Attribute, Dimension, Initial Value, Direction, or String Size in ascending or descending order.
- For Trusted ${ }^{\circledR}$ projects, sort variables by Name, Comment, Data Type, Wiring, Initial Value, Attribute, Direction, String Size, Modbus Address, Message True, Message False, Retained, or Unit in ascending or descending order.

The items selected in the Sections pane modify the items that display in the tree view of the TOC Preview pane. Select or clear items in the TOC Preview pane. The final selection in the TOC Preview pane displays what is generated in the documentation.

The items that display in the Sections and TOC Preview panes also depend on the element selected in the Application View when using the Generate Documentation command.

- Example: if a POU is selected in a Trusted application, only associated sections (diary, local variables, and the POU diagram) display in the Generate Documentation dialog box.

When the project is selected in the Application View, all sections (project, global variables, defined words, structures, arrays, targets, etc.) display in the dialog box. Where a project, resource, device, POU, or library element is not selected, the documentation generator tries to find the associated element.

- Example: if Local Variables of a POU are selected, the POU is considered the associated element and therefore displays in the Sections and TOC Preview panes. If the document generator is unable to find an associated element, the Generate Documentation command does not appear in the File menu.

The document generator retains the selections made in the three panes for each element across project sessions. Reset the pane selections by clicking Default Settings.

Users can create their own custom templates for the Sections Templates (*.xml) and Microsoft Word templates (*.dotx). When creating a custom XML template, the following syntax must be used:

| Section | Description |
| :--- | :--- |
| TitlePageSection | The title page |
| TOCPageSection | The table of contents |
| SolutionSection | The title of the solution name |
| ProjectSection | The title of the project name |

Chapter 1
Introducing the AADvance-Trusted SIS Workstation software

| Section | Description |
| :--- | :--- |
| ArraysSection | The table displaying arrays |
| StructuresSection | The table displaying structures |
| DefinedWordsSection | The table displaying defined words |
| ConfigurationSection | The title of the controller or device name and the table displaying network links |
| ProgramSection | The title of the program name |
| POUContentSection | The POU diagrams |
| VariableSection | The tables for local and global variables. Also displays the extended attributes <br> for global variables. |
| IOWiringSection | The I\O wiring table |
| TargetSection | The table displaying the targets |
| BindingSection | The table displaying the bindings |

When creating a custom Microsoft Word template (*.dotx), how sections display may be modified, but the following styles and table styles defined by Microsoft Word or AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software must be maintained:

| Style | Description |
| :--- | :--- |
| Heading 1 | How Header 1 displays in the documentation. |
| Heading 2 | How Header 2 displays in the documentation. |
| Heading 3 | How Header 3 displays in the documentation. |
| Heading 4 | How Header 4 displays in the documentation. |
| Heading 5 | How Header 5 displays in the documentation. |
| Heading 6 | How Header 6 displays in the documentation. |
| Heading 7 | How Header 7 displays in the documentation. |
| Heading 8 | How Header 8 displays in the documentation. |
| Heading 9 | How Header 9 displays in the documentation. |
| Alias | How the Alias section displays in the documentation. |
| Comment | How comments displays in the documentation. |
| Table Style | Description |
| IOWiring | The tables displaying information for I/0 wiring and targets. |
| NormalStyle | The tables displaying information for bindings. |
| VariableTableStyle | The tables displaying information for variables, arrays, structures, and defined <br> words. |

## To generate documentation

1. In the Application View, select the element (project, controller or device, POU, library element, etc.) for which to generate documentation.
2. From the File menu, click Generate Documentation.
3. Specify the required options, and then click Generate.
4. In the Save As dialog box, specify the file name and save location, and then click Save.
A progress bar appears over the Generate Documentation dialog box displaying the generation progress. Click Cancel to stop the document generation process. Any files created during the generation process are deleted.
Once generation is complete, the *.docx application displays the documentation.

# Find and Replace Utility 

## Quick Find

Use the Find and Replace to perform these operations:

- Quick Find: Find strings or expressions in files using the Quick Find utility. Quick Find steps from one search result to the next in sequence, either backwards or forwards from the insertion point. When reaching the end or beginning of a document, Quick Find automatically jumps to unsearched sections. When the search is complete, a message displays.
- Quick Replace: Replace strings or expressions in files using the Quick Replace utility. Quick Replace steps from one search result to the next in sequence, either backwards or forwards from the insertion point. When reaching the end or beginning of a document, Quick Replace automatically jumps to unsearched sections. When the search is complete, a message displays.


## See also

Quick Find on page 52
Quick Replace on page 53
Find and replace display settings on page 71
When all search options are defined, choose to find the next instances of the required string or expression within the specified scope.

- When using Find and Replace, perform searches and replacements for a selected POU body.
Press CTRL+F, set the Look In field to Current Document and use only the Find Next or Replace commands. Find and Replace retains the last searched string for a specific document. Enter a different string in the Find what box.

| Search option | Description | Procedure |
| :--- | :--- | :--- |
| Find what | String or expression to find <br> within the open document. | Type the required string into the field, select one of the last twenty searches from the Find <br> what drop-down combo box, and use regular expressions in searches. |
| Look in | Scope for the search. | Select the required scope from the Look in drop-down combo box. |
| Find options | Options to refine the search. | Search for case sensitive matches using Match Case. Disregard partial word matches by <br> selecting Match whole word. Include regular expressions by selecting Use Regular <br> Expressions. Also define the file types to search in the Look at these file types text box. |
| Result options | Options for search results. | Choose to display the results in the Find results 1 window or Find results 2 window. Choose to <br> display only file names in the search results by selecting Display file names only. |


| Task | Procedure |
| :--- | :--- |
| Find a string or expression in a file | Perform searches for a selected program organization unit (POU) body. <br>  <br>  <br>  <br>  <br>  <br>  <br> 1. From the Edit menu, click Find and Replace, and then click Quick Find (or press Ctrl+F). <br> 2. In the Find in Files dialog box, enter the required information: <br> a. In the Find what field, type the string or expression to search for. <br> b. In the Look in field, select Current Document. <br> 3. Click Find Next. |
| Use regular expressions when <br> finding strings or expressions | 1. From the Find utility, expand Find options, and then select Use Regular Expressions. <br> 2. In the Find what field, type the required regular expression. |

## See also

## Quick Replace on page 53

## Quick Replace

When all search options are defined, choose to find the next instance of the required string or expression within the specified scope, then replace individual or all instances of searched items.

| Replace option | Description | Procedure |
| :--- | :--- | :--- |
| Find what | String or expression to find <br> within the open document. | Type the required string or expression in the field, select one of the last twenty searches from <br> the Find what drop-down combo box, and use regular expressions in searches. |
| Replace with | String or expression that will <br> replace each match found. | Type the required string or expression in the field provided, or select one of the last twenty <br> items entered using the drop-down combo box. Delete matches found by leaving the Replace <br> with field empty. Use regular expressions in the Replace with field. |
| Look in | Scope for the search. | Select the required scope from the Look in drop-down combo box. |
| Find options | Options to refine the search. | Search for case sensitive matches using Match Case. Disregard partial word matches by <br> selecting Match whole word. Include regular expressions by selecting Use Regular Expressions. <br> Also define the file types to search in the Look at these file types text box. |
| Result options | Options for search results. | Choose to display the results in the Find results 1 window or Find results 2 window. Choose to <br> display only file names in the search results by selecting Display file names only. |


| Task | Procedure |
| :--- | :--- |
| Replace a string or expression in a <br> file | Perform replacements for a selected program organization unit (POU) body. <br> 1. From the Edit menu, click Find and Replace, and then click Quick Replace (or press Ctrl+F). <br> 2. In the Replace in Files dialog box, enter the required information: <br> a. In the Find what field, type the string or expression to search for. <br> b. In the Replace with field, type the replacement string or expression. <br> c. In the Look in field, select Current Document. |
|  | 3. Click Find Next, Replace, or Replace All. |
| Use regular expressions when <br> replacing strings or expressions | 1. From the Replace in Files dialog box, expand Find options, then select Use Regular Expressions. <br> 2. In the Find what or Replace with fields, type the required regular expression. |

## See also

## Quick Find on page 52

## Spy Lists

Spy to monitor changes in values, for variables, arrays (if supported by the application), structures (if supported by the application), and instances of function blocks by adding them to spy lists directly, from the dictionary, or from the language editors. Create spy lists, then proceed to adding the required items for monitoring while designing, simulating, or running an application. From spy lists, also modify values for the displayed items.

View spy lists in a spy list window.

| Spy List Counter |  |  |  |
| :---: | :---: | :---: | :---: |
| Name: Spy List Counter |  | Refresh Rate: 500 |  |
| Name | Alias | Logical Yalue | Physical Value |
| $\cdots \pi^{*}$ | - $A^{*}$ | $-\operatorname{da}^{*}$ | - $\boldsymbol{f o}^{*}$ |
| InitialValue |  | 3448 | OFFLINE |
| Counter |  | 3455 | OFFLINE |
|  |  |  |  |

Spy list variables properties are:

| Column | Description |
| :--- | :--- |
| Name | Name of the variable or function block instance |
| Alias | Any name (for use in Ladder Diagram (LD) PoUs) |
| Logical Value | Available when online. The displayed value differs depending on the direction: <br> - Input: Locked <br> - Output: Updated by the running TIC code <br> - Internal: Locked |
| Physical Value | Available when online. The displayed value differs depending on the direction: <br> - Input: Updated by the field value <br> - Output: Locked <br> - Internal: Updated by the running TIC code |
| Lock | Available when online. The indication of whether the value of the variable or function <br> block instance is locked. Locking operates differently for simple variables, array and <br> structure elements, and function block parameters. For simple variables, individual <br> variables are locked directly. For structure and array elements, locking an element <br> locks all the elements of the structure or array. |
| Comment | User-defined text |
| Access Path | The location of the variable or function block instance within the project. |
| Data Type | Data type of the variable |

Spy list properties are:

| Properties |  |
| :--- | :--- |
| Name | Name of the spy list that displays in the spy list title bar and the menu. |
| Refresh Rate | When monitoring, the rate at which the values of variables refreshes in the spy list, in <br> milliseconds. Change the refresh rate while in design mode. |

In a spy list, navigate using the mouse controls and arrow keys to move up and down the list.

## Keys

| Arrow keys | Move up or down in the list |
| :--- | :--- |
| Enter key | When selecting variables using the Name field, save the <br> selected variable to the grid. |

Customize spy lists by arranging the columns to display and setting the display colors. In the spy list, refine the contents of the grid by grouping items in a list, sorting items in a list, and filtering items in a list.

| Task | Procedure |
| :--- | :--- |
| Arrange the columns to display | 1. To move a column, drag the column header to another location. <br> When dragging a column header, arrows indicate the current position of the header. <br> 2. To show or hide a column, right-click on a column header, and then click the required column name. |
| Sort items in a spy list | • Click the required column header to toggle the sort order between ascending and descending. |
| Filter items in the grid | Filtering displays only the entries containing specified characters. <br> The filter row is the top row of the grid. Filter items by typing alphabetical and numerical characters in the cells of the filter row. <br> Also select from the drop-down combo box. Matching items automatically display. <br> - In the filter row of the spy list, click the required cell, then do one of the following: <br> - Type the characters to use in the filtering operation <br> $\bullet$ - Select the required structure from the drop-down combo box |
| Group items in a spy list | • Drag the required column header to toggle the sort order between ascending and descending. |

When managing spy lists, perform these tasks:

| Task | Procedure |
| :---: | :---: |
| Create a spy list | 1. From the View menu, point to Watch, and then click Create Spy List. <br> 2. To assign a name other than the default name, specify the required name in the Name field of the spy list window. A spy list with an empty grid displays. |
| Access an existing spy list | - From the View menu, point to Watch, and then click the required spy list. |
| Delete a spy list | 1. From the View menu, point to Watch, and then click the name of the spy list to remove. <br> 2. In the spy list window, verify the correct name of the spy list to remove, and then click |
| Add items to a spy list | 1. To add items directly in a spy list: <br> a. From the View menu, point to Watch, and then click the required spy list name. <br> b. In the spy list window, click the available Name cell and type the specific alphanumerical characters contained in the name of the item to add to the spy list. <br> c. From the drop-down list displaying the available items with the characters in their name, select the required item and then press Enter. <br> 2. To add items from the dictionary: <br> a. Access the dictionary instance containing the item to add to the spy list. <br> b. In the dictionary instance, right-click the name field for the item to add, point to Add to Spy List, and then click the name of the required spy list from the available list. <br> 3. To add items from a language editor: <br> a. Access the POU containing the item to add to the spy list. <br> b. In the language container, right-click the item to add, point to Add to Spy List, and then click the name of the required spy list from the available list. |
| Remove items from a spy list | When selecting an item, an indicator arrow displays in the leftmost column of the list. For AADvance projects, remove unused items from a spy list. <br> - In the spy list, right-click the item to remove, and then click Delete. |
| Save a spy list | - From the required spy list, click the Close button at the top-left corner of the spy list window. |
| Cut, copy, and paste items between spy lists | 1. In the grid of the required spy list, cut or copy the required items. <br> - To remove variables, select the required item or items, right-click the selection, and then click Cut. <br> - To copy variables, select the required item or items, right-click the selection, and then click Copy. <br> 2. In the grid of the required spy list, right-click the required location, and then click Paste. |
| Drag items between spy lists | 1. Access the spy lists containing the required items and their destination. <br> 2. From the spy list containing the required items, select the items. <br> The selection indicator displays in the leftmost column. <br> 3. Drag to the destination, placing it at the required location within the list. |
| Force the value of a spy list variable | For spy list variables with a write access value of false, restrict access before forcing the values. For variables with a write access value of true: <br> - From the spy list, double-click the variable's corresponding cell in the Value column, then in the dialog box, click Write. For Boolean variables, click the required Boolean value. |

## See also

## Offline Grid Settings on page 100

Online Grid Settings on page 100

## External Tools

Launch external tools and applications by adding items to the Tools menu. Also create keyboard shortcuts for external tools added to the Tools menu. Supported file types include .exe, .bat, .com, .cmd, and .pif.

When specifying a tool for handling arguments, the required argument is immediately transferred to the tool when the external tool is launched. At this time, choose to edit required arguments. Upon subsequent startups of the external tool from the Tools menu, selected arguments are automatically passed to the tool. When Prompt for Arguments is selected, the Arguments dialog box displays.

Define a working directory for tools or commands. Also specify additional arguments when the command is launched.

From the External Tools dialog box, perform these tasks:

| Task | Procedure |
| :--- | :--- |
| Add an external tool | 1. From the Tools menu, click External Tools. <br> 2. In the External Tools dialog box, in the Title field, type a name for the menu option. To include a keyboard shortcut, type an <br> ampersand (\&) before the letter in the title to use as shortcut. For example: "My External Tool", the letter "x" is the keyboard <br> shortcut. <br> 3. In the Command field, type the path to the file, or browse for the file by clicking <br> 4. (optional) Select the Use Output window and Close on exit check boxes. The Use Output window option is only available for <br> .bat and .com files. |
| 5. Click Add, then click OK. |  |

Working in the Development Environment

In the development environment, perform many tasks:

- View output messages in the Output window
- View build errors in the Error List
- Zoom the contents of program organization units (POUs)
- Arrange and dock windows
- Perform searches of the development environment from different windows
- Navigate the development environment with the Windows dialog box
- Customize, create, and rename toolbars and commands
- Export, import, or reset environment settings with the Import and Export Settings Wizard
- Use keyboard shortcut combinations to perform multiple actions


## See also

## Display the Output Window on page 57

Arrange and Dock Windows on page 58
Navigate in the Development Environment on page 61
Import and Export Environment Settings on page 65
Development Environment Keyboard Shortcuts on page 68

## Zooming the Contents of Active POUs

Set the zoom factor for viewing the contents of individual active graphic program organization unit (POU) documents:

- Zoom in, enlarges the contents of an active graphic POU document
- Zoom out, provides a larger overview of the contents of an active graphic POU document, as if viewed from further away
- Zoom to fit all, sets the magnification factor to display the contents within the width and height of an active graphic POU document
- Zoom to page width, sets the magnification factor to display the contents within the width of an active graphic POU document
- Zoom percentage factor, enables specifying the magnification factor for the contents of an active graphic POU document


## To set the zoom factor for an active POU document

1. Click the active graphic POU document for which to adjust the zoom factor.
2. From the Zoom toolbar, click the required zoom option.

## See also

Display the Output Window on page 57

## Displaying the Output Window

## To access the Output window

From the View menu, click Output (or press Ctrl+Alt+O).
Review generated messages from the Output window. From the Output window, perform these tasks:

| Task | Procedure |
| :--- | :--- |
| Review generated status messages | • In the Output window, from the Show output from drop-down combo box, click the required feature. |
| Manage the contents of the Output | Manage the word wrapping and clear the contents of the window. |
| window | • To wrap text to continue on the next line, click |
|  | - To delete the contents of the window, click |

Output window toolbar commands are:

| Show output from: | Selects individual features for which to view generated status <br> messages |
| :--- | :--- |
| Find Message in Code | Moves the insertion point to the line in the language editor containing <br> the selected build error |
| Go to Previous Message | In the Output window, jumps to the previous build error message. In <br> the code editor, locates the build error and automatically moves the <br> insertion point to the error. |
| Go to Next Message | In the Output window, jumps to the next build error message. In the <br> code editor, locates the build error and automatically moves the <br> insertion point to the error. |
| Clear all | In the Output window, deletes all displayed messages. |
| Toggle Word Wrap | Wraps text to continue on the next line for messages extending beyond <br> the viewing area |

## See also

## Use the Error List on page 58

## Use the Error List

Use the Error List to view errors, warnings, and messages produced when editing program organization units (POUs) and performing build operations.

Navigate from one error to the next using the contextual menu options. Also navigate between errors using the keyboard arrows.

| Column | Description |
| :--- | :--- |
| Category | Displays an icon identifying the type of error |
| Default Order | Displays an integer indicating the order in which the error <br> occurred relative to the other errors |
| Description | Displays the error message text |
| File | Displays the POU name or the POU location and POU name |
| Line | Displays the line number |
| Column | Displays the column number |
| Project | Displays the name of the project |

Error List toolbar commands are:

| Icon | Description |
| :--- | :--- |
| 2 Errors | Displays the number of generated errors. Click to toggle <br> between displaying and hiding the errors in the list. |
|  | Displays the number of generated warnings. Click to toggle <br> between displaying and hiding the warnings in the list. |
| (i) 0 Messages | Displays the number of generated messages. Click to toggle <br> between displaying and hiding the messages in the list. |

Sort the contents of the Error List. Customize the Error List by resizing columns and arranging the columns to display.

| Task | Procedure |
| :--- | :--- |
| Display the Error List | • From the View menu, click Error List (or press Ctrr $+\mathrm{l}, \mathrm{Ctrl}+\mathrm{E}$ ). |
| Sort errors | - In the Error List, click the required column heading for which to sort. To further sort the list, click another column <br> heading while pressing Shift. |
| Search messages in the Error List | • In the Search Error List field of the Error List, enter text contained in the message to locate. |
| Customize the Error List | 1. To move a column, drag the column heading to the required location. <br> 2. To modify the width of columns, drag the column dividers to the required location. |

## See also

## Display the Output Window on page 57

## Arranging and Docking Windows

Arrange the development environment by placing tool windows and document windows in various positions. Undock and move tool windows or document windows from their default location. Also dock multiple windows
together and move these as a group. While moving windows, a guide diamond indicates the available positions for the window.


Tool windows are available from the View menu. Open document windows are listed in the Window menu with the top-most window listed first.

For tool windows like the Application View, Communication View, Equipment View, Toolbox, and Properties windows, choose the following window positions:

- Float, the window floats over or outside the workspace
- Dock, the window remains open at a chosen location
- Dock as Tabbed Document, the window is placed as a tabbed document
- Auto Hide, the window is reduced to a navigation bar item on the edges of the workspace. Clicking toggles between showing and reducing the window.
- Hide, the window does not display in the development environment

For document windows like programs, choose the following positions:

- Float, the window floats over or outside the workspace
- Dock, the window remains open at a chosen location
- Dock as tabbed, the window is placed as a tabbed document

When omitting to set floating tab wells or tool windows to stay on top of the main window, use the View > All Windows option to bring forward windows hidden behind the development environment or any other application on the desktop.

| Task | Procedure |
| :--- | :--- |
| Display tool windows and document <br> windows | 1. For a tool window, click the window to display from the View menu. <br> 2. For a document window, double-click the document from the respective view. For example, open a program by double- <br> clicking it from the Application View. |
| Dock tool windows and document <br> windows | 1. Locate and click the tool window or document window to dock. <br> 2. Drag the window toward the center of the workspace. To move a docked window without snapping it into place, press <br> Ctrl while dragging the window. <br> A guide diamond appears. The four arrows of the diamond point toward the four sides of the workspace. For tool <br> windows, an additional four arrows point to the four edges of the workspace. |
|  | 3. When the window reaches the docking location, move the pointer over the corresponding portion of the guide diamond. |
|  | The shaded area indicates the respective window position. |
|  | 4. To dock the window in the indicated position, release the mouse button. |
|  | 5. To return a tool window or document window to its previous position, double-click its title bar while pressing Ctrl. |

## See also

Arrange the windows layout on page 60
Display the Output Window on page 57
Zoom the Contents of Active POUs on page 56

## Arrange the windows layout

When arranging the windows layout, auto hide tool windows, or display windows as tabbed documents within one or more tab groups. Reduce tool windows set to auto hide to a navigation bar item on the edges of the workspace and remain open when closing all documents. Reset the windows layout to the default layout. Return a window to its previous position by double-clicking the title bar while pressing Ctrl.

## To arrange the windows layout

1. To auto hide a tool window, dock the window and then click Auto Hide from the Window menu.
2. To display a window as a tabbed document, click the window and then click Dock as Tabbed Document from the Window menu.
3. To add a horizontal or vertical tab group for tool windows or document windows:
a. For a horizontal tab group, click a window to add to the new group and then click New Horizontal Tab Group from the Window menu. The window displays in the new tab group.
b. For a vertical tab group, click a window to add to the new group and then click New Vertical Tab Group from the Window menu. The window displays in the new tab group.
c. To move subsequent windows to the next tab group, click each window and then click Move to Next Tab Group from the Window menu.
4. To close a tool window, click the tool window and then click Hide from the Window menu.
5. To close a document window, click the Close button on the window tab.
6. To close all tool windows and document windows at once, click Close All Documents from the Window menu. The windows set to auto hide remain open.
7. To bring forth floating tab wells and tool windows hidden behind the main window or any other application on the desktop, click All Windows from the View menu. This feature is only available when you clear the Floating tab wells always stay on top of the main window and Floating tool windows always stay on top of the main window items in the Tabs and Windows environment options in the Options dialog box.
8. To reset the windows layout to the default layout, click Reset Window Layout from the Window menu. Click Yes when prompted to confirm the restoration of the windows default layout.

## See also

## Searching in the Development Environment

## Navigating in the Development Environment

Arrange and Dock Windows on page 58
Perform searches for text strings in various windows of the development environment. Use these searches to locate specific information or filter displayed information.

- Options dialog box (available from the Tools menu), to search for text contained in entries of the table of contents
- Toolbox, to filter the number of displayed elements by entering text contained in the required elements
- Error List, to filter the number of displayed messages by entering text contained in the required messages. Press Esc to return filtered messages to their unfiltered results. Press Enter to open the document containing the source of a displayed error.
- Block Library, to filter the number of displayed blocks by entering text contained in the required blocks. For example, a search for "and" results in the display of the AND, AND_MASK, and RAND standard blocks.


## To search in the development environment

- In the respective tool window or dialog box, enter the required text in the search field.
The search results display the information containing the specified text.


## See also

Toolbox on page 39
Use the Error List on page 58
Block Library on page 36
Options for the Development Environment on page 70
Navigating in the development environment is simplified with the use of the Windows dialog box. The Windows dialog box displays the active files open in
the current project. Active files consist of language containers, the Application View and other windows docked in the workspace.

| Windows | Path |
| :--- | :--- |
| Name Application View <br> Communication  <br> Communication View  <br> Equipment  <br> Equipment View  |  |
| FB1-POU <br> Prog1-POU <br> Prog2-POU <br> Prog3-POU <br> Prog4-POU |  |

From the Windows dialog box, perform these tasks for active files:

- Switch between active files
- Save changes to one or more active files
- Close active files


## To navigate using the Windows dialog box

1. From the Window menu, click Windows.

The Windows dialog box displays the list of active files.
2. To switch to another active file in the list, select the required file, then click Activate.
3. To save changes to active files, select the required files from the list, then click Save.
4. To close active files, select the required files from the list, then click Close Window(s).

## See also

Language Editor on page 26
Application View on page 24
Arrange and Dock Windows on page 58

## Customize Toolbars

For standard toolbars, modify docking locations. For custom toolbars, modify docking locations, rename toolbars, and delete toolbars. The Customize dialog box lists the provided toolbars and any custom user toolbars.

## To customize a toolbar

1. From the Tools menu, click Customize...
2. From the Customize dialog box, click the Toolbars tab, make the required changes, and then click Close.

- To modify the docking location for a toolbar, select the required toolbar from the Toolbars list, click Modify Selection, then click the preferred location for docking the toolbar. Available docking locations are top, left, right, and bottom.
- To rename a custom toolbar, select the required toolbar from the Toolbars list, click Modify Selection, and then type the required name in the text field.
- To delete a custom toolbar, select the required toolbar from the Toolbars list, and then click Delete.


## See also

## Create Toolbars on page 63

Customize Commands on page 63

## Create Toolbars

## To create a custom toolbar

1. From the Tools menu, click Customize...
2. From the Customize dialog box, click the Toolbars tab, and then click New.
3. In the New Toolbar dialog box, type a name for the custom toolbar, and then click OK.

The custom toolbar name adds to the Toolbars list.

## See also

Customize Toolbars on page 62
Customize Commands on page 63
Customize menu bar, toolbar, and contextual menu commands by selecting a set of commands, then choosing an individual command to modify using the available options. Add, rename, reset, delete, and rearrange the order of commands in the menus. Also delimit groups of commands in menus and specify display options.

Customize different levels and options for menu items including categories and subcategories.

Perform these menu tasks:

| Task | Procedure |
| :--- | :--- |
| Add a menu category to the menu | 1. From the Tools menu, click Customize... <br> bar |
|  | 2. From the Customize dialog box, click the Commands tab. <br> 3. Select Menu Bar from the Menu bar drop-down combo box. <br> 4. To add a menu category to the menu bar, click Add New Menu. <br> 5. The menu category is added to the menu bar. <br> 6. Rename the menu item by clicking Modify Selection, then typing the required name in the text field. |
| Add a menu item to an existing | Menu items are either commands or subcategories leading to submenus. Before adding a menu item, arrange the required <br> order by selecting the menu item following the location of the new item in the list or rearranging the menu items after <br> insertion. |
| menu category, toolbar, or |  |
| contextual menu | 1. From the Tools menu, click Customize.... <br> 2. From the Customize dialog box, click the Commands tab. <br> 3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo boxes. <br> 4. Perform one of these operations: <br> a. To add a menu item to an existing menu category, toolbar, or contextual menu, select the item following the location <br> for the new item, and then click Add New Menu. <br> b. To add a command to an existing menu category, toolbar, or contextual menu, select the item following the location for <br> the new item, click Add Command. From the Add Command dialog box, select the category, choose from the available <br> commands in the Commands list, and then click OK. <br> The menu item adds to the existing menu category, toolbar, or contextual menu. |
| 5. To rename the menu or command, click Modify Selection, and then type the required name in the text field. |  |


|  |  |
| :--- | :--- |
| Rearrange menu items | 1. From the Tools menu, click Customize... <br> 2. From the Customize dialog box, click Commands. <br> 3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo-boxes. <br> 4. To place the menu item at a different location in the selected menu or toolbar, select the menu item in the Controls list, <br> then click Move Up or Move Down to move across the existing menu items. |

## See also

## Specify the command display options on page 65 <br> Customize Toolbars on page 62 <br> Create Toolbars on page 63

## Specify the command display options

## Importing and Exporting Settings

## Export Selected Environment Settings

Initially, the display options for commands are set to default. In menus, the default display option is Image and Text, while in toolbars it is Text Only (in Menus). The Text Only (in Menus) option displays an image in a toolbar or text in a menu. The Text Only (Always) option displays text in a menu or toolbar. The Image and Text option displays both image and text in a menu or toolbar. Do not associate an image with a command.

## To specify the display options for a command

1. From the Tools menu, click Customize...
2. From the Customize dialog box, click the Commands tab
3. Select the menu to modify from the Menu bar, Toolbar, or Context menu drop-down combo boxes.
4. To specify the display options, select the required command in the Controls list, click Modify Selection, then click one of these:

- Default style
- Text Only (Always)
- Text Only (in Menus)
- Image and Text


## See also

## Customize Commands on page 63

Import or export specific categories of settings, or reset the environment to one of the default collections of settings. The environment settings include the settings for the various development views, editors, and tools.

- Export Selected Environment Settings
- Import Selected Environment Settings
- Reset all Settings


## To import, export, or reset environment settings

1. From the Tools menu, click Import and Export Settings.
2. Select the required option, then follow the on-screen instructions.

## See also

Import and export settings on page 72
Import Selected Environment Settings on page 66
Reset all Settings on page 67
Operations Summary on page 68
When exporting selected environment settings, choose the settings to export from the list of available environment settings. Environment settings identified with a warning symbol are not selected by default since these may
contain intellectual property or sensitive information. Some categories may have sub-categories visible upon expanding the arrows to the left of the category item.

The settings exportation process requires the following operations:

1. Choose environment settings to export.
2. Name a settings file.

During the environment settings export process, a window indicates the progress of the operation. Upon completion of the environment settings export process, a summary page shows the results of the operation.

## See also

Naming a Settings File on page 66
Import Selected Environment Settings on page 66
Reset all Settings on page 67

Naming a Settings File

When exporting selected environment settings, a settings file to store the exported settings must be specified. The default location of this settings file is:
\%USERPROFILE\% \Documents <version>\Settings\SIS Workstation

## See also

## Settings Export in Progress on page 66

## Settings Export in Progress

Import Selected Environment Settings

During the environment settings export process, a window indicates the progress of the operation.

When importing selected environment settings, choose a file containing the settings to import, then select the required settings to import from the list of available environment settings in the file. Environment settings identified with a warning symbol are not selected by default since these may contain intellectual property or sensitive information. Some categories may have subcategories visible upon expanding the arrows to the left of the category item.

The settings importation process requires these operations:

1. Choose whether to save the current environmental settings or overwrite the current setting with the settings to import.
2. Choose a file containing the collection of environmental settings to import.
3. Select the individual settings to import from the list of available environment settings in the settings file.

During the environment settings import process, a dialog indicates the progress of the operation. Upon completion of the environment settings import process, a summary page shows the results of the operation.

## See also

Choosing a Collection of Settings to Import on page 67
Choosing Settings to Import on page 67
Settings Import in Progress on page 67
Reset all Settings on page 67

# Choosing a Collection of Settings to Import 

When importing selected environment settings, choose a file containing the settings to import.

## See also <br> Choosing Settings to Import on page 67 <br> Settings Import in Progress on page 67

## Choosing Settings to Import

When importing selected environment settings, select the required settings to import from the list of available environment settings contained in the settings file. Environment settings identified with a warning symbol are not selected by default since these may contain intellectual property or sensitive information. Some categories may have sub-categories visible upon expanding the arrows to the left of the category item.

## See also

Settings Import in Progress on page 67
During the environment settings import process, a window indicates the progress of the operation.

## See also

Choosing a Collection of Settings to Import on page 67
Choosing Settings to Import on page 67
Revert the environment settings to the initial settings. When resetting the environment settings, choose whether to save the current environment settings to a file.

## See also

Settings Reset in Progress on page 67
Import Selected Environment Settings on page 66
During the environment settings reset process, a window indicates the progress of the operation.

## See also

## Reset all Settings on page 67

## Operations Summary

When performing one of these tasks regarding the environment settings, the wizard displays the results (whether successful or unsuccessful) for the operation.

- Export Selected Environment Settings
- Import Selected Environment Settings
- Reset all Settings


## See also

Import Selected Environment Settings on page 66
Reset all Settings on page 67

## Development Environment Keyboard Shortcuts

When working in the development environment, use keyboard shortcuts for the following tasks:

- Accessing windows
- Connecting and simulating
- Navigating in the dictionary
- Getting help
- Saving and closing
- Working with the cross reference browser and find utility
- Zooming the contents of active POUs
- Navigating in the development environment

Some keyboard shortcuts do not apply or may differ while debugging.
Keyboard shortcuts specific to the programming languages, debugging, and version control are indicated on their respective keyboard shortcut pages.

## Accessing Windows

| Ctrl+Alt+T | Accesses the Block Library |
| :---: | :---: |
| Ctrl+W, Crrl+C | Accesses the Cross Reference Browser |
| Ctrl $+1, C$ Crl + E | Accesses the Error List |
| Ctrl+Shift+N | Accesses the New Project dialog box (not available while debugging) |
| Ctrl+Shift+0 | Accesses the Open Project dialog box (not available while debugging) |
| Ctrl+Alt+0 | Accesses the Output window |
| F4 | Accesses the Properties window |
| Alt+Enter | Accesses the Properties window |
| Ctrl +F | Accesses the Quick Find utility |
| Ctrl+ H | Accesses the Quick Replace utility |
| Shift+Alt+M | Brings to the front floating windows |
| Ctrl+Alt+X | Accesses the Toolbox |

## Connecting and Simulating

| Ctrl+Shift $+B$ | Builds the project (not available while connected) |
| :--- | :--- |


| F5 | Displays the Connect dialog box |
| :--- | :--- |
| F10 | While debugging, steps over the next rung or line of code |
| F11 | While debugging, steps into the next rung or line of code |
| Shift + F5 | Disconnects from the controllers |
| Ctrl + D | Only available in debug mode for the date data type. When the Write Logical <br> Value dialog box is open, enters the current date. |

## Navigating in the Dictionary

| Up Arrow | Moves up the grid between cells |
| :--- | :--- |
| Down Arrow | Moves down the grid between cells |
| Left Arrow | Moves left across the grid between cells |
| Right Arrow | Moves right across the grid between cells |
| CtrI + PLUS SIGN on numeric <br> keypad ( + ) | Expands the fields of complex data types |
| Ctrl + MINUS SIGN on numeric <br> keypad ( - ) | Collapses the fields of complex data types |

## Getting Help

| Ctrl + F1 | Accesses the Help Viewer |
| :--- | :--- |
| Shift + F1 | Accesses help for the selected window |
| F1 | Accesses help for the selected element |

## Saving and Closing

| Ctrl + S | Saves the selected elements (not available while debugging) |
| :--- | :--- |
| Ctrl + Shift $+S$ | Saves all files making up a project (not available while debugging) |
| Alt + F4 | Exits the AADvance- -Trusted ${ }^{\oplus}$ SIS Workstation software |
| Ctrl + F4 | Closes files and windows located in the workspace |
| Shift + Esc | Closes selected windows except for POUs |

## Working with the Cross Reference Browser and Find utility

| Ctrl $+\mathrm{T}, \mathrm{Ctrl}+\mathrm{R}$ | Refreshes the Cross Reference Browser data |
| :--- | :--- |
| F8 | Jumps to the selected instance of an element |
| Shift + F8 | Jumps to the selected instance of an element |
| F3 | Finds next text in a selected window |

## Zooming the contents of active POUs

| Ctrl+mouse wheel | Increases and decreases the magnification factor of the contents of POUs |
| :--- | :--- |

## Navigating in the Development Environment

| Shift + F10 | Displays the contextual menu for the selected item |
| :--- | :--- |


| Alt +- | For language containers, displays various menu options including saving, <br> docking, and tiling. <br> For other windows, displays docking options. |
| :--- | :--- |
| Ctrl + F6 | Navigates to the next Active File |
| Ctrl+Shift + F6 | Navigates to the previous Active File |
| Alt + F6 | Navigates to the next Active Tool Window |
| Alt+Shift + F6 | Navigates to the previous Active Tool Window |

## See also

Navigate in the Development Environment on page 61
Find and Replace on page 52
When setting the development options, customize aspects of the development environment including:

- Setting Environment Options
- Specifying Project Options
- Specifying Block Library Settings
- Specifying Trusted Settings
- Specifying Cross Reference Browser Settings
- Specifying Documentation Generator Options
- Setting Grid Options
- Setting IEC Language Options
- Setting ISaVIEW Options
- Defining Spy List Settings


## To search entries in the Options table of contents

- In the Search field of the Options dialog box, enter text contained in the entry to locate in the table of contents.

The Options table of contents displays only the entries containing the specified text.

## See also

Setting Environment Options on page 70
Specify Project Options on page 75
Setting Grid Options on page 79
Setting IEC Language Options on page 84
Setting ISaVIEW Options on page 92
Define the environment options for the following:

- Find and Replace Settings
- Import and Export Settings
- Shortcut Keyboard Combinations
- Startup
- Tabs and Windows

Modify the general settings for the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software by accessing the general environment options. Some changes to the general settings take effect after restarting the software.

- Visual experience, indicates the color theme to use in the development environment and specifies whether the visual experience is set automatically or explicitly. For the color theme, the possible colors are blue and light. For the visual experience, the adjustment may change the display of colors from gradients to flat colors, or it may restrict the use of animations in menus or popup windows. Enabling the full visual experience includes gradients and animations. Clear this option when using remote desktop connections or older graphics adapters because these features may have poor performance in such cases. Use hardware graphics acceleration if available rather than software acceleration.
- Window menu, indicates the number of windows (ranging from 1 to 24) that display in the Windows list of the Window menu. For the number of items shown in the Window menu, the default value is 10 .
- Recently used, indicates the number of recently used files displayed in menus. The items shown in recently used lists field defines the number of recent projects and files (ranging from 1 to 24) that display in the File menu. For the number of items shown in recently used lists, the default value is 4 .
- Show status bar, displays the status bar showing progress information for ongoing operations.
- Close button affects active tool window only, enables the Close button to shut down the active window only. This option is selected by default.
- Auto Hide button affects active tool window only, enables Auto Hide to hide the active window only.


## To access the general environment options

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Environment, then click General.

## See also

Find and replace display settings on page 71
Import and export settings on page 72
Shortcut keyboard combinations on page 73
Startup on page 74
Tabs and windows options on page 74

Find and replace display settings

Define the display settings for the Find and Replace dialog box. Choose to display informational messages and warnings and populate the Find What field with text from an open editor.

## To define the display settings for the Find and Replace dialog box

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Environment, then click Find and Replace.
3. In the Options dialog box, select the required options, then click OK.

## See also

Find and Replace on page 52
Import and export settings on page 72
Shortcut keyboard combinations on page 73
Startup on page 74
Tabs and windows options on page 74

## Import and export settings

Define options for saving settings files. Choose to save settings to a *.vssetting file located on the system or to a shared settings files. When saving settings to a shared *.vssettings file, provide a UNC path or local path to the shared file.

- Automatically save my settings to this file, displays the name and path to the *.vssettings file currently in use. Change the setting file used by typing a different path or browsing to locate the required settings file on the system.
- Use team settings file, enables navigating to a shared *.vssettings file. Browse to locate the required settings file. This *.vssettings file is automatically re-applied following each modification.


## To define the options for saving the settings file

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Environment, then click Import and Export Settings.
3. In the Options dialog box, define the required name and location of the settings file, then click OK.

## See also

Import and Export Environment Settings on page 65
Shortcut keyboard combinations on page 73
Startup on page 74
Tabs and windows options on page 74
Find and replace display settings on page 71

## Shortcut keyboard combinations

Use the keyboard options to perform tasks regarding the keyboard shortcuts for the various commands available in the development environment.

Use keyboard shortcuts for quicker operation of the environment. The keyboard options enable viewing the defined keyboard shortcuts mapping schemes available for commands. Only the default keyboard shortcut mapping scheme is available. View commands in the Show commands containing section listing all available commands and their respective keyboard shortcuts. In the text field, type text to find a specific command. By default, only some commands have pre-defined shortcuts. Users can define (add) a shortcut to a command or modify an existing shortcut by adding a new shortcut and removing an unwanted shortcut. Manage keyboard combinations from these options:

- Apply the following additional keyboard mapping scheme, only the default mapping scheme.
- Show commands containing, displays all available commands. The list displays all entries containing the specified characters when typing characters in the text box.
- Shortcuts for selected command, lists mapped keyboard shortcuts for the command selected in the Show commands containing list.
- Use new shortcut in, specifies the scope of the keyboard shortcut. Use the shortcut globally in the development environment or only within a specific context (or window). The default setting is global, meaning the shortcut key works in any active window. If a global keyboard shortcut and context specific shortcut are identical, the context specific shortcut takes precedence. For example, commands with the language editor scope have precedence over commands with the global scope. A context specific keyboard shortcut remains in effect only while the context (or window) is active.
- Press shortcut keys, enables pressing a key combination to be used for the currently selected command. Use one or more modifier keys like CTRL, ALT, or SHIFT combined with various keys. SHIFT cannot be combined with letters or numbers. The F1-F12 keys can be used with or without a modifier. Enter one or two key combinations to use as a shortcut. For example, enter CTRL+Y, or enter F6, CTRL+Y. Regardless of their scope, shortcut key combinations cannot contain these keys:
- Prt Scr/Sys Rq
- Application key
- ScrLk
- Num Lock
- Caps Lock
- Ctrl+Alt+Delete key combination
- Escape
- Shortcut currently used by, displays the command assigned to the current keyboard shortcut combination. The text box is only activated when assigning a key combination that is already assigned to another command. To replace the current shortcut keyboard combination with a custom one, define a new keyboard shortcut mapping scheme.

Tasks to perform are:

| Task | Procedure |
| :--- | :--- |
| View existing commands and <br> keyboard shortcuts | 1. From the Tools menu, click Options. <br> 2. In the Options dialog box, expand Environment, then click Keyboard. <br> The keyboard options display in the Options dialog box. <br> 3. In the Show commands containing field, scroll to find a command or type the required command name without spaces. For <br> example, ShowNextStatement. <br> 4. In the Show commands containing list, select the required command. For example, Debug.ShowNextStatement. <br> The shortcuts for the selected command drop-down combo box displays the required shortcut key combinations. |
| Define keyboard shortcuts | 1. From the keyboard options, in the Show commands containing field, type the required command name without spaces. For <br> example, ShowNextStatement. |
|  | 2. In the Show commands containing list, select the required command. For example, Debug. ShowNextStatement. <br> 3. In the Use new shortcut in drop-down combo box, select the scope. For example, MLGE. <br> 4. In the Press shortcut keys field, type the new key combination. <br> 5. Click Assign, then click OK. |
| Remove keyboard shortcuts | 1. From the keyboard options, in the Show commands containing field, type the required command name without spaces. For <br> example, ShowNextStatement. |
| 2. In the Show commands containing list, select the required command. For example, Debug.ShowNextStatement. |  |
| 3. In the Shortcuts for selected command field, select the keyboard shortcut to be removed. |  |
| 4. Click Remove. |  |

## See also

## Tabs and windows options on page 74

## Startup

Specify the startup behavior when launching the development environment:

- Open Home Page, automatically displays the home page
- Load last loaded solution, opens the last opened project
- Show Open Project dialog box, automatically displays the Open Project dialog box
- Show New Project dialog box, automatically displays the New Project dialog box
- Show empty environment, opens without displaying any project or dialog box


## See also

Import and export settings on page 72

## Tabs and windows options

Arrange tabs and windows in the development environment. These options also enable specifying the mouse-over behavior for tool windows.

| Tab or Window | Options |
| :---: | :---: |
| Tab Wells | - Insert new tabs to the right of existing tabs, opens document windows as new tabs to the right of existing tabs on tab wells. The default placing of new tabs on tab wells is to the left of existing tabs. <br> - Floating tab wells always stay on top of the main window, keeps tab wells set to floating over or outside the development environment on top of the main window. When omitting to set floating tab wells to stay on top of the main window, the View > All Windows menu item is available when the environment includes at least one floating tab well. <br> - Floating tool windows always stay on top of the main window, keeps tool windows set to floating over or outside the development environment on top of the main window. When omitting to set floating tool windows to stay on top of the main window, the View > All Windows menu item is available when the environment includes at least one floating tool window. |
| Pinned Tabs | - Show pinned tabs in a separate row, places pinned tabs in a row separate from unpinned tabs. <br> - Show pin button in unpinned tabs, displays and makes available pin buttons in unpinned tabs. <br> - Maintain pin status if document is removed from well, holds the pinned or unpinned status for a document tab following manipulations including closing and reopening. |
| Preview Tab | - Allow new files to be opened in the preview tab, Not supported <br> - Preview selected files in Solution Explorer (Alt+click to avoid previewing), Not supported <br> - Preview selected files in Find Results, Not supported <br> - Preview selected files in Navigate to, Not supported |
| Tool Windows | - Show auto-hidden windows on mouse over, displays tool windows set to auto hide upon mouse-over. |

## To specify tabs and windows options

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Environment, then click Tabs and Windows.
3. In the Options dialog box, select the required options, then click OK.

## Specify Project Options

## See also

## Shortcut keyboard combinations on page 73

Specify the default locations and behavior of project components. Set default paths for projects and templates. For the Output window and Application View, set the default behavior during project creation and building. Also set the options for building projects.

- Projects location, User project templates location, and User item templates location, enable defining the default path to project folders used in dialog boxes. The Projects location path is used in the Open Project dialog box to define the My Projects location. The User project template location is used in the New Project dialog box to define the My templates list. The User item template location is used in the Add New Item dialog box to define the My Templates list. When defining these default paths, type directly in the field or browse for the required location.

Tip: Limit the Projects location path length to 70 characters. Setting a longer path may exceed the maximum length of 260 characters supported by Windows operating systems. When setting the Projects location, specify a local path; network paths may cause unexpected results.

- Always show Error List if build finishes with errors, enables opening the Error list window when errors occur during a build operation. When the build operation is complete, the Error List displays containing the errors generated by the build operation.
- Track Active Items in Solution Explorer, Not supported
- Show advanced build configurations, Not supported
- Always show solution, Not supported
- Save new projects when created, Not supported
- Warn user when the project location is not trusted, displays a warning message when opening projects from an untrusted location.
- Show Output window when build starts, enables displaying the Output window when starting build operations.
- Prompt for symbolic renaming when renaming files, enables displaying a message prompting to select whether to rename all references in the project or just the selected file.


## To specify the default locations and behavior of project components

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Projects, then click General.
3. In the Options dialog box, type the required paths or browse for their locations, select the required options, then click OK.

## See also

## Build options on page 76

Online settings on page 77

## Build options

Specify whether a message displays before cleaning operations execute. After performing cleaning operations, online changes are unavailable.

- Proceed to cleaning without asking, displays a message indicating that online updates become unavailable after performing a cleaning operation.


## To enable the display of messages prior to cleaning operations

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Projects, then click Build.
3. In the Proceed to cleaning without asking drop-down combo box, select False, then click OK.

## See also

## Online settings on page 77

## Online Settings

Specify Block Library settings

When monitoring applications, display messages prompting to confirm the locking or unlocking of variables. Also specify the number system and number of significant digits used to display the numerical values of the different data type categories.

- Prompt for Lock or Unlock, displays messages prompting users to confirm the locking or unlocking of selected variables.
- Bool display format, displays Boolean values in bool (TRUE/FALSE), bit (1/0), or mixed (TRUE (1)/FALSE (0)) format.
- Integer, displays integer values in decimal, hexadecimal, octal, or binary format.
- REAL, displays REAL values using scientific notation or a specific number of significant digits after the decimal.
- LREAL, displays LREAL values using scientific notation or a specific number of significant digits after the decimal.


## To enable the message prompts display when locking and unlocking variables

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Projects, then click Online.
3. Select the Prompt for Lock or Unlock option.
4. (optional) In the Numerical Display section, set the required values for the different data type categories.
5. Click OK.

## See also

Build options on page 76
Specify the display mode for the Block Library on startup. Display the Block Library using expanders or tabs.

## To adjust the display mode for the block library

1. From the Tools menu, click Options.
2. In the Options dialog box, expand Block Library Settings, then click General.
3. In the Options dialog box, select the required display mode from the drop-down menu, and then click OK.

## Specify Trusted settings

For Trusted variables, specify the display format for the Modbus address. The Modbus address displays in hexadecimal or decimal format.

# To adjust the Modbus address format for variables 

2. In the Options dialog box, expand CAM Trusted Settings, and then click General.
3. Select the required display type from the drop-down menu, and then click OK.

## See also

Specify Block Library settings on page 77
Specify Cross Reference Browser settings on page 78

## Specify Cross Reference Browser settings

## Specify Documentation Generator options

For Trusted ${ }^{\circledR}$ projects, customize the colors that display in the Cross Reference Browser including the colors for the grid rows, alternating grid rows, selected references, selected symbols, and more. The Cross Reference Browser automatically alternates colored rows with white rows.

## To customize the colors displayed in the Cross Reference Browser

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Cross Reference Browser Settings, then click Grid Settings.
3. Customize the required options, then click OK.

- To change the colors applied to grid rows, alternate rows, selected reference rows, selected symbol rows, or other items, click the respective option, then select a color from the drop-down combobox.


## See also

Specify Block Library settings on page 77
Specify Trusted settings on page 77
Specify the default Sections Template and the Word ${ }^{\circledR}$ options for the generated documentation. The selected Sections Template modifies the items listed in the Sections pane of the Documentation Generator. The Word options modifies the different aspects of the print output including page orientation, paper format, margins, Word template, scaling of diagrams, link type, and comment style.

## To specify the Documentation Generator options

1. From the Tools menu, click Options...
2. In the Options dialog box, do these:
a. Expand Document Generator, click General, then select the required default Sections Template.

## b. Click Word and specify the required settings.

3. Click OK.

## Setting Grid Options

## Arrays View

## To access the grid options

From the Tools menu, click Options. In the Options dialog box, expand Grid Settings, then click the required grid type.

Customize the colors that display in various grids. Access the grid options for the following grids:

- Arrays View
- Defined Words View
- Dictionary View
- Parameters Grid
- Structures View
- Variable Groups View
- Variable Selector Settings


## See also

Arrays View on page 79
Defined Words View on page 80
Dictionary View on page 80
Parameters Grid on page 81
Structures View on page 81
Customize colors that display in the Arrays grid including column headers, row headers, and rows. The Arrays grid automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the color to display. Also define the color for the filter, available row, and the color to indicate disabled rows. For the top left corner of the grid, define the color to display. Choose whether to display the filter bar in the Arrays grid.

## To customize the display colors in the Arrays grid

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Arrays.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid,
click the respective option, and then select a color from the dropdown combo box.


## See also

Structures View on page 81
Dictionary View on page 80
Defined Words View on page 80

## Defined Words View

## Dictionary View

Customize the colors that display in the Defined Words grid including column headers, row headers, and rows. The Defined Words grid automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors displayed. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, define the color displayed. Choose whether to display the filter bar in the Defined Words grid.

## To customize the colors displayed in the Defined Words grid

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Defined Words.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.


## See also

## Arrays View on page 79

## Structures View on page 81

Dictionary View on page 80
Customize the colors that display in Dictionary instances including column headers, row header, and rows. The Dictionary grid automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors displayed. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, define the color displayed. Choose whether to display the filter bar in the Dictionary.

## To customize the colors displayed in the Dictionary

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Dictionary.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.


## See also

## Arrays View on page 79

## Structures View on page 81

Defined Words View on page 80

## Parameters Grid

## Structures View

Customize the colors that display in Parameters grid including column headers, row headers, and rows. The Parameters grid automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors displayed. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, define the color displayed. Choose whether to display the filter bar in the Parameters grid.

## To customize the colors displayed in the Parameters grid

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Parameters.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.


## See also

## Dictionary View on page 80

Customize the colors that display in Structures grid including column headers, row headers, and rows. The Structures grid automatically alternates colored
rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors displayed. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, Define the color displayed. Choose whether to display the filter bar in the Structures grid.

## To customize the colors displayed in the Structures grid

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Structures.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.


## See also

Arrays View on page 79
Defined Words View on page 80
Dictionary View on page 80

# Variable Groups View 

Customize the colors displayed in the Variable Groups view including column headers, row headers, and rows. The Variable Groups view automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors displayed. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, define the color displayed. Choose whether to display the filter bar in the Variable Groups view.

## To customize the colors displayed in the Variable Groups view

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Variable Groups.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.


## See also

## Dictionary View on page 80

Customize the colors that display in Variable Selector including column headers and rows. The Variable Selector automatically alternates colored rows with white rows. Adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, define the colors that display. Also define the color used for the filter and available row, and the color used to indicate disabled rows. For the top left corner of the grid, define the color that displays. Choose whether to display the filter bar in the Variable Selector. Also specify whether the Variable Selector opens displaying the local or global (controller) variables tab.

## To customize the colors displayed in the Variable Selector

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Grid Settings, and then click Variable Selector.
3. Customize the required options, and then click OK.

- To specify the number of consecutive rows for the alternating sequence, for Consecutive Rows, indicate the required value.
- To change the colors applied to headers, alternate rows, disabled rows, the filter/available row, and the top left corner of the grid, click the respective option, and then select a color from the dropdown combo box.
- To specify the tab displayed when the Variable Selector opens, for Default Selection, select Local or Global.


## See also

Dictionary View on page 80
Specify settings for Trusted ${ }^{\circledR}$ I/O wiring.

- Always keep devices expanded, specifies whether devices are expanded to display information such as the slot order, number of channels, data type, and description
- Show empty device slots, specifies whether empty device slots are displayed when viewing I/O wiring
- Show full device names, specifies whether I/O devices are displayed with their full names beside the slot number
- Prompt on device removal, specifies whether to prompt users before removing devices


## To specify the Trusted I/O wiring settings

1. From the Tools menu, click Options.
2. In the Options dialog box, expand I/O Wiring, and then click General.
3. Specify the settings as required, and then click OK.

## See also

1/O Wiring on page 638

## Setting IEC Language Options

## Function Block Diagram Settings

Customize the display settings for programs built in different IEC languages:

- Function Block Diagram Settings
- Ladder Diagram Settings
- Sequential Function Chart Settings
- Structured text Settings


## See also

## Function Block Diagram Settings on page 84

Ladder Diagram Settings on page 87
Sequential Function Chart Settings on page 88
Structured Text Settings on page 90
Customize the display settings for Function Block Diagram (FBD) diagrams.

- Choose the comment position for variables.
- Choose whether to display instance names for function blocks.
- Define the colors used to display FBD elements and text and define the variable information that displays in FBD diagrams.
- Choose the width for FBD elements in the language container.
- Choose whether to display grid lines inside FBD language containers.
- Choose to display links as arrows, present solid, dashed, dotted, dashed-dotted, dashed-dotted-dotted, or custom line styles, and apply a normal, rounded, or rounded with jump line types.

Options for customization are:

| Option | Description |
| :--- | :--- |
| Block Style | The function and function block background color. Possible <br> colors are custom, web, and system colors. |
| Background Color | The function and function block background gradient color. <br> Possible colors are custom, web, and system colors. |
| Background Gradient Color | The width for a function or function block, in number of grid <br> cells. |
| Cell Width | The indication of whether to display instance names for <br> function blocks. |
| Display Instance Names | While in design mode, enables going to the definition on double <br> click. |
| In design mode, go to definition on double |  |
| click | The level of transparency. The possible values range from 0 to <br> 255 where 0 indicates complete transparency. |
| Transparency |  |

[^1]| Option | Description |
| :---: | :---: |
| Block Style |  |
| Background Color | The comment background color. The possible colors are custom, web, and system colors. |
| Constant Style |  |
| Background Color - Events | The constant background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color - Events | The constant background gradient color. Possible colors are custom, web, and system colors. |
| Cell Width | The width for a constant, in number of grid cells. |
| Comment Position | The position of the comment in reference to the constant shape. Possible positions are none, top, bottom, left, and right. |
| Transparency | The level of transparency. Possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information to display for variables. Possible values are name, alias, name and alias, or name and wiring. |
| Container Settings |  |
| Auto Resize Elements when Modifying | When modifying, automatically resize blocks and variables to accommodate length of text. |
| Automatically Invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |
| Display Grid | Determines whether to display the grid in the language container. |
| Event Link Style |  |
| Arrow | Determines whether to display an arrow at the end of the link. |
| Line Style | The style of the line. Possible values are solid, dash, dot, dashdot, dash-dot-dot, and custom. |
| Line Type | The type of line. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over. |
| Link Event Color | The color of event links. Possible colors are custom, web, and system colors. |
| Jump |  |
| Cell Width | The width for a jump, in number of grid cells. |
| Label |  |
| Cell Width | The width for a label, in number of grid cells. |
| Left Power Rail |  |
| Background Color | The left power rail background color. Possible colors are custom, web, and system colors. |
| Background Gradient Color | The left power rail background gradient color. Possible colors are custom, web, and system colors. |
| Link Style |  |
| Arrow | Determines whether to display an arrow at the end of the link. |
| Line Style | The style of the line. Possible values are solid, dash, dot, dashdot, dash-dot-dot, and custom. |
| Line Type | The type of line. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over. |
| Link Color | The color of links. The possible colors are custom, web, and system colors. |


| Option | Description |
| :--- | :--- |
| Block Style |  |

Online Settings

| Online Text Color - Variable and Block | The color of text that displays the value of the variables in <br> online mode and simulation mode. |
| :--- | :--- |
| Operator Style | The operator background color. Possible colors are custom, <br> web, and system colors. |
| Background Color | The operator background gradient color. Possible colors are <br> custom, web, and system colors. |
| Background Gradient Color | The width for an operator, in number of grid cells. |
| Cell Width | Determines whether to display instance names for operators. |
| Display Instance Names | While in design mode, enables going to the definition on double <br> click. |
| In design mode, go to definition on double |  |
| click | The level of transparency. Possible values range from 0 to 255 <br> where 0 indicates complete transparency. |
| Transparency | The region background color. Possible colors are custom, web, <br> and system colors. |
| Region Style | The header color of a region. Possible colors are custom, web, <br> and system colors. |
| Background Color | The level of transparency of the header section of a region. The <br> possible values range from 0 <br> complete transparency. |
| Header Color where 0 indicates |  |


| Right Power Rail |  |
| :--- | :--- |
| Background Color | The right power rail background color. Possible colors are <br> custom, web, and system colors. |
| Background Gradient Color | The right power rail background gradient color. Possible colors <br> are custom, web, and system colors. |
| Variable Style | The variable background color. Possible colors are custom, <br> web, and system colors. |
| Background Color | The variable background gradient color. Possible colors are <br> custom, web, and system colors. |
| Background Gradient Color | The width for a variable, in number of grid cells. |
| Cell Width | The position of the comment in reference to the variable shape. <br> Possible positions are none, top, bottom, left, and right. |
| Comment Position | The level of transparency. Possible values range from 0 to 255 <br> where 0 indicates complete transparency. |
| Transparency | The information that displays for variables. Possible values are <br> name, alias, name and alias, or name and wiring. |
| Variable Information |  |

## To customize the display settings for FBD diagrams

1. From the Tools menu, click Options.
2. From the Options dialog box, expand IEC Languages, then click Function Block Diagram (FBD).

Using the available options, customize the required settings, then click OK.

## See also

## Reset Container and Shape Settings in FBD POUs on page 34

Ladder Diagram Settings on page 87

## Sequential Function Chart Settings on page 88

## Structured Text Settings on page 90

## Ladder Diagram Settings

Customize the display settings for Ladder diagrams.

- Choose to display grids, images, and instance names.
- Define the colors used to display LD elements and text and define which variable information displays in LD diagrams.
- Choose the width and height for LD elements in the language container.

Options for customization are:

| Block Settings |  |
| :---: | :---: |
| Display Image | The indication of whether to display block images. |
| Display Instance Names | The indication of whether to display instance names for function blocks. |
| Enable EN/ENO | Forces EN and ENO parameters onto all operators, functions, and function blocks. |
| Function Blocks Background Color | The function block background color. The possible colors are custom, web, and system colors. |
| Function Blocks Background Gradient Color | The function block background gradient color. The possible colors are custom, web, and system colors. |
| Functions Background Color | The function background gradient color. The possible colors are custom, web, and system colors. |
| Functions Background Gradient Color | The function background gradient color. The possible colors are custom, web, and system colors. |
| Go to Definition on Double-click | While in design mode, enables going to the definition on double click. |
| Operators Background Color | The operator background color. The possible colors are custom, web, and system colors. |
| Operators Background Gradient Color | The operator background gradient color. The possible colors are custom, web, and system colors. |
| Container Settings |  |
| Cell Height | The height of individual cells making up the grid, in pixels. |
| Cell Width | The width of individual cells making up the grid, in pixels. |
| Comment text font | The font type for comment text. The definition includes the font name, size, unit of measure, CGI Character Set, GDI Vertical Font, and the indication of whether to apply bold, italic, strikeout, and underline styles. |
| Display Grid | The indication of whether to display the grid. |
| Element Height | The height of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. |
| Element Width | The width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. |
| Font | The type of font. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDICharSet and GDIVerticalFont properties are not editable. |
| Rung Line Thickness | The thickness of the rung line. The possible values range from 1.0 to 3.0. |
| Editor Settings |  |
| Automatically Invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |
| Rung Settings |  |
| Coil Alignment | Indicates whether to align all coils on the rightmost section of the rung. |
| Comment Background Color | The comment background color. The possible colors are custom, web, and system colors. |
| Comment text color | The text color for comments. The possible colors are custom, web, and system colors. |
| Comment text font | The font type for comment text. The definition includes the font name, size, unit of measure, CGI Character Set, GDI Vertical Font, and the indication of whether to apply bold, italic, strikeout, and underline styles. |
| Display Comment | The indication of whether to display comments for rungs. |


| Block Settings |  |
| :--- | :--- |
| Display Label | The indication of whether to display labels for rungs. When not displaying labels, an arrow appears in the leftmost <br> section of the rung indicating the existence of a label. |
| Label Color | The color for rung labels. The possible colors are custom, web, and system colors. |
| Power Flow False Color | The color displayed when power flow monitoring is false. The possible colors are custom, web, and system colors. |
| Power Flow True Color | The color displayed when power flow monitoring is true.The possible colors are custom, web, and system colors. |
| Power Rail Color | The color for power rails. The possible colors are custom, web, and system colors. |
| Rung Header Color | The color for rung headers. The possible colors are custom, web, and system colors. |
| Variables Settings |  |
| Text Color | The color of text displayed while in design mode. The possible colors are custom, web, and system colors. |
| Text Color - Online | The color of text displayed while running online. The possible colors are custom, web, and system colors. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Background Color | The variable background color. The possible colors are custom, web, and system colors. |
| Variable Background Gradient Color | The variable background gradient color. The possible colors are custom, web, and system colors. |
| Variable Information | The indication of whether to display the variable name only, alias only, name and alias, or name and wiring. |

## To customize the display settings for LD diagrams

1. From the Tools menu, click Options.
2. From the Options dialog box, expand IEC Languages, and then click Ladder Diagram (LD).

Using the available options, customize the required settings, and then click ОК.

## See also

Reset Visual Settings in LD POUs on page 35

## Sequential Function Chart Settings

Customize the display settings for SFC diagrams. Choose the orientation of the pane splitting when displaying SFC diagram and actions/conditions programming simultaneously in the language container. Choose to display grids and sequence control types and diagram background and grid colors for design and online modes. For action blocks, jumps, and transitions, define the background, gradient, and font colors and the font style. For steps, define the active and inactive step and step gradient colors, the font color and style and the action list and list gradient colors.

Options for customization are:

| Setting | Description |
| :--- | :--- |
| Action Block Settings | The background color of action blocks. The possible colors are custom, <br> web, and system colors. |
| Action Block Color | The font definition used for the text displayed in an action block. The <br> definition includes the font name, size, unit of measure, and the <br> indication of whether to apply bold, italic, strikeout, and underline styles. <br> The GDI Character Set and GDI Vertical Font properties are not editable. |
| Action Block Font | The color of the font for action blocks. The possible colors are custom, <br> web, and system colors. |
| Action Block Font Color |  |


| Setting | Description |
| :---: | :---: |
| Action Block Gradient Color | The background gradient color of action blocks. The possible colors are custom, web, and system colors. |
| Container Settings |  |
| Background Color - Design | The background color for SFC diagrams while in design mode. The possible colors are custom, web, and system colors. |
| Background Color - Online | The background color for SFC diagrams while online. The possible colors are custom, web, and system colors. |
| Container Split Orientation | Controls the orientation for the splitting of the container between the SFC diagram and Actions/Conditions views. The possible values are vertical or horizontal. |
| Display Grid | The indication of whether to display the grid. |
| Display Sequence Control Type | The indication of whether to display the sequence controls type. |
| Display Transition Priority | The indication of whether to display the transition priority. |
| Grid Color - Design | The color of the grid while in design mode. The possible colors are custom, web, and system colors. |
| Grid Color - Online | The color of the grid while running online. The possible colors are custom, web, and system colors. |
| Jump Settings |  |
| Jump Color | The background color of jumps. The possible colors are custom, web, and system colors. |
| Jump Font | The font definition used for the text displayed in a jump. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Jump Font Color | The color of the font for jumps. The possible colors are custom, web, and system colors. |
| Jump Gradient Color | The background gradient color of jumps. The possible colors are custom, web, and system colors. |
| Macro Call Settings |  |
| Macro Call Color | Not supported |
| Macro Call Font | Not supported |
| Macro Call Font Color | Not supported |
| Macro Call Gradient Color | Not supported |
| Step Settings |  |
| Action List Color | The background color of action lists. The possible colors are custom, web, and system colors. |
| Action List Gradient Color | The background gradient color of action lists. The possible colors are custom, web, and system colors. |
| Step Color | The background color of steps. The possible colors are custom, web, and system colors. |
| Step Color - Active | The background color of active steps while online. The possible colors are custom, web, and system colors. |
| Step Font | The font definition used for the text displayed in a step. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Step Font Color | The color of the font for steps. The possible colors are custom, web, and system colors. |
| Step Gradient Color | The background gradient color of steps. The possible colors are custom, web, and system colors. |
| Step Gradient Color - Active | The background gradient color of active steps while online. The possible colors are custom, web, and system colors. |
| Transition Settings |  |
| Transition Color | The background color of transitions. The possible colors are custom, web, and system colors. |


| Setting | Description |
| :--- | :--- |
| Transition Font | The font definition used for the text displayed in a transition. The <br> definition includes the font name, size, unit of measure, and the <br> indication of whether to apply bold, italic, strikeout, and underline styles. <br> The GDI Character Set and GDI Vertical Font properties are not editable. |
| Transition Font Color | The color of the font for transitions. The possible colors are custom, <br> web, and system colors. |
| Transition Gradient Color | The background gradient color of transitions. The possible colors are <br> custom, web, and system colors. |

## To customize the display setting for SFC diagrams

1. From the Tools menu, click Options.
2. From the Options dialog box, expand IEC Languages, then click Sequential Function Chart.
3. Using the available options, customize the required settings, then click OK.

## See also

## Function Block Diagram Settings on page 84

Ladder Diagram Settings on page 87

## Structured Text Settings on page 90

## Structured Text Settings

Define the default display setting for Structured Text (ST) elements and text that displays in ST language containers. Choose the font used when displaying comments, editor text, identifiers, numbers, operators, POUs, punctuation, reserved words, and strings. Choose to display these in bold, italic, strikethrough, or underlined text and define their text color and size.

Options available for customization are:

| Option | Description |
| :--- | :--- |
| Comment | The font definition used for comment text. The definition includes <br> the font name, size, unit of measure, and the indication of whether <br> to apply bold, italic, strikeout, and underline styles. The GDI <br> Character Set and GDI Vertical Font properties are not editable. |
| Comment Font <br> Comment text Color <br> Editor <br> Editor Font <br> web, and system colors. |  |
| Editor Text Area Background Color | The font definition used for the ST editor. The definition includes <br> the font name, size, unit of measure, and the indication of whether <br> to apply bold, italic, strikeout, and underline styles. The GDI <br> Character Set and GDI Vertical Font properties are not editable. |
| Identifier | The color of the ST editor background. Possible colors are custom, <br> web, and system colors. |
| Identifier Font | The font definition used for identifiers. The definition includes the <br> font name, size, unit of measure, and the indication of whether to <br> apply bold, italic, strikeout, and underline styles. The GDI Character |
| Set and GDI Vertical Font properties are not editable. |  |


| Option | Description |
| :---: | :---: |
| Comment |  |
| Identifier Text Color | The color of the font for identifiers. Possible colors are custom, web, and system colors. |
| Number |  |
| Number Font | The font definition used for numbers. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Number Text Color | The color of the font for numbers. The possible colors are custom, web, and system colors. |
| Operator |  |
| Operator Font | The font definition used for operators. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Operator Text Color | The color of the font for operators. The possible colors are custom, web, and system colors. |
| POU |  |
| POU Font | The font definition used for POUs. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| POU Text Color | The color of the font for POUs. Possible colors are custom, web, and system colors. |
| Punctuation |  |
| Punctuation Font | The font definition used for punctuation. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Punctuation Text Color | The color of the font for punctuation. Possible colors are custom, web, and system colors. |
| Reserved Word |  |
| Reserved Word Font | The font definition used for reserved words. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Reserved Word Text Color | The color of the font for reserved words. Possible colors are custom, web, and system colors. |
| String |  |
| String Font | The font definition used for strings. The definition includes the font name, size, unit of measure, and the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| String Text Color | The color of the font for strings. Possible colors are custom, web, and system colors. |

## To customize the display settings for ST programs

1. From the Tools menu, click Options.
2. From the Options dialog box, expand IEC Languages, then click Structured Text (ST).
3. Expand the respective category, customize the required settings, then click OK.

## See also

Function Block Diagram Settings on page 84
Ladder Diagram Settings on page 87
Sequential Function Chart Settings on page 88

# Setting ISaVIEW Options 

## ISaVIEW Animation Settings

Use the animation settings to customize the animation settings for ISaVIEW screens including action, displacement, rotation, and size. Also define the refresh rate of ISaVIEW screens and their default background color.

## To customize the animation settings for ISaVIEW screens

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, and then click ISaVIEW Animation Settings.
3. Customize the required settings, and then click OK.

## See also

ISaVIEW Edition Settings on page 92
ISaVIEW Objects Settings on page 93
ISaVIEW Edition Settings
Use the edition settings to define the default settings for ISaVIEW screens and generic object properties.

## To define the default edition settings for ISaVIEW screens and objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, and then click ISaVIEW Edition Settings.
3. Define default display settings, and then click OK.

## See also

ISaVIEW Animation Settings on page 92

## ISaVIEW Objects Settings on page 93

## ISaVIEW Objects Settings

## Arc Settings

Use the object settings to specify default values for the individual object properties and grouping properties.

- Arc Settings
- Arrow Settings
- Bar Meter Settings
- Button Settings
- Edit Box Settings
- Ellipse Settings
- Gauge Settings
- Group Settings
- Image Settings
- Line Settings
- Polygon Settings
- Rectangle Settings
- Rounded Rectangle Settings
- Slider Settings
- Triangle Settings
- Web Container Settings


## See also

Button Settings on page 94
Group Settings on page 96
Image Settings on page 96
Line Settings on page 97
Web Container Settings on page 99
Use the arc settings to specify default values for the individual object properties.

## To specify the default settings for arc objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Arc Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Arrow Settings on page 94
Bar Meter Settings on page 94
Button Settings on page 94
Edit Box Settings on page 95

## Ellipse Settings on page 95

## Arrow Settings

## Bar Meter Settings

## Button Settings

Use the arrow settings to specify default values for the individual object properties.

## To specify the default settings for arrow objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Arrow Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Arc Settings on page 93
Bar Meter Settings on page 94
Button Settings on page 94
Edit Box Settings on page 95
Ellipse Settings on page 95
Use the bar meter settings to specify default values for the individual object properties.

## To specify the default settings for bar meter objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Bar Meter Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Button Settings on page 94
Edit Box Settings on page 95
Ellipse Settings on page 95
Gauge Settings on page 96
Group Settings on page 96
Use the button settings to specify default values for the individual object properties.

## To specify the default settings for button objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Button Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Edit Box Settings on page 95
Ellipse Settings on page 95
Gauge Settings on page 96
Group Settings on page 96
Image Settings on page 96

## Edit Box Settings

Ellipse Settings

Use the edit box settings to specify default values for the individual object properties.

## To specify the default settings for edit box objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Edit Box Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Ellipse Settings on page 95
Gauge Settings on page 96
Group Settings on page 96
Image Settings on page 96
Line Settings on page 97
Use the ellipse settings to specify default values for the individual object properties.

## To specify the default settings for ellipse objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Ellipse Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Gauge Settings on page 96

## Gauge Settings

Group Settings

Image Settings

Use the gauge settings to specify default values for the individual object properties.

## To specify the default settings for gauge objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Gauge Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Group Settings on page 96
Image Settings on page 96
Line Settings on page 97
Polygon Settings on page 97
Rectangle Settings on page 98
Use the group settings to specify default values for the individual object properties.

## To specify the default settings for grouped objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Group Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Image Settings on page 96
Line Settings on page 97
Polygon Settings on page 97
Rectangle Settings on page 98
Rounded Rectangle Settings on page 98
Use the image settings to specify default values for the individual object properties.

## To specify the default settings for image objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Image Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Line Settings on page 97
Polygon Settings on page 97
Rectangle Settings on page 98
Rounded Rectangle Settings on page 98
Slider Settings on page 98
Use the line settings to specify default values for the individual object properties.

## To specify the default settings for line objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Line Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Polygon Settings on page 97
Rectangle Settings on page 98
Rounded Rectangle Settings on page 98
Slider Settings on page 98
Triangle Settings on page 99
Use the polygon settings to specify default values for the individual object properties.

## To specify the default settings for polygon objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Polygon Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Rectangle Settings on page 98
Rounded Rectangle Settings on page 98
Slider Settings on page 98
Triangle Settings on page 99
Web Container Settings on page 99

## Rectangle Settings

Use the rectangle settings to specify default values for the individual object properties.

## To specify the default settings for rectangle objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Rectangle Settings.
3. Specify the default values for the required properties, then click OK.

## See also <br> Rounded Rectangle Settings on page 98 <br> Slider Settings on page 98 <br> Triangle Settings on page 99 <br> Web Container Settings on page 99

Rounded Rectangle Settings
Use the rounded rectangle settings to specify default values for the individual object properties.

## To specify the default settings for rounded rectangle objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Rounded Rectangle Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Slider Settings on page 98
Triangle Settings on page 99
Web Container Settings on page 99
Use the slider settings to specify default values for the individual object properties.

## To specify the default settings for slider objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Slider Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Triangle Settings on page 99
Web Container Settings on page 99
Use the triangle settings to specify default values for the individual object properties.

## To specify the default settings for triangle objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Triangle Settings.
3. Specify the default values for the required properties, then click OK.

## See also

Web Container Settings on page 99
Arc Settings on page 93
Arrow Settings on page 94
Use the web container settings to specify default values for the individual object properties.

## To specify the default settings for web container objects

1. From the Tools menu, click Options.
2. From the Options dialog box, expand ISaVIEW Settings, then expand ISaVIEW Object Settings, and then click Web Container Settings.
3. Specify the default values for the required properties, then click OK.

## See also <br> Arc Settings on page 93 <br> Arrow Settings on page 94 <br> Image Settings on page 96 <br> Line Settings on page 97

## Offline Grid Settings

Online Grid Settings

Customize the offline and online behavior options and look and feel of spy lists. Available behavior options are:

- Offline grid settings
- Online grid settings


## To customize spy lists

1. From the Tools menu, click Options.
2. From the Options dialog box, expand Spy List Settings, and perform this:
a. Click Offline Grid Settings and make the required changes.
b. Click Online Grid Settings and make the required changes.
3. Click OK.

## See also

Offline Grid Settings on page 100
Online Grid Settings on page 100
Customize offline behavior options and look and feel of spy lists. Available behavior options are:

- Filter Row, displays a row below the column heading to filter items in the list.
- Grouping Drop Area, displays an area at the top of spy lists to group items in a list by column types.
- Indent Sub-items, indents sub-items of arrays, structures, and function blocks.
- Item Count Rows, displays rows with the item count for complete spy lists and individual arrays, structures, and function block instances.
- Look and Feel options, the available options to customize the colors used for the headers, various rows, and borders as well the text colors.
- Default Monitoring Rate, the default rate to refresh spy list variables, in milliseconds. Also modify the refresh rate of individual spy lists. Modify refresh rates while offline.


## See also

Online Grid Settings on page 100
Defining Spy List Settings on page 100
Customize online behavior options and look and feel of spy lists. Available behavior options are:

- Filter Row, displays a row below the column heading to filter items in the list
- Grouping Drop Area, displays an area at the top of spy lists to group items in a list by column types
- Indent Sub-items, indents sub-items of arrays, structures, and function blocks
- Item Count Rows, displays rows with the item count for complete spy lists and individual arrays, structures, and function block instances
- Look and Feel options, the available options to customize the colors used for the headers, various rows, and borders as well the text colors.


## See also

Offline Grid Settings on page 100
Defining Spy List Settings on page 100

## Description Window

## To access the Description window

- From the View menu, select Description Window.

Use the Description Window to add descriptions to projects, program organization units (POUs), and AADvance controllers. These descriptions are free-formatted text using rich text format (RTF). All content is automatically saved. When editing descriptions, a text editor toolbar provides the means for performing basic formatting operations like selecting a font, size, style, and color.

The Description Window is dockable and scalable. The contents of the Description Window automatically displays the description for items selected in the Application View.

Tip: While in debug mode, the content in the Description Window is read-only.

## See also

Application View on page 24
Arrange and Dock Windows on page 58
Customize Toolbars on page 62

# Working with AADvance Applications 

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software is the development environment for the AADvance controller enabling the creation of one comprehensive project to control individual AADvance controllers. The AADvance-Trusted SIS Workstation software enables the creation of AADvance applications supporting multi-process control. Applications consist of virtual machines running on hardware components. The development process consists of creating a project made up of controllers, representing individual hardware, which are downloaded to the physical controllers. At runtime, the virtual machines run on this hardware equipment.

Develop projects with these programming languages from the IEC 61131-3 standard: Functional Block Diagram (FBD), Ladder Diagram (LD), Structured Text (ST), and Sequential Function Chart (SFC). Other than SFC, all produced code is validated for use in safety applications. AADvance-Trusted SIS Workstation software does not validate SFC code. When building, controllers are compiled to produce very fast "target independent code" (TIC).

Declare variables using standard IEC 61131-3 data types (Boolean, integer, real, etc.) or user-defined types such as arrays or structures.

Develop projects on a Windows development platform. The AADvanceTrusted SIS Workstation software graphically represents and organizes controllers, program organization units (POUs), and networks within a project from many views.

- Dictionary
- Equipment view
- Communication view

Libraries enable defining functions, function blocks, and data types for reuse throughout projects.

The AADvance-Trusted SIS Workstation software downloads individual controllers onto hardware equipment with SNCP networks.

Simulate the running of a project, after building a project, with high-level debugging tools, before actually downloading controllers to hardware equipment. Then connect the AADvance-Trusted SIS Workstation software to monitor real-time information and perform updates. A standard Ethernet network connects the AADvance-Trusted SIS Workstation software to controllers and controllers to other controllers.

## See also

Creating an AADvance Project on page 104

Dictionary on page 160
Equipment View on page 231
Communication View on page 175

## Creating an AADvance Project

Create AADvance projects from the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software. A project consists of hardware platforms called controllers. A project can contain one or more program organization units (POUs). POUs are programs, functions, or function blocks. POUs are linked together in a treelike architecture and can be described using any of Sequential Function Chart (SFC), Structured Text (ST), Function Block Diagram (FBD), or Ladder Diagram (LD) programming languages.

Tip: $\quad$ Regularly save changes made to projects and libraries. Before performing modifications, save the project to prevent the loss of modified communication information.

Templates available for AADvance projects:

- Import AADvance Project, enables importing an AADvance project.
- Restore Project from an Archive, enables restoring an AADvance project that was previously archived from a repository in AADvanceTrusted SIS Workstation software version 1.0 or AADvance ${ }^{\circledR}$ Workbench 2.1.
- Create Empty Series 9000 Project, enables creating an AADvance project containing one controller.

To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required for AADvance projects, right-click the SIS Workstation Windows desktop shortcut and select Run as administrator.

Set these properties for projects:

| Property | Description |
| :--- | :--- |
| Comment | Text displayed next to the project name |
| Description | Free-form text describing a project |
| Name (Read-only) | Name of the project. Project names are limited to 32 characters <br> beginning with a letter or single underscore followed by letters, digits, <br> and single underscores. The last character for a project name must <br> be a letter or digit; project names cannot end with an underscore <br> character. Reserved words, defined words, or data types such as, <br> elementary, structures, or arrays are not valid names. Use unique <br> names for projects within a directory. |
| Password Protected (Read-only) | Indication that the project is protected by a password controlling its <br> access |
| Path (Read-only) | Complete path where the project files are stored on the computer. <br> The path is automatically assigned: <br> \%USERPROFILE\% IDocumentsISIS Workstation <br> 1.OPProjectsIProjectName |
| Modify the storage location for projects in the Projects options from |  |
| the Tools > Options menu. Set a local path for the project location; |  |
| network paths may cause unexpected results. |  |

Perform these tasks for projects:

| Task | Procedure |
| :--- | :--- |
| Create an AADvance project | 1. From the File menu, click New Project... (or press Ctrl+Shift+N). <br> 2. In the New Project dialog box, in the AADvance tab, click Series 9000 in the Project Types list. <br> 3. In the Templates list, click the Create Empty Series 9000 Project template. <br> 4. Specify a name and location for the project, then click OK. |
| Set properties for a project | - In the Application View, right-click the project, and then click Properties. |
| Import target definitions | Tip: Do not import target definitions. |
| Build a project | - In the Application View, right-click the project element, and then click Build All (or press Ctrl+Shift+B). |
| Clean a project | - In the Application View, right-click the project to clean, and then click Clean All. |
| Download project code | Tip: Avoid downloading an application containing zero POUs or empty POU <br>  <br>  <br>  <br>  <br> Build the project code. <br> - From the Application View, right-click the project element, and then click Download. |

Control access for projects by setting a password restricting modifications like adding controllers, programs, functions, and function blocks.

## See also

Import an AADvance Project on page 105
Archiving and Restoring Repository Projects on page 153
AADvance Libraries on page 118
Setting Project Access Control on page 121
Controllers on page 108

## Import an AADvance Project

Import existing projects created using any version of the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software or AADvance ${ }^{\circledR}$ Workbench.

Important: $\quad$ To import a project with its library using AADvance-Trusted SIS Workstation software, see Import an AADvance Project with Library on page 106.

## To import an AADvance project

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required for AADvance projects, right-click the SIS Workstation Windows desktop shortcut and then select Run as administrator.
2. From the File menu, click New Project.....
3. In the New Project dialog box, in the AADvance tab, click Import in the Project Types list.
4. In the Templates list, click Import AADvance Project.
5. Specify a name for the project, and then click Browse to select the database path.
6. In the Select Database File dialog box, locate and select the project database file (*.mdb or *. accdb), and then click Open.
7. A message may display asking to update the database to the current version. Note that after updating the database, the project can no longer be opened in a previous version of AADvance Workbench. To continue the importation process, click OK.
8. In the New Project dialog box, click OK. The imported AADvance project is available for use in AADvance-Trusted SIS Workstation software.

## See also

Creating an AADvance Project on page 104
Closing a Project on page 120

## Import an AADvance Project with Library

Import existing projects with their library created using any version of the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software or AADvance ${ }^{\circledR}$ Workbench.

## To import an AADvance project with its library

1. Before importing the original project, rename the library folder that contains the MDB file used by this project.
2. Import the project. For more information refer to Import an AADvance Project on page 105.

The dialog box appears.
a. If you are warned there is a Project Missing Files, select Yes.
b. If you are asked to update the database, select Ok.
3. Select the project in the Application View and select File > Version Control > Check Out.

The Check-Out Confirmation dialog box appears.
4. Select OK.
5. Add a New Library. For more information refer to AADvance Libraries on page 118.

The Security Password dialog box may appear. If it does, enter the password of the library again and select OK.
6. Select the project in the Application View and select File > Version Control > Check In.

The Check-In Confirmation dialog box appears.
7. Select OK.
8. Select FILE> Save All.

When importing an AADvance ${ }^{\circledR}$ Workbench version 1.1 or version 1.2 project, upgrade the Series 9000 target definition.

$$
\begin{array}{ll}
\text { Important: } & \begin{array}{l}
\text { When using AADvance }{ }^{\oplus} \text { Workbench software to import a project with its } \\
\text { library, and if the Series } 9000 \text { target definition is not up-to-date, see }
\end{array} \\
& \text { Importing an AADvance Project with Library with outdated Series } 9000
\end{array}
$$

## To upgrade the Series $\mathbf{9 0 0 0}$ target definition

1. Import an AADvance Workbench version 1.1 or version 1.2 project.

A subfolder for the imported project is created in the AADvance projects folder on your hard drive. The default AADvance projects folder path is \%USERPROFILE\%\Documents\SIS Workstation <version>\Projects.
2. Exit the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software.
3. Delete the subfolder for the imported project in the AADvance projects folder on your hard drive.
4. Open the project from AADvance-Trusted SIS Workstation software.

## See also

## Import an AADvance Project on page 105

File System Paths on page 146
Opening an AADvance Project on page 120

Import an AADvance Project with Library with outdated Series 9000 Target Definition

If existing projects and their libraries were created using version 1.1 or 1.2 of AADvance Workbench use the following procedure to import the projects and their libraries.

## To import an AADvance project with its library with outdated Series 9000 target definition

1. Before importing the original project into AADvance ${ }^{\circledR}$ Workbench version 2.1, rename the library folder that contains the MDB file used by this project.
2. Import the project. For more information refer to Import an AADvance Project on page 105.
The AADvance ${ }^{\circledR}$ Workbench dialog box appears.
3. If you are warned there is a Project Missing Files, select Yes.
4. Select the project in the Application View and select File > Version Control > Check In.
The Check-in Confirmation dialog box appears.
5. Select OK.
6. Import the project library, by using the Add New Library functionality. For more information refer to AADvance Libraries on page 118.
The Security Password dialog box may appear. If it does, enter the password of the library again and select OK.
7. Select FILE> Save All.
8. Select FILE> Close Project.
9. Import the now closed project. For more information refer to Import an AADvance Project on page 105.
10.Right-click the newly imported project in the Application View and select Version Control > Check Out.

The Check Out dialog box appears.
11.Enable Recursive.
12.Select OK.
13. Select the project in the Application View and select File > Version Control > Check In.
The Check-in Confirmation dialog box appears.

## 14.Select Yes.

15. Close and reopen the project.
16. Right-click the new project in the Application View and select Version Control > Check Out.

The Check Out dialog box appears.
Enable Recursive.
17. Select OK.
18. Select Check in all checked items in the Pending Checkins panel.

The Check-in Confirmation dialog box appears.

## 19.Select Yes.

20.Select PROJECT > Version Control > Archive.

The Archive <project> dialog box appears.
21.Select Latest Version from the Archive Mode dropdown.
22.Select Archive.

The Save As dialog box appears.
23.Save the latest version of the project.
24. Note: AADvance ${ }^{\circledR}$ Workbench version 2.1 is no longer required for the following steps.
25.In AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software, select FILE> New Project.
The New Project dialog box appears.
26. Select Restore from Project Types.
27. Browse the Archive Path to the newly created VSC file.
28. Select OK.

The project is restored.

## Controllers

A controller corresponds to a programmable logic controller. Add controllers to existing projects.

Specify these properties for controllers:

| Property | Description |
| :--- | :--- |
| Code | Produce code for simulation for an application |
| Code For Simulation |  |


| Property | Description |
| :---: | :---: |
| Compiler Options | Each controller has defined compiler options. The Code Generator uses these parameters to to build and optimize the controller code. In the Compilation Options for a controller, select the type of code to generate according to corresponding equipment and set up the optimizer parameters according to the expected compilation and run-time requirements. The general compiler options are: <br> - Check Array Index - (Read-only) Verify array indices <br> - Dump Configuration Files - Generate controller level files containing debugging information and place them at the root of the controller folder. Names the files using the controller name as a prefix with .ttc and .tws as extensions. <br> - Dump Network - Generate network and controller level files containing debugging information. Places the files at the root of the network folder and at the root of the controller folder. The files in the network folder are named "NetworkConf" and have the extensions .ttc and .tws. The files in the controller folder are named using the controller name as a prefix and have .ttc and .tws as extensions. <br> - Dump POU Files - Generate controller-level files containing debugging information and place these at root of the controller folder. Names the files using the controller name as a prefix, the program organization unit (POU) name as a suffix, and the extensions .ttc and .tws. Names other files using the POU name with .Ist and .unc as extensions. <br> - Enable Compiler Verification - Validate the compilation process for all POUs of a controller. Comparison errors display in the Output window. <br> - Function Internal State Enable - Produce internal state information for functions. Functions containing no internal state information denote that the invocation of a function with the same arguments always yields the same values. <br> - Generate Map File - Generate controller-level files containing debugging information. Places the files at the root of the controller folder and names these using the controller name as a prefix with .ttc, .tws, and .map as extensions. |
| TIC Code | (Read-only) Compiler produces TIC code |
| General |  |
| Battery Alarm | For the 9110 processor module, enables the battery alarm informing of a low voltage for the battery and need for replacement. For standard and mixed controllers, the default value is enabled. For Eurocard controllers, the default value is disabled. |
| Remote Fault Reset | Resets faulty processors from a remote location and joins processors to a running system. To enable, enter a non-zero hexadecimal value matching the value of the control integer Allow Remote Fault Reset. Possible values range from $0 \times 1$ to $0 x F F F F F F F F$, or $0 \times 0$ to disable. |
| Hardware |  |
| Memory IO Structures | Memory space allocated for I/0 structures. The default value is 150000 bytes. |
| Memory Size | Space reserved for monitoring variables, constants, and temporary compiler variables |
| Target | (Read-only) Target type to which is attached the controller |
| Update I/O Device Online | For targets supporting online changes, enables modifying I/O devices and structures while running an application online. <br> After an update, the module is configured but not connected. Press Fault Reset on the AADvance controller or perform a remote fault reset from AADvance-Trusted SIS Workstation software to connect the I/O module to the running system. The faults disappear and the connection is successful. |
| Info |  |
| Comment | Single line of text with a maximum of 255 characters |
| Compilation Version | (Read-only) Compilation version number |
| Description | Free-form text describing a controller |


| Property | Description |
| :---: | :---: |
| Full Name | (Read-only) Full name of the controller indicating the project to which it belongs |
| Last Compilation Date | (Read-only) Date of the last compilation of the controller code |
| Name | (Read-only) Name of the controller. Controller are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a controller name must be a letter or digit; controller names cannot end with an underscore character. Reserved words, defined words, or data types such as, elementary, structures, or arrays are not valid names. The same type of elements within a scope must have unique names. |
| Number | Unique number identifying the controller within the project. Also known as the resource number in AADvance Discover. This number is automatically assigned. When changing the number, assign a value that is unique within the project. The resource number identifies the physical AADvance controller that runs the application code. |
| Password Protected | (Read-only) Controller is protected by a password controlling its access |
| Path | (Read-only) Complete path where the controller files are stored on the computer. The path is automatically assigned to a controller folder within the project folder. |
| Memory Size for Online Changes |  |
| Code Size | Defines the amount of memory that is reserved for all updates on the controller. The Code Size value can be increased up to 1441640 bytes. The total usage of the controller memory space is the sum of the Code Size value and the length of the controller name, project name, and POU names. <br> For example, controller memory space usage can consist of these values: <br> - Project name with 8 characters <br> - Controller name with 11 characters <br> - POU name with 5 characters <br> - Code Size value of up to 1441640 bytes <br> Or <br> - Project name with 32 characters <br> - Controller name with 24 characters <br> - POU name with 21 characters <br> - Code Size value of up to 1441384 bytes |
| Maximum Extra POUs | Maximum number of POUs (other than SFC) that can be added while performing updates |
| SFC States Mem Size | The memory space allocated for step and transition structures. A step requires 40 bytes and a transition requires 20 bytes. |
| User Variable Size | For updates, the amount of memory reserved for adding variables data. When generating monitoring symbols information for a POU, the same amount of memory is also reserved for the POU. |
| Safety |  |
| Process Safety Time | The maximum time, in milliseconds, that the controller outputs remain in the ON state after detecting important diagnostic or application faults. The outputs go into their specified safe states when exceeds the defined PST. Set the PST for the controller. The default value for the controller is 2500 ms . By default, each group of I/O modules inherits the PST from the controller. Also set an alternate PST for each I/O group. |
| Settings |  |
| Cycle Time | Amount of time given to each cycle. If a cycle is completed within the cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the POUs of the controller, then updating physical outputs. The virtual machine executes the controller code according to the execution rules. |


| Property | Description |
| :---: | :---: |
| Cycle Time Units | Unit of measure for the cycle time. Possible values are ms (milliseconds) or $\mu \mathrm{s}$ (microseconds). To use $\mu \mathrm{s}$, the equipment must support this unit of measure. |
| Detect Errors | Store errors. Define Nb Stored Errors. |
| Memory for Retain | Location where retained values are stored (the required syntax depends on the implementation). <br> Tip: Do not edit the Memory for Retain property. |
| Nb Stored Errors | Number of entries. For example, the size of the queue (FIFO) in which detected errors are stored |
| Start Simulation Mode | Controller executes in real time or cycle-to-cycle. Real-time mode is the run time normal execution mode where the cycle timing triggers the controller cycles. In cycle-to-cycle mode, the virtual machine loads the controller code but does not execute it until execute one cycle or activate real-time mode. When generating debug information for POUs, the controller automatically switches to step-by-step mode when the application encounters a breakpoint. |
| Trigger Cycles | Controller cycle executes according to the defined Cycle Time |
| SFC Dynamic Behavior Limits |  |
| Gain Factor | For Sequential Function Chart (SFC), specifies factor of dynamic behavior limits determining the amount of memory, allocated by a controller at initialization time, designated to manage token moving. The amount of allocated memory is calculated as a linear relation with the number of SFC POUs: <br> Alloc Mem (bytes) = N * NbElmt * sizeof(typVa) <br> NbElmt = GainFactor * NbOfSFC + OffsetFactor |
| Offset Factor | For SFC, specifies factor of dynamic behavior limits determining the amount of memory, allocated by a controller at initialization time, designated to manage token moving. The amount of allocated memory is calculated as a linear relation with the number of SFC POUs: <br> Alloc Mem (bytes) = N * NbElmt * sizeof (typVa) <br> NbElmt = GainFactor * NbOfSFC + OffsetFactor |
| System |  |
| System Type | Type of the controller. Possible values are: <br> - Standard, (AADvance controller) configuring 48 empty IOB IO slots for I/O modules (IOB IO Bus 1 and IO Bus 2) <br> - Eurocard, configuring 18 empty slots on IOB Bus 1 <br> - Mixed, configuring 48 empty IOB IO slots for I/O modules (IOB Bus 1 and IO Bus 2). |
| Perform these tasks for controllers: |  |
|  |  |
| pplication View, right-click the project element, point to Add, and then select New Controller. |  |
| trollers as exchange files (*.pxf). |  |
| Application View, right-click the controller and then select Export.... <br> Save Controller As dialog box, browse for the location in which to store the exchange file, then select Save. |  |


| Task | Procedure |
| :---: | :---: |
| Add an existing controller from an exchange file | 1. In the Application View, right-click the Project element, point to Add, and then select Existing Controller.... <br> 2. In the Select Exchange File dialog box, locate the exchange file containing the controller, and then select Open. <br> 3. In the Add Existing Controller dialog box, select the element to import, and then select Import. <br> Tip: - The SNCP binding links for the controller are not imported. <br> - After importing a controller, select one element only in the Application View before performing a check-in. <br> - Avoid renaming program organization units (POUs) in the Import dialog box when importing. Renaming may cause a loss of function block references. Rename the POU from the Application View. <br> - Importing AADvance Workbench version 2.0 .pxf exchange files does not import CIP bindings and generates build errors in the Error List. Recreate CIP bindings in AADvance-Trusted SIS Workstation software or import a project from an .mdb file. |
| Set properties for a controller | - In the Application View, right-click the controller, and then select Properties. |
| Copy and paste a controller | 1. In the Application View, right-click the controller, and then select Copy. <br> 2. Right-click the project element and then select Paste. |
| Rename a controller | - In the Application View, right-click the controller, then select Rename, and then type a name for the controller. |
| Delete a controller | If a CIP consumer link is defined for a controller and the controller is deleted before adding CIP links to other controllers, close and then reopen AADvance-Trusted SIS Workstation software. <br> - In the Application View, right-click the controller, then select Delete. |

## See also

## Programs on page 112

Functions on page 113
Function Blocks on page 115
Download code to controllers on page 129
Cleaning a Project on page 127
Defining Bindings on page 200
Import an AADvance Project on page 105

## Programs

Define programs in the Programs section of a controller in the Application
View. Sequential programs must be adjacent within a Programs section.
Programs belonging to a same section must have different names.
Specify these properties for programs:

| Property | Description |
| :--- | :--- |
| Code Generation | Generate information required for debugging with step-by-step execution |
| Generate Debug Info |  |


| Property | Description |
| :--- | :--- |
| Generate Monitoring Symbols | For graphical program organization units (POUs), generate information to <br> graphically display the output values of elements for debugging or <br> simulating |
| Info | Single line of text with a maximum of 255 characters |
| Comment | Free-form text describing a program |
| Description | (Read-only) Full name of the program with the project and controller to <br> which it belongs |
| Full Name | (Read-only Programming language of the POU |
| Language | (Read-only) Name of the program. Program names are limited to 128 <br> characters beginning with a letter followed by letters, digits, and single <br> underscores. The last character for a program name must be a letter or <br> digit. Program names cannot end with an underscore character. Reserved <br> words, defined words, or data types such as elementary, structures, or <br> arrays are not valid names. The same type of elements within a scope <br> must have unique names. |
| Order | Position of the program within the execution order |
| Password Protected | (Read-only) Program is protected by a password controlling its access |
| Path | (Read-only) Complete path to the program files on the computer |

Perform these tasks for programs:

| Task | Instruction |
| :---: | :---: |
| Add a program | In the Application View, right-click the Programs element for a controller, point to Add Programs, and then select one of these: <br> - New ST: Structured Text <br> - New FBD: Function Block Diagram <br> - New LD: Ladder Diagram <br> - New SFC: Sequential Function Chart |
| Export a program | Export programs as exchange files (*.pxf). <br> 1. In the Application View, right-click the program, and then click Export.... <br> 2. In the Save <Program> As dialog box, browse for the location where to store the program exchange file, and then click Save. <br> Tip: If a user-defined structure contains user-defined structures, create at least one variable for each user-defined structure to support a successful POU export. |
| Add a program from an exchange file | Import programs previously exported as exchange files (*.pxf). <br> Tip: For POUs in version control, check out the POU before importing the exchange file (*.pxf). <br> In the Application View, right-click the Programs element, point to Add Programs, then Existing Program.... <br> 1. In the Select Exchange File dialog box, locate the exchange file containing the program, and then select Open. <br> 2. In the Add Existing Program dialog box, select the element to import, and then select Import. |
| Set properties for a program | In the Application View, right-click the program, and then select Properties. |
| Copy and paste a program | 1. In the Application View, right-click the program, and then select Copy. <br> 2. Right-click the Program element, and click Paste. |
| Rename a program | In the Application View, right-click the program, then select Rename, and then type a name for the program. |
| Delete a program | In the Application View, right-click the program, and then select Delete. |

## See also

## Controllers on page 108

Functions on page 113

## Function Blocks on page 115

## Functions

Define functions in the Functions section of a controller in the Application View. Program functions with Function Block Diagram (FBD), Ladder Diagram (LD), or Structured Text (ST).

For functions, specify these properties:

| Properties |  |
| :--- | :--- |
| Code Generation | Description |
| Generate Debug Info | Generate information required for debugging with step-by-step execution. |
| Info | Single line of text with a maximum of 255 characters |
| Comment | Free-form text describing a function |
| Description | (Read-only) Full name of the function with the project and controller to which <br> it belongs |
| Full Name | (Read-only) Programming language of the program organization unit (POU) |
| Language | (Read-only) Name of the function. Function names are limited to 128 <br> characters beginning with a letter followed by letters, digits, and single <br> underscores. The last character for a function name must be a letter or digit <br> function names cannot end with an underscore character. Reserved words, <br> defined words, or data types, such as elementary, structures, or arrays are <br> not valid names. The same type of elements within a scope must have <br> nuique names. |
| Order | Execution order of the POU |
| Password Protected | (Read-only) Password controls access to the function |
| Path | (Read-only) Complete path to the function files on the computer |

When adding functions, define required parameters. Functions can have a maximum of 128 parameters (inputs and outputs). When defining parameters, consider the following limitations:

- Parameter names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a parameter name must be a letter or digit; parameter names cannot end with an underscore character. Reserved words, defined words, or data types such as, elementary, structures, or arrays, are not valid names. Use unique names for the same type of elements within a scope.
- Possible data types for parameters are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, Function blocks
- For String type variables, string capacity is limited to 255 characters excluding the terminating null character ( 0 ), a byte for the current length of the string, and a byte for the maximum length of the string
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF
- For dimensions, example: [1..10] for a one dimensional array, [1..4,1..7] for a two dimensional array
Perform these tasks for functions:

| Task | Procedure |
| :--- | :--- |
| Add a function | 1. In the Application View, right-click the Functions element, point to Add Functions, and then select the programming language <br> for the function. |
|  | 2. To define the parameters for the function, right-click the function, and then select Parameters. |


| Task | Procedure |
| :---: | :---: |
| Export a function | Export functions as exchange files (*. ${ }^{*}$.xff). Also export functions from libraries. <br> 1. In the Application View, right-click the function, and then select Export.... <br> 2. In the Save <Function> As dialog box, browse for the location in which to store the function exchange file, and then select <br> Save. <br> Tip: If a user-defined structure contains user-defined structures, create at least one variable for each user-defined structure to support a successful POU export. |
| Add a function from an exchange file | Import functions previously saved as exchange files (*.pxf). <br> Tip: $\quad$ For POUs in version control, check out the POU before importing the exchange file (*.pxf). <br> In the Application View, right-click the Functions element, point to Add Functions, and then select Existing Function.... <br> 1. In the Select Exchange File dialog box, locate the exchange file containing the function, and then select Open. <br> 2. In the Add Existing Function dialog box, select the element to import, and then select Import. |
| Set properties for a function | In the Application View, right-click the function, and then select Properties. |
| Copy and paste a function | 1. In the Application View, right-click the function, and then select Copy. 2. Right-click the Function element, and then select Paste. |
| Rename a function | In the Application View, right-click the function, then select Rename, and then type a name for the function. |
| Delete a function | In the Application View, right-click the function, and then select Delete. |

## See also

Programs on page 112
Function Blocks on page 115
Variables on page 117
Verifying POU Syntax on page 126
Define function blocks in the Function Blocks section of the Application View. Program functions using Function Block Diagram (FBD), Ladder Diagram (LD), Sequential Function Chart (SFC), or Structured Text (ST).

Specify these properties for function blocks:

| Property | Description |
| :--- | :--- |
| Code Generation | Generate information required for debugging using step-by-step <br> execution. |
| Generate Debug Info | For graphical program organization units (POUs), generate information <br> required for graphically displaying the output values of elements when <br> debugging or simulating |
| Generate Monitoring Symbol | Size of memory reserved for each function block instance for adding <br> symbols monitoring information during updates. A string-type output <br> takes up 260 bytes. |
| Instance Symbols Extra Bytes | Single line of text with a maximum of 255 characters |
| Info | Free-form text describing a function block |
| Comment | (Read-only) Full name of the function indicating the project and controller |
| to which it belongs |  |


| Property | Description |
| :--- | :--- |
| Name | Read-only) Name of the function block. Function block names are limited <br> to 128 characters beginning with a letter followed by letters, digits, and <br> single underscores. The last character for a function block name must be <br> a letter or digit; function block names cannot end with an underscore <br> character. Reserved words, defined words, or data types, such as <br> elementary, structures, or arrays are not valid names. The same type of <br> elements within a scope must have unique names. |
| Order | Position of the function block within the execution order |
| Password Protected | (Read-only) Function is protected by a password controlling its access |
| Path | (Read-only) Complete path where the function block files are stored on the <br> computer |

When adding function blocks, define required parameters. Function blocks can have a maximum of 128 parameters (inputs and outputs). When defining parameters, consider these limitations:

- Parameter names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a parameter name must be a letter or digit; parameter names cannot end with an underscore character. Reserved words, defined words, or data types, such as elementary, structures, or arrays are not valid names. The same type of elements within a scope must have unique names.
- Possible data types for parameters are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, Function blocks
- For String type variables, string capacity is limited to 255 characters excluding the terminating null character ( 0 ), a byte for the current length of the string, and a byte for the maximum length of the string
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF
- For dimensions, example: [1..10] for a one dimensional array, [1..4,1..7] for a two dimensional array

For instances of function blocks, reset the initial values defined for individual instances.

Perform these tasks for function blocks:

| Task | Procedure |
| :--- | :--- |
| Add a function block | 1. In the Application View, right-click the Function Blocks element, point to Add Function Blocks, and then select the <br> programming language for the function block. <br> 2. To define the parameters for the function block, right-click the function block, and then select Parameters. |
| Export a function block | Export function blocks as exchange files (**.pxf). Also export functions from libraries. <br> 1. In the Application View, right-click the function block, and then select Export.... <br> 2. In the Save <Function Block> As dialog box, browse for the location in which to store the function block exchange file, and <br> then select Save. <br> Tip: If a user-defined structure contains user-defined structures, create at least <br> one variable for each user-defined structure to support a successful Pou <br> export. |


| Task | Procedure |
| :--- | :--- |
| Add a function block from an <br> exchange file | Import function blocks previously saved as exchange files (*.pxf). For program organization units (POUs) in version control, <br> check out the POU before importing the exchange file (*.pxf). <br> Avoid renaming POUs in the Import dialog box when importing. Renaming may cause a loss of function block references or <br> broken binding links for local variables. Rename the POU from the Application View. <br> 1. In the Application View, right-click the Function Blocks element, point to Add Function Blocks, and then select Existing <br> Function Block.... |
| 2. In the Select Exchange File dialog box, locate the exchange file containing the function, and then select Open. |  |
| 3. In the Add Existing Function Block dialog box, select the element to import, and then select Import. |  |

## See also

Programs on page 112

## Functions on page 113

Verifying POU Syntax on page 126
Variables
Variables are defined for their scope. For instance, controller variables are global, for example, available for use throughout the programs, functions, and functions blocks of a controller. Whereas, variables defined for a program, a function, or a function block are local to that element. Define variables in the Variables grid of the Dictionary. For controller variables, create groups then add existing controller variables. Variables can belong to multiple groups. For individual variable scopes, import and export variables data having the Microsoft Excel (*.xls) or comma-separated values (*.csv) format.

Variable names are limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for a variable name must be a letter or digit; variable names cannot end with an underscore character. Reserved words, defined words, or data types such as, elementary, structures, or arrays, are not valid names. The same type of elements within a scope must have unique names.

When defining complex variables like arrays and structures, the syntax for the variable name is:

- For arrays: arrayname[index]

Name, Alias, Data
Type, StringSize, InitValue, Direction, Wiring, Attribute
-••
array1, , BOOL, 0, , ...
"array1[1,1]", , BOOL, 0, , ...
"array1 [1, 2]", , BOOL, 0, , ...
"array1[1,3]", , BOOL, 0, , ...
"array1 [1, 4]", , BOOL, 0, , , ..
"array1 [1,5]", ,BOOL,0, , , ..

- For structures: structurename.membername

Name, Data
Type, Dimension, Alias, Comment, InitValue, Direction ...

```
structure1,,T9K_DI_FULL,0,, ...
structure1.DI,,BOOL,0,, ...
structure1.LF,,BOOL,0,, ...
structure1.DIS,,BOOL,0,, ...
structure1.CF,,BOOL,0,, ...
structure1.V,,UINT,0,, ...
structure1.STA,,USINT,0,, ...
```

When managing variables data, create groups for controller variables data.
When adding controller variables to a group, add these to the group from the variables grid or drag these between the variables grid and the group grid.

## To create a controller variable group

1. In the Application View, expand the Variables item, right-click Groups, and then click Add New Variable Group.

The group is added.
2. To add a single variable to the group:

- Open the controller variables grid, right-click the left most column for the required variable, point to Add to Group, and then click the required group.

3. To add multiple variables to the group by dragging:
a. Open the variables grid for the group by double-clicking and place both grids side-by-side.
b. In the controller variables grid, select the consecutive variables, then click and drag from the cell having the arrow in the left most column to within the variables group grid.

## See also

## Programs on page 112

## Function Blocks on page 115

Functions on page 113

## Variable Dependencies on page 171

Import and Export Variables Data on page 123

# AADvance Libraries 

Libraries consist of functions, function blocks, and data types including arrays, structures, and defined words for reuse throughout AADvance projects. Projects can depend on more than one library. A library cannot use functions, function blocks, and data types including arrays, structures, and defined words belonging to another library. Using the IEC 61131-3 languages, write functions (FBD, LD, or ST) and function blocks (FBD, LD, ST, or SFC).

## Tip: Regularly save changes made to libraries.

Add libraries from different sources to projects:

| Task |  |
| :--- | :--- |
| Add an AADvance library | Create a library based on the AADvance template. <br> 1. In the Application View, right-click the Libraries item, point to AddEnter a library name and description, and then <br> click to locate the MS in the Library Typestn the New Library dialog box, click AADvance Library in the Library <br> Types list, and then click the Create Empty Library template. <br> 2. Enter a library name and description, and then click OK. |
| Add a library from a repository | Add a library from a repository. Libraries previously published are available from a version control repository. <br> 1. In the Application View, right-click the Libraries item, point to Add, and then click Published Library.... <br> 2. In the Add Published Library dialog box, browse for the repository containing the published library, indicate whether <br> to only show the latest versions of the published libraries, and then select the required library from the list. |
| 3. Click OK. |  |

Libraries are stored within the projects where these are created. When a library becomes unresolved, restore the reference to the library.

Library functions, function blocks, and data types must have unique names within a project, including other libraries used in the project. When these have the same names as those defined in a project in which these are used, only used in the project. When these have the same names as those defined in a project in which these are used, only those from the project are recognized.

Tip: Avoid using library data types.

Library functions and function blocks are compiled with the project. However, verify the syntax for library functions and function blocks.

> Tip: Before setting a password for a library, check in the library. Setting a password without checking in the library does not apply passwordprotection to the library.

Tasks to perform with libraries are:

| Task | Procedure |
| :---: | :---: |
| Copy and paste a function or function block between libraries | 1. In the Application View, right-click the function block in the source library, and then click Copy. <br> 2. Right-click the respective section of the destination library, and then click Paste. |
| Export a function or function block | 1. In the Application View, right-click the function or function block to export, and then click Export.... <br> 2. In the Save As dialog box, browse for the location in which to store the element exchange file, and then click Save. |
| Import a function or function block | - In the Application View, right-click the library, and then click Delete. |
| Drag a function or function block between libraries | - In the source library, drag the function or function block to the respective section in the destination library. <br> Tip: Do not drag and drop POUs between libraries and controllers. Use a PXF exchange file to add the library POU to the controller, and then delete the POU from the library. |
| Rename a library | - In the Application View, right-click the library, then click Rename, and then type a name for the library. |
| Delete a library | Avoid removing or renaming a library. Create a new project if you need to rename or delete a library, or create a library immediately after project creation. <br> Tip: Before deleting a library, ensure all POUs using the library are editable, then save and close the POUs. <br> In the Application View, right-click the library, and then click Delete. |

## See also

Function Blocks on page 115
Functions on page 113
Programs on page 112
Verifying POU Syntax on page 126
Check In Elements to Version Control on page 149

Opening an AADvance Project

## To open an AADvance Project

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required for AADvance projects, right-click the SIS Workstation Windows desktop shortcut and select Run as administrator.
2. From the File menu, click Open Project... (or press Ctrl+Shift+O).
3. In the Open Project dialog box, select the required project, and then click OK.

## See also

AADvance Libraries on page 118
Creating an AADvance Project on page 104
Closing a Project on page 120
When closing a project, the AADvance-Trusted SIS Workstation software prompts to save changes made to the project.

## To close a project

1. From the File menu, click Close Project.
2. When prompted to save changes, click Yes.

## See also

## Opening a Trusted Project on page 532

## Setting Project Access Control

For project security, set access control using a password for projects, controllers, program organization units (POUs), libraries, and library functions and function blocks. Password definitions are limited to eight characters and can consist of letters and digits. Enter the password to open a passwordprotected project. Project sub-elements, can have their own level of access control. Example: a POU with its own password remains locked and cannot be modified without entering its password.

Since POUs are encrypted, password definitions must be retained.
In the Application View, the following indicates the security state for elements:

品 Indicates that a lock is applied to the element
When opening a project with password-protected elements, there is a prompt to enter the password once for each element. Password-protected elements have these modification restrictions:

| Password-Protected Element | Modification Restrictions |
| :--- | :--- |
| Project | Opening the project |
| Controller | Adding, editing, and deleting a program, modifying the <br> communication protocols, modifying the system type and <br> controller properties, wiring variables, and adding, editing, and <br> deleting I/O modules. |
| Program | Viewing the program |
| Library | Adding, editing, and deleting a library function or function block |
| Library Function | Viewing the function |
| Library Function Block | Viewing the function block |
| Edit existing passwords for projects and project sub-elements. Also remove |  |
| existing passwords. When copying, pasting, importing, and exporting |  |
| elements with access control, password definitions are retained. |  |


| Task | Procedure |
| :--- | :--- |
| Set a password | 1. In the Application View, right-click the required element, and then click Set Password. <br> 2. In the Set Password dialog box, enter the required information, then click OK. <br> a. In the Password field, type the required password. <br> b. In the Confirm Password field, re-type the required password. <br> Before setting a password for a library, check in the library. Setting a <br> password without checking in the library does not apply password- <br> protection to the library. |
| Edit a password | 1. In the Application View, right-click the required element, and then click Password. <br> 2. In the Set Password dialog box, enter the required information, then click OK. <br> a. In the Old Password field, type the current password. <br> b. In the Password field, type the required password. <br> c. In the Confirm Password field, re-type the required password. |


| Task | Procedure |
| :--- | :--- |
| Remove a password | 1. In the Application View, right-click the required element, and then click Set Password. <br> 2. In the Change Password dialog box, enter the required information, then click OK. <br> $\bullet$ •In the Old Password field, type the current password. <br> $\bullet$ - The New Password and Confirm Password fields must remain blank. |

## See also

Opening an AADvance Project on page 120
Setting Target Access Control on page 122
Check In Elements to Version Control on page 149

## Setting Target Access Control

For controller security, set access control by defining a password for the target AADvance controller. Password definitions are limited to eight characters and can consist of letters, digits, and symbols. Target access control prevents the connection of all IXL clients not having the password for the target controller. Users having the password can attach the target to controllers in different projects.

Tip: The password definitions for controller targets are saved on target systems.

Targets with passwords are protected from:

- Stopping the application from the AADvance ${ }^{\circledR}-$ Trusted $^{\circledR}$ SIS Workstation software
- Downloading an application
- Updating an application
- Locking a variable
- Modifying the value of a variable

Existing passwords for targets can be edited. Existing passwords for target controllers can be removed. When setting, editing, and deleting the password for a controller target, the attached target must be running.

| Task | Procedure |
| :--- | :--- |
| Set a password for a target | 1. In the Application View, right-click the controller instance, and then click Set Target Password. <br> 2. In the Set Target Password dialog box, enter the required information: <br> a. In the New Password field, type the required password. <br> b. In the Confirm Password field, re-type the required password. <br> 3. Click OK. |
| Edit a password for a target | 1. In the Application View, right-click the required controller, and then click Set Target Password. <br> 2. In the Set Password dialog box, enter the required information, then click OK. <br> a. In the Old Password field, type the current password. |
|  | b. In the New Password field, type the required password. <br> c. In the Confirm Password field, re-type the required password. |
| Remove a password for a target | 1. In the Application View, right-click the required element, and then click Set Password. <br> 2. In the Change Password dialog box, enter the required information, then click OK. <br> - In the Old Password field, type the current password. <br> - The New Password and Confirm Password fields must remain blank. |

## See also

## Setting Project Access Control on page 121

Import and Export Variables Data

Save imported variables as Microsoft ${ }^{\circledR}$ Excel ${ }^{\circledR}$ spreadsheets (.xls) or commaseparated values (.csv).

- Export variables to manage simple and complex variables data including adding, removing, and modifying variables.
- Import variables data files into controllers and programs in the same project or in other projects.
Tips: - AADvance ${ }^{\oplus}$-Trusted ${ }^{\oplus}$ SIS Workstation software uses the default values for missing data when importing a variables data file missing any columns.
- Make sure imported variables match their corresponding definitions in the project when importing variables.

Specify a location to save exported files when exporting variables. Export an empty file as a template to develop the contents of variables data files (*.xls or *.csv) in a respective editor like Microsoft Excel or Notepad.

A variables data file includes a header row, a mapping row, a version number, and the variables data. The header row displays the names of the data columns. The column names of the header row are the various variable properties. The mapping row displays the internal names of data columns in brackets used for processing. An automatically generated version number indicates the version of the import/export feature. The individual variable data is placed in the respective columns.


This table shows the syntax used in the variables data files for the variable properties and the associated internal names:

| Variable Property | Internal Name | Description |
| :--- | :--- | :--- |
| Name | (Name) | Name of the variable. Variable name are limited to 128 characters beginning with a letter or single <br> underscore character followed by letters, digits, and single underscore characters. The last character <br> for a variable name must be a letter or digit; variable names cannot end with an underscore character. <br> Names cannot be reserved words, defined words, or data types (elementary, structures, or arrays). <br> Names must be unique. |
| Data Type | (DataType) | Data type of the variable |
| Dimension | (Dimension) | Number of elements defined for an array |
| String Size | (StringSize) | The maximum character length for string-type variables. |
| Initial Value | (InitialValue) | (Direction) |
| Direction | Forue theld by a variable when the virtual machine starts executing the controller <br> an input, or an output. For controller variables, indicates whether the variable is internal, an input, an <br> output, a system variable, or an I/O channel. |  |
| Attribute | (Attribute) | Read and write access rights |
| Comment | User-defined free-format text for variables and array elements. Each array element of the same type <br> can have a different comment. |  |

Chapter 2 Working with AADvance Applications

| Variable Property | Internal Name | Description |
| :---: | :---: | :---: |
| Alias | (Alias) | Any name |
| Wiring | (Wiring) | The I/0 channel wired to the variable <br> During variable import, the raw and engineering values of analog I/O channels reset: <br> When imported variables are not wired, the raw values are: 0 for low, and 1 for high. The engineering values are: 0 for low, and 1 for high. <br> When imported variables are wired, the raw values are: 1024 for low, and 5120 for high. The engineering values are: 0 for low, and 100 for high. |
| Address | (UserAddress) | User-defined address of the variable |
| Retained | (IsRetained) | The virtual machine saves the value of the variable at each cycle. Limitations: <br> - Do not use the Retained property for safety-related settings. <br> - Do not set the Retained property on wired T9K_xx_FULL structures. |
| Retained Flags | (RetainFlags) | Enables retaining specific elements of a variable and indicates whether to use the initial value of a variable or the value previously retained on the target. |
| Groups | (Groups) | Variable group containing the variables listed in alphabetical order |
| OPC Write | (AllowOPCWrite) | An external client can write to the variable |
| Message True | (MsgTrue) | Message defined for the TRUE value message |
| Message False | (MsgFalse) | Message defined for the FALSE value message |
| Base Address | (ModBusBaseAddress) | For Modbus communications, the base address of the variable |
| Type | (ModBusType) | For Modbus communications, indicates one of these variable types: <br> - When the data type is BOOL and the type is True, the variable type is Coils. <br> - When the data type is BOOL and the type is False, the variable type is Discrete Inputs. <br> - When the data type is DINT and the type is True, the variable type is Holding Registers. <br> - When the data type is DINT and the type is False, the variable type is Input Registers. |
| Write Protected | (ModBusWriteProtected) | For Modbus Communications for Coils and Holding Registers, indicates whether data can be copied from the Modbus master to the slave device. |
| Reference Variable | (SOERefVar) | For the SOE service, the referenced variable of any elementary data type. |
| Falling Edge Level | (SOEFallingLevel) | Te SOE service detects a fall from TRUE to FALSE. |
| Rising Edge Level | (SOERisingLevel) | The SOE service detects a rise from FALSE to TRUE. |
| Filter Time | (SOEFilterTime) | For the SOE service, the minimum time lapse between two events. |
| CIP Kind | (CIPKind) | For CIP communications, indicates whether the variable is a producer or consumer. |
| Producer Name | (CIPProducerName) | When the CIP variable is a consumer, indicates the CIP producer from which the consumer receives data. |
| Path To Producer | (CIPCPathToProducer) | When the CIP variable is a consumer, indicates the path to the producer. |
| Consumer Remote Tag | (CIPRemoteTag) | When the CIP variable is a consumer, indicates the remote tag name of the producer variable communicating with the consumer variable. |
| RPI (ms) | (CIPCRPI) | When the CIP variable is a consumer, indicates the frequency in milliseconds that the remote controller offers the variable to the AADvance controller. |
| Producer Remote Tag | (CIPPRemoteTag) | When the CIP variable is a producer, indicates the remote tag name of the consumer variable communicating with the producer variable. |
| Max Number of Connections | (CIPPMaxConnections) | When the CIP variable is a producer, the maximum number of simultaneous consumer variables. |
| To Controller | (SNCPToController) | For the SNCP producer binding variable, the corresponding controller name for each consumer variable. For multiple consumer variables, controller names are separated by a comma. For example: Controller2,Controller2,Controller3. |
| From Controller | (SNCPFromController) | For the SNCP consumer binding variable, the name of the controller producing the variable binding group. |
| From Symbol | (SNCPFromSymbol) | For the SNCP consumer binding variable, the name of the producer variable. |
| Error Value | (SNCPErrorValue) | For the SNCP consumer binding variable, the value defined for the error behavior. When no value is specified for the error behavior, the binding uses the last value. |
| Update Value | (SNCPUpdateValue) | For the SNCP consumer binding variable, the value defined for the update behavior. When no value is specified for the update behavior, the binding uses the last value. |

A progress bar shows advancement of import and export operations. Use the progress bar to cancel import and export operations.

- For import operations, the process stops after importing the last variable in progress.
- For export operations, the process does not produce an exported variables file.

| Task | Procedure |
| :--- | :--- |
| Import variables | 1. In the Application View, right-click the destination receiving the Local Variables or Controller Variables, and then click Import <br> Variables.... <br> 2. From the Import Variables dialog box, select the file type containing the variables, locate the file, and then click Open. |
| Export variables | 1. In the Application View, right-click the Local Variables or Controller Variables containing the variables to export, and then click <br> Export Variables.... <br> 2. From the Export Variables dialog box, specify a name and locate the destination in which to store the file containing the <br> exported variables, select the file type and then click Save. |

## Generating Code

## See also

Creating an AADvance Project on page 104
AADvance Libraries on page 118
Before downloading code onto a target systems, build the code for the whole project. This operation verifies and builds the code for all libraries, controllers, programs, functions, and function blocks including builds information used to recognize systems on networks. Once a project is built, subsequent build operations only recompile the parts of the project that need regeneration.

Choose to build the project, controllers, or libraries. Also verify the syntax of program organization units (POUs). Building or verifying a selected project element, only verifies or builds the code for the selected element.

Choose to clean projects and controllers. Cleaning projects and controllers deletes the intermediate and output files generated during the last build operation. However, after cleaning, updates cannot be performed. To retain the capacity to perform updates, perform subsequent builds for a project or controller instead of cleaning and then building it.

## See also

## Building an AADvance Project on page 125

## Verifying POU Syntax on page 126

## Cleaning a Project on page 127

Choose to generate code and the necessary files for a project. Also, choose to build only a selected element within a project such as a controller or library.
When building projects, controllers, or libraries, view the progress of the build operation in the Output window. When the build operation is complete, view generated errors in the Error List.

Tips: - Always perform a build operation after getting a specific version of a project.

- Verify all elements are connected before performing a build.


## Building an AADvance Project

| Build a project | In the Application View, right-click the project element, and then click Build All (or press Ctrl+Shift+B). |
| :--- | :--- |
| Build a selected element within a <br> project | 1. In the Application View, click the required controller or library. <br> 2. From the Build menu, click Build <Element>. <br> The build process is initiated for the selected element only. |
| View the build progress and <br> generated errors | 1. From the Tools menu, click Options. <br> a. In the Options dialog box, and then expand Projects. |
|  | b. Select General, and then select the following options: <br> - Always show Error List if build finishes with errors <br> - Show Output window when build starts |
| c. Select OK. |  |

The AADvance-Trusted SIS Workstation software rebuilds controllers and increases the build number in these conditions:

- The controller has been modified since the last compilation
- A dependent library is not compiled, was compiled after the controller, or was modified since the last compilation
- Contains a program organization unit (POU) needing compilation: a POU has been modified since the last compilation, a POU is an LD or FBD function block for which symbol monitoring is activated, and a POU has local array variables when array index verification is activated.
- The controller contains variables in binding consumer links or variables in CIP consumer links

The compiler generates different code for simulation than for targets. Specify the code for simulation in the controller properties before building.

Generating monitoring symbols information to graphically monitor the values of variables requires a significant amount of memory space. When compiling, an error message stating that the memory limit has been reached may display in the output window. To enable compiling, either disable monitoring for the POU, remove elements from the POU, or reduce the size of the POU.

## See also

Generating Code on page 125
Verifying POU Syntax on page 126
Cleaning a Project on page 127
Compiler Verification on page 300

## Verifying POU Syntax

Verify the syntax for programs, functions, and function blocks. Verifying the syntax only verifies the programming syntax without producing code. View verification progress in the Output window.

| Task | Procedure |
| :--- | :--- |
| Verify POU syntax | Before performing a build, ensure all elements are connected. <br> Tip: $\quad$ After getting a specific version of a project, clean and then build the <br> project. |
|  | In the Application View, right-click the program, function, or function block, and then click Verify <POU>. |


| Task | Procedure |
| :--- | :--- |
| View verification progress and | 1. From the Tools menu, click Options. |
| generated errors | 2. In the Options dialog box, expand Projects, click General, select these options, and then click OK. |
|  | - Always show Error List if build finishes with errors |
|  | - Show Output window when build starts |
|  | 1. Verify the required program, function, or function block. |

## See also

Building an AADvance Project on page 125
Generating Code on page 125

## Cleaning a Project

Delete intermediate and output files belonging to projects or controllers. Cleaning projects, for example deleting these files, enables the generation of new files during the next build operation. When cleaning individual controllers, the intermediate and output files are deleted for the controller only.

After cleaning a project, no updates for any controllers may be performed. After cleaning a controller, no updates for the controller can be performed. To retain the capacity to perform updates, perform a subsequent build rather than cleaning and building.

| Task | Procedure |
| :--- | :--- |
| Clean a project | • In the Application View, right-click the project to clean, and then click Clean All. |
| Clean a controller | • In the Application View, right-click the controller to clean, and then click Clean Controller. |

## See also

## Building an AADvance Project on page 125

## Run an Application Online

Running online means that an application is connected to physical controllers. The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software ensures the execution of the code and files with the version control label corresponding to the running version of the controller. While running online, modify cycle timings for controllers, force the values of variables, and monitor the values of variables.

Before running an application on physical controllers, build the project code and download the application code onto the target.

Important:

- Attempting to connect or simulate a project having a number of variables exceeding the available memory displays the "OPC server is unable to load project..." message and the "OutOfMemoryException: calling OPC Server shutdown" error message in the Output window. The connect or simulate operation ends and the project returns to design mode.
- If an "unable to connect" error occurs when trying to connect to a controller with a valid IP address, use AADvance Discover to confirm that the resource number specified in the AADvance project is correct.
- Connecting to multiple controllers after a library update uses the library content that matches the topmost selected controller from the Connect dialog box. The Connect In Progress dialog box may warn that some libraries may not match the running application for other controllers. To use library content that matches another controller, connect only to that controller.
Simulate the running of an application for debugging purposes,. The compiler generates different code for simulation than for online.


## To run an AADvance application online

1. From the Communication View, specify the IP addresses for the controllers in the project.
2. From the Properties window, specify the required Number for the controller. Each controller in a project must have a unique Number matching the Resource Number of the required AADvance controller. From the AADvance Discover tool, view and change the Resource Number of AADvance controllers.
3. To set the connection mode for the project, from the Project menu, point to Connect Mode, and then click Online.
The Online menu category appears with the online command options.
4. Build the project code.
5. Download the application code onto the physical controllers. A version control label is applied to the downloaded code and files.
6. Connect the controllers with the physical equipment.

The AADvance-Trusted SIS Workstation software saves the application state and gets the corresponding version control label for the running version of the controller. Disconnecting returns the controllers to the state previous to connecting.

## See also

Generating Code on page 125
Download code to controllers on page 129
Performing Updates on page 130
Connect with controllers on page 134
Simulate an application on page 137

## Download code to controllers

Download project code to the physical controllers. No download is required to simulate an application.

## Prerequisites:

- Connect the computer where the AADvance-Trusted SIS Workstation software is installed to the hardware controller through an SNCP network.
- Fit the program enable key into the AADvance ${ }^{\circledR}$ Control System controller.
- Set the Code for Simulation property to True for all controllers.
- Set the value of the SFC States Mem Size property to reserve sufficient space for SFC program organization units (POUs).
Tip: $\quad$ To change the resource number of a controller, clear the application from the AADvance Control System controller. Modify the resource number in AADvance Discover and in the AADvance Control System controller properties, then download the code to the AADvance Control System controller.

When downloading code, the AADvance-Trusted SIS Workstation software verifies that the latest build reflects the current project.

Tip: If unable to download an application to a controller, free memory by clearing the applications from the AADvance Control System controller.
When unable to connect to a AADvance Control System controller with a valid IP address:

- Cancel the update or download in progress and then retry the operation.
- Verify that the resource number specified in the AADvance project is correct using AADvance Discover.


## To download code to controllers

1. Build the project code.
2. Verify the program enable key is plugged into the key connector on the 9100 processor base unit.
3. From the Application View, right-click the project, and then click Download.
4. In the Download window, select the controllers, and then click Download.

The AADvance-Trusted SIS Workstation software checks in the controllers to download into the version source control repository, then downloads the code to the physical controllers. The AADvanceTrusted SIS Workstation software applies a label to the controller in the source control repository for each successful download.

## See also

Starting and Stopping Controllers on page 133
Connect with controllers on page 134

## Disconnect from Controllers on page 137

Performing Updates

Modify a controller while it runs. This is sometimes necessary for processes where any interruption may jeopardize production or safety. When performing builds with unsupported update modifications, warnings may display.

> | Important: $\quad \begin{array}{l}\text { When performing updates, ensure there is sufficient free memory space } \\ \\ \text { for the storage of modified or added items. Updates are denied when } \\ \\ \\ \text { memory space is insufficient. Specify the available memory size for } \\ \text { updates in the controller properties. }\end{array}$ |
| :--- | :--- |

Modify the I/O configuration of a controller. Modifications to the controller I/O must match the hardware configuration. To modify the controller I/O, the Update I/O Device Online controller property must be enabled. The default value for this property is False.

- Performing updates while the application is running can affect the Safety
Function of the system. Updates should only be performed when absolutely
required since the safety integrity of the system may be reduced during the
modification.
- Before modifying the I/O configuration and performing updates while the
controller is connected, set up alternative safety measures lasting the duration
of the update.
- Performing an update on a Safety Implemented System is the responsibility of
the user. The running application may stop if modifications are performed
incorrectly.

When modifying properties of controllers, perform a download for the changes to take effect. Also, the initial values of variables are applied upon starting controllers. Updates do not start controllers. An update is split into the three phases:

- Downloads the update to the required controllers
- Confirms the download is error free and verifies if the controller will accept the update
- Realizes the update

During an update, all bindings are closed and all consumed variables go to their defined Update behavior states. The controller updates and then attempts to re-establish the bindings. If the amount of time between closing and re-establishing the bindings exceeds the binding Update Timeout value, the consumed variables transition from their Update behavior values to their Error behavior values.

> Tip: If AADvance-Trusted SIS Workstation software is unable to connect to the AADvance Control System controller, cancel the update in progress and then retry the operation.

For all AADvance-Trusted SIS Workstation software versions, the update limitations are:

- Declared or user-defined arrays and structures cannot be modified. Declared arrays and structures are defined as data types.
- Modifications to controller properties are not applied after an update.

Tasks available for the latest AADvance-Trusted SIS Workstation software version when performing updates are:

| Bindings | Adding, deleting, and editing. <br> Creating and deleting bindings between variables. <br> Changing the consume status variable and consumption behavior of a binding. <br> Changing the producing variable, consuming variable, or network for a binding <br> creates a new one. <br> Adjusting the update timeout period in the network parameters. The update timeout <br> period is the maximum time during which the consumer can remain in the update <br> state. |
| :--- | :--- |
| Internal variables | Adding, deleting, and editing internal variables. <br> When renaming or changing the data type of internal variables, the AADvance- <br> Trusted SIS Workstation software creates new variables. Therefore, variables are <br> initialized. <br> Changing the alias, initial value, group, scope, direction, retain setting, address, and <br> Comment of variables. When changing the initial value of a read-only internal value, <br> the AADvance-Trusted SIS Workstation software reinitializes the variable. When <br> Changing the scope of a variable, the AADvance-Trusted SIS Workstation software <br> Blocks Function <br> reinitializes the variable. <br> Modifying the length of string variables. When decreasing the length, the contents of <br> the string is truncated to the new length. <br> Switching a variable attribute between the input and output attribute. Variable <br> cannot be switched between the internal and input/output attribute. <br> Adding and removing elements in arrays for internal variables. For multi-dimensional <br> arrays, only add elements to the first dimension. The AADvance-Trusted SIS <br> Workstation software initializes these new elements. Adding elements to other <br> dimensions causes the AADvance-Trusted SIS Workstation software to initialize a <br> new array. <br> Renaming, adding variables to, removing variables from the group to which a |
| Adding, deleting, and moving function blocks. Adding and deleting function block |  |
| instances. |  |
| Renaming and modifying user-defined functions and function blocks. |  |
| qariable belongs or moving the variable to another group. |  |
| Adding, removing, and modifying the parameters of user-defined functions and |  |
| function blocks. When modifying the parameters, instance data is not preserved. |  |
| Recompile modified functions and function blocks called by other POUs and the |  |
| calling POUs. |  |
| Changing the wired variable. |  |
| For analog I/O channels, modifying the raw and engineering values to configure the |  |
| gain and offset settings. |  |
| For input channels, modifying the threshold values. |  |



The Update window contains the following columns:

| Name | The name of the controller |
| :--- | :--- |
| Comment | Free-format text |
| Local Version | The latest build version. |
| Controller Status | The status of the AADvance controller. Possible values are Running, <br> Stopped, Unable to connect, or Unrecognized application. <br> This column also displays the current version of the application running <br> on the AADvance controller. The Controller Status version must match the <br> Required Version. |
| Required Version | The version of the application downloaded or updated on the AADvance <br> controller. <br> If Download Required displays in the Required Version column, cancel the <br> update and perform a download operation. |

Tip: Updates are unavailable after getting a previous version from the repository; perform a download operation.

## To perform an update

Perform an update after building a project. Updates are unavailable after cleaning a project.

1. Make sure that the memory size properties for the required controllers indicate sufficient memory space.
2. Verify the program enable key is plugged into the key connector on the 9100 processor base unit.
3. From the Application View, right-click the project or controller for which to perform the update, and then click Update.
4. In the Update window, select the controller or controllers to update, and then click Update.

## See also

## Download code to controllers on page 129

Clearing an Application on page 134
Starting and Stopping Controllers on page 133
Connect with controllers on page 134
Disconnect from Controllers on page 137

## Starting and Stopping Controllers

Stop an AADvance controller while it runs. Also start a stopped AADvance controller. Start or stop individual controllers or all controllers in a project. Starting or stopping a controller is only available when the controller is disconnected. For individual controllers, view the version information and the current status and any comments.


## To start or stop controllers

1. Disconnect from the running controllers.

Tip: $\quad$ Starting or stopping a controller that has restricted access, that is, when another user is debugging a project in the controller, does not execute the start or stop command.
2. From the Project menu, click Start/Stop.

The Start/Stop Application dialog box displays.
3. In the Start/Stop Application dialog box, set one of the following actions for each controller:

- None
- Start
- Stop

4. Click Apply. A warning displays to confirm the actions. Click OK.

The applications running on the AADvance controllers stop or start.

## See also

Download code to controllers on page 129
Connect with controllers on page 134

## Disconnect from Controllers on page 137

## Clearing an Application

## Connect with controllers

Previously downloaded applications may be cleared from a physical controller. For a project, choose to clear either individual controllers or all controllers. Applications may only be cleared from disconnected controllers. Replace an application in an AADvance controller by clearing the controller and then downloading a new application.

When clearing applications, specify the individual controllers from those existing in a project, then specify the controllers to clear applications. For individual controllers, view version information for the local and physical controller applications, status of the controller, and any comments.

## Tip: When clearing an application, only the files for the current resource number are deleted. It is recommended to clear the application before changing resource number.

## To clear an application

1. Disconnect from the running controllers.
2. In the Application View, select the project, then from the Project menu, click Clear.
3. In the Clear Application dialog box, perform one of the following:

- To clear applications for individual controllers, select the required controllers, and then click Clear.
- To clear applications for all controllers, click Select All, and then click Clear.

The application files are removed from the required physical controllers. Now download a different application with the same IP address to the physical controller.

## See also

Download code to controllers on page 129
Starting and Stopping Controllers on page 133
Disconnect from Controllers on page 137
Connect with the physical controllers when developing an application. A virtual machine executes the code for each controller in real-time on the equipment.

## Tip: Breakpoints, cycle-to-cycle, and step-by-step execution are only available while simulating an application.

Connect individual controllers within a project. Use the Connect dialog box to select individual or all controllers. When connected, perform tasks that include:

- Monitor program execution and the values of variables
- Lock and unlock I/O channels of an I/O module
- Modify values of variables
- Modify cycle timing

Important: • When connected, do not modify any controllers. Modifications may be lost after disconnecting.

- Connecting to multiple controllers after a library update uses the library content that matches the topmost selected controller from the Connect dialog box. The Connect In Progress dialog box may warn that some libraries may not match the running application for other controllers. To use library content that matches another controller, connect only to that controller.

When connected, a local or remote AADvance-Trusted SIS Workstation software locks a controller from other users while performing any operation modifying the controller. Only one AADvance-Trusted SIS Workstation software can lock a controller at any given time. Operations that lock a controller include performing downloads and updates, modifying the values of variables and I/Os, modifying cycle timing, and more. The AADvance-Trusted SIS Workstation software unlocks access after disconnecting from the controller or after completing a download. When the AADvance-Trusted SIS Workstation software locks a controller, the identity is indicated as the user name from the computer. If the AADvance-Trusted SIS Workstation software is unable to restrict access to the controller, the operation modifying the controller fails.

Tip: Check in all pending changes before connecting.
Status information for a controller appears as a lightning bolt icon in the Application View, Communication View, and Equipment View:

| Icon | Description |
| :--- | :--- |
| 8 | Yellow. Controller is unlocked and available to any AADvance-Trusted SIS Workstation software. |
| 8 | Green. Controller is locked by a local AADvance-Trusted SIS Workstation software and unavailable <br> from other AADvance-Trusted SIS Workstation software. |
| 8 | Red. Controller is locked by a remote AADvance-Trusted SIS Workstation software and unavailable to <br> local AADvance-Trusted SIS Workstation software. |
| 8 | Grey. Controller is not connected or is in simulation mode. |

From these views, obtain detailed status information for controllers including the identity of an AADvance-Trusted SIS Workstation software locking a controller by mousing over the controller status icons. The Controller Status displays the identities of AADvance-Trusted SIS Workstation software locking controllers.

| Important: | Before initially connecting to controllers, build the project, then download the code to <br> the controller. Subsequent connections automatically extract the controller code from <br> the version control repository resulting from the latest download or update operation. |
| :--- | :--- |

## To connect with controllers

1. In the Communication View, specify the IP addresses for the controllers in the project.

The compiler generates different code for simulation than for targets.
2. From the Project menu, point to Connect Mode, and then click Online.

The Online menu category appears with the online command options.
3. Build the controller or project code.
4. Download the code to the physical controller or controllers.
5. In the Application View, select the controller or project to connect, then from the Online menu click Connect (or press F5).
6. In the Connect window, select the controller or controllers to connect, and then click Connect.

The selected controllers are connected.

## See also

Starting and Stopping Controllers on page 133
Download code to controllers on page 129
Disconnect from Controllers on page 137
Simulate an application on page 137
Controllers on page 108

## Modifying Cycle Timing

While running an application online, for example, connected, the cycle timing for individual controllers can be modified. When modifying cycle timing, view the following cycle timing information from system variables for individual controllers:

| Cycle Information | System Variable | Description |
| :--- | :--- | :--- |
| True Cycle Time | _-SYSVA_TCYTRUECURRENT | The real duration of a cycle |
| Current Cycle Time | _-_SYSVA_TCYCURRENT | The time spent with the virtual machine |
| Maximum Cycle Time | _-SYSVA_TCYMAXIMUM | The longest period of time used for a current cycle (__SYSVA_TCYCURRENT), since <br> connecting to the controller |
| Cycle Overflow | The number of current cycles (__SYSVA_TCYCURRENT) having exceeded the <br> programmed cycle time |  |
| Programmed Cycle Time | _-_SYSVA_TCYOVERFLOW | The defined cycle time for the controller |

The minimum cycle time for an AADvance application is 64 milliseconds. The cycle time for most applications is between 64 and 260 milliseconds, depending on the system size. The cycle time increases during an update or when a new processor is synchronizing.

Tip: Setting the Programmed Cycle Time to 0 allows the controller to execute cycles as fast as possible, for example, at a rate close to the True Cycle Time.

## To modify cycle timing

1. From the Online menu, click Cycle Timing.

The Cycle Timing dialog box displays cycle time information for all controllers defined in the project.
2. Select the controller for which to modify the cycle timing, indicate a new programmed cycle time, and then click Update.
3. (optional) To refresh the maximum cycle time and cycle overflow information display for controllers, click Reset all Statistics.

## See also

Controllers on page 108
Run an Application Online on page 127

## Disconnecting from Controllers

For a project, disconnect all running controllers executing applications. Before downloading code to controllers or clearing applications from controllers, disconnect controllers.

Tip: Disconnecting from a controller does not restore the project folder to the same state before it connected to the controller. If cleaning the project before connecting to a controller, clean the project again after disconnecting from the controller.

## To disconnect controllers

1. In the Application View, select the controller or project.
2. In the Online menu, click Disconnect.

## See also

Download code to controllers on page 129
Connect with controllers on page 134
Performing Updates on page 130
Controllers on page 108
Cleaning a Project on page 127

## Simulate an application

Simulating the running of an application signifies that virtual machines execute the code of individual controllers on the Windows ${ }^{\circledR}$ platform. Virtual machines disregard inputs and outputs.

The compiler generates different code for simulation than for online.
Build the code for the project and controllers to simulate an application. Start all controllers or start individual controllers for simulation.

The Windows platform executes all tasks such as binding exchanges and execution of program organization units (POUs). A virtual machine on the computer running the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software executes one controller.

During simulation, execute controller code with one of these execution modes:

- Real-time, the run time normal execution mode where the programmed cycle timing triggers target cycles. Switch to cycle-tocycle mode while in real-time mode.
- Cycle-to-cycle, a cyclical execution mode where the virtual machine loads the code but only executes the code after executing one cycle or activating real-time mode.
Tip: Breakpoints, cycle-to-cycle, and step-by-step execution are only available while simulating an application.

The simulation of an application follows this process:

1. Start simulation.
2. AADvance-Trusted SIS Workstation software connects to the simulator.
3. The simulator starts execution.
4. AADvance-Trusted SIS Workstation software sends the list of breakpoints to the simulator.
5. Execution stops when the application encounters the next breakpoint.

Tip: For applications with breakpoints, execution stops at the next breakpoint after receiving the list of breakpoints.
6. The controller switches to step-by-step execution.
7. Step into or step over the next code.

Set breakpoints for Ladder Diagram (LD) and Structured Text (ST) programs set to generate debug information.

## To simulate an application

1. For each controller to simulate, set these properties in the Properties window:

- Set the Code for Simulation property to True.
- Set the Start Simulation Mode to Real Time or Cycle to Cycle.

Tip: $\quad$ Set the controllers to Cycle to Cycle mode to apply breakpoints to the first cycle.
2. For the applicable programs and function blocks, set these properties in the Properties window:

- Set Generate Monitoring Symbols to True to monitor the values for operators and functions.
- Set Generate Debug Info to True to set breakpoints for programs and function blocks.

3. From the Project menu, point to Connect Mode, and then click Simulation to set the connection mode for the project.

The Simulation menu category appears with the simulation command options.
4. Build the project code.
5. In the Application View, select the project, then select one of these options from the Simulation menu:

- Click Start Simulation Now to start the simulation of all controllers in the project.
- Click Start Simulation, then in the Simulate dialog box, select the controllers for which to start simulation, and then click Simulate to start the simulation of some controllers in the project.
Tip: $\quad$ Simulate one controller by right-clicking the required controller in the Application View, and then clicking Start Simulation.

6. (optional) Set breakpoints for the applicable programs and function blocks, and then set Start Simulation Mode to Real Time for each controller to simulate.

## See also

## Building an AADvance Project on page 125

## Pausing projects and controllers during simulation on page 139

## Pausing Projects and Controllers

Pause projects or individual controllers during simulation. Pausing a project affects all controllers running in simulation. Projects or controllers running online cannot be paused. After pausing a project, resume, for example, continue simulation. After pausing a controller, continue. For example, resume simulation or execute one cycle.

| Task | Procedure |
| :--- | :--- |
| Projects | From the Simulation menu, click Pause. |
| Pause a project | From the Simulation menu, click Continue. |
| Resume a paused project | From the Simulation menu, click Pause Controller. |
| Controllers | From the Simulation menu, click Continue Controller. |
| Pause a controller | From the Simulation menu, click Execute One Cycle on Controller. |
| Resume a paused controller |  |

## See also

## Simulate an application on page 137

Forcing the Values of Variables

While connected or simulating, force or override the values of variables. These variables can be user-defined or directly represented. The behavior of a variable is defined by its logical value, physical value, lock state, and direction. When forcing the values of variables, the value to overwrite depends on the direction of the variable. Force the values of variables from the Dictionary, language containers, and the Watch window.

Locking and unlocking operates only on elementary data types for variables, array elements, and structure elements that are wired.

For locked variables, the values displayed in the Logical Value and Physical Value columns differ depending on their direction:


Example: To force the temperature reading from a sensor.

[^2]

Example: To force the closing of an actuator valve.

Tip: When you force variable values, the computer running AADvance-Trusted SIS Workstation software or AADvance Standalone OPC server sends an IXL write message to the controller. If the IXL write message is processed right after Execute target independent code (TIC) in the application execution loop (step 4 in Execution Rules on page 398), the forced value is used for the producer binding, wired output, or retained variable.
If the IXL write message is processed right after Save retain variables in the current application execution loop (step 8 in Execution Rules on page 398), the step Execute target independent code (TIC) in the next application execution loop may overwrite the forced value with a different value for the producer binding, wired output, or retained variable.
To help achieve deterministic behavior, avoid forcing an:

- internal variable that is bound as a producer or retained (internal variables cannot be locked)
- unlocked output variable


## To force the values of variables

1. From the Dictionary instance, locate the required variable.
2. Write the required value in the respective value column:

- For an input variable, write the value in the Logical Value column.
- For an output variable, write the value in the Physical Value column if the variable is locked and write the value in the Logical Value column if the variable is unlocked.


## See also

Monitoring the Values of Variables on page 140
Connect with controllers on page 134
Elementary IEC 61131-3 Types on page 402

# Monitoring the Values of Variables 

While running an application online, for example, connected, or simulating, monitor values, updated by the running code or simulation code, in Dictionary instances graphical programs, and function blocks. For individual graphical program organization units (POUs), enable monitoring by generating monitoring symbols for operators and functions.

For dictionary instances, the logical values, physical values, and lock status of variables are displayed in their respective columns. For graphical programs and function blocks, values display differently depending on type:

- Values of variables with the boolean type display using color. The output value color continues to the next input. The default colors are red when True and blue when False.
- Values of variables with the SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type display as a numeric or textual value. When the output is a structure type, the displayed value is the selected member.
Tip: $\quad$ Values of arrays with a variable as an index cannot be monitored.
When variables are unavailable, the logical and physical values display the following messages:
- OFFLINE, indication that the variable is not present in the running application code
- N/A, indication that the complex input to a user-defined function is not present in the running application code
- WAIT, indication that the variable is either:
- In online mode and attempting to connect to the target
- In simulation mode and attempting to connect to the simulator


## To monitor the values of variables

1. In the Application View, select the graphical POU for which to generate symbols monitoring.
2. In the Properties window, set the Generate Monitoring Symbols property to True.

## See also

Run an Application Online on page 127
Connect with controllers on page 134
Simulate an application on page 137

## Accessing the Values of System Variables

While running an application online, access diagnostic information from system variables for individual controllers, updated by the running code or simulation code. When running an application online, the cycle timing system variables are also available from the Cycle Timing dialog box, accessed from the Online menu.

System variables hold the values of information relating to cycle count, timing, kernel bindings, and controller information. Monitor system variables from the dictionary instances for resources. Read from and write to system variables. Available system variables are:

| Variable name | Type | Read/Write | Description |
| :--- | :--- | :--- | :--- |
| _-SYSVA_CYCLECNT | DINT | Read | Cycle counter |
| _-SYSVA_CYCLEDATE | UDINT | Read | Timestamp of the beginning of the cycle in milliseconds |
| _-SYSVA_KVBPERR | BOOL | Read/Write | Kernel variable binding producing error (production error) |
| _SYSVA_KVBCERR | BOOL | Read/Write | Kernel variable binding consuming error (consumption error) |
| _-SYSVA_RESNAME | STRING | Read | Resource name (max length=255) |
| _-SYSVA_SCANCNT | DINT | Read | Input scan counter |
| __SYSVA_TCYCYCTIME | TIME | Read/Write | Programmed cycle time. The defined cycle time for the controller. |


| Variable name | Type | Read/Write | Description |
| :---: | :---: | :---: | :---: |
| __SYSVA_TCYCURRENT | TIME | Read | Current cycle time. The time spent with the virtual machine. |
| __SYSVA_TCYMAXIMUM | TIME | Read | Maximum cycle time. The longest period of time used for a current cycle (__SYSVA_TCYCURRENT), since connecting to the controller. |
| __SYSVA_TCYOVERFLOW | DINT | Read | Cycle overflow. The number of current cycles (__SYSVA_TCYCURRENT) that have exceeded the programmed cycle time. |
| _-SYSVA_RESMODE | SINT | Read | Resource execution mode. Possible modes are: <br> - -5: Non-fatal error <br> - -4: Stopped in stepping mode after bound check exception <br> - -3 : Stopped in stepping mode after division by zero exception <br> - -2: Stopped in stepping mode after exception <br> - -1: Fatal error <br> - 0: No resource available <br> - 1: Stored resource available NOT USED (CMG) <br> - 2: Ready to run <br> - 3 : Running in real time <br> - 4: Running in cycle by cycle <br> - 5: Stopped from encountering an SFC breakpoint <br> - 7: Stopped while in stepping mode |
| __SYSVA_CCEXEC | BOOL | Write | Execute one cycle when application is in cycle to cycle mode |
| __SYSVA_TCYTRUECURRENT | TIME | Read | True cycle time. The real duration of a cycle. |

## To access the values of system variables in design mode

- From a program organization unit (POU), right-click the required variable, and select Variable Monitoring (or press Ctrl+I).


## To access the values of system variables in online or simulation mode

1. Create a spy list.
2. Open the Variable Selector from a language editor.
3. Select the System Variables tab.
4. Select all variables.
5. Right-click and select Add to <name of created spy list>.
6. Run the application online or simulate the application.
7. Open the created spy list.

## See also

## Spy Lists on page 53

Variable Selector on page 40
Run an Application Online on page 127
Simulate an application on page 137
AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software has these name conventions and limitations:

## Name conventions and limitations

| Projects |  |
| :--- | :--- |
| Project names | Project names have a maximum of 32 characters beginning <br> with a letter or single underscore followed by letters, digits, and <br> single underscores. The last character for a project name must <br> be a letter or digit; project names cannot end with an <br> underscore character. Do not use reserved words, defined <br> words, or data types (elementary, structures, or arrays) for <br> names. Names must be unique for projects within a directory. |
| Controller names | Controller names have a maximum of 128 characters beginning <br> with a letter followed by letters, digits, and single underscores. <br> The last character for a controller name must be a letter or |
| digit; controller names cannot end with an underscore |  |
| character. Do not use reserved words, defined words, or data |  |
| types (elementary, structures, or arrays) for names. Names |  |
| must be unique for the same type of elements within a scope. |  |.


| Projects |  |
| :---: | :---: |
| Function block parameter names | Function block parameter names have a maximum of 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a parameter name must be a letter or digit; parameter names cannot end with an underscore character. Do not use reserved words, defined words, or data types (elementary, structures, or arrays) for names. Names must be unique for the same type of elements within a scope. |
| Variables |  |
| Dictionary variables | The dictionary contains up to 4294967296 entries for each variable type. |
| Variable names | Variable names have a maximum of 128 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for a variable name must be a letter or digit; variable names cannot end with an underscore character. Do not use reserved words, defined words, or data types (elementary, structures, or arrays) for names. Names must be unique for the same type of elements within a scope. |
| BOOL variables | Boolean variables have the Boolean value TRUE (1) or FALSE (0). |
| SINT variables | SINT variable integer values range from -128 to +127 . Short integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is " $2 \#$ ". |
| USINT variables | USINT variable integer values range from 0 to 255 . Unsigned short integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is "8\#", and for BINARY values the prefix is "2\#". |
| BYTE variables | BYTE variable integer values range from 0 to 255 . BYTE constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is " $2 \#$ ". |
| INT variables | INT variable integer values range from -32768 to +32767 . Integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is "2\#". |
| UINT variables | UINT variable integer values range from 0 to 65535 . Unsigned integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is "2\#". |
| WORD variables | WORD variable integer values range from 0 to 65535. WORD constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is " $2 \#$ ". |
| DINT variables | DINT variable integer values range from -2147483648 to +2147483647 . Double integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is " $2 \#$ ". |
| UDINT variables | UDINT variable integer values range from 0 to 4294967295. Unsigned double integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is " $16 \#$ ", for OCTAL values the prefix is "8\#", and for BINARY values the prefix is " $2 \#$ ". |


| Projects |  |
| :---: | :---: |
| DWORD variables | DWORD variable integer values range from 0 to 4294967295. Double word constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is " $2 \#$ ". |
| LINT variables | LINT variable integer values range from - 9223372036854775808 to +9223372036854775807 . Long integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is "8\#", and for BINARY values the prefix is "2\#". |
| ULINT variables | ULINT variable integer values range from 0 to 18446744073709551615. Unsigned long integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is "8\#", and for BINARY values the prefix is "2\#". |
| LWORD variables | LWORD variable integer values range from 0 to 18446744073709551615. Long word constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16\#", for OCTAL values the prefix is " $8 \#$ ", and for BINARY values the prefix is "2\#". |
| REAL variables | Real variables have six significant digits. For larger values, the maximum possible value is $\pm 3.402823466 \mathrm{E}+38$ while for smaller values, the minimum possible value is $\pm 1.175494351 \mathrm{E}-$ 38. Values greater than $\pm 3.402823466 \mathrm{E}+38$ and greater than 0.0 but less than $\pm_{1.175494351 E-38}$ are not supported. Write real literal values with either decimal or scientific representation. The exponent part of a real scientific expression must be a signed integer value ranging from - 37 to +37. The scientific representation uses the 'E' letter to separate the mantissa part and the exponent. |
| LREAL variables | Long real variables have 15 significant digits. For larger values, the maximum possible value is $\pm 1.7976931348623158 \mathrm{e}+308$ while for smaller values, the minimum possible value is $\pm$ 2.22507385850721E-308. Values greater than $\pm$ $1.7976931348623158 \mathrm{e}+308$ and greater than 0.0 but less than $\pm 2.22507385850721 \mathrm{E}-308$ are not supported. Write long real literal values with either decimal or scientific representation. The range of a real scientific expression must be a signed integer value from 1.7E -308 to $1.7 \mathrm{E}+308$. |
| TIME variables | Time variables can have positive values ranging from 0 to 49d17h2m47s294ms. The time literal value must begin with the "T\#" or "TIME\#" prefix. |
| DATE variables | Date variable values range from 1970-01-01 to 2038-01-18. The date literal expression must begin with the "D\#" or "DATE\#" prefix. |
| STRING variables | STRING variable string capacity has a maximum of 255 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. String variables can contain any character of the standard ASCII table. Characters must be preceded and followed by single quote (') characters. When placing single quote (') characters within a string literal, these characters must be preceded by the dollar (\$) character. |
| Alias names | Alias names have a maximum of 128 characters consisting of letters, digits, and these special characters: !, \#, \$, \%, \& , , , *, +, <br>  |
| User-defined Typ |  |

\(\left.$$
\begin{array}{l|l}\hline \text { Projects } & \\
\hline \text { Array names } & \begin{array}{l}\text { Array names have a maximum of } 128 \text { characters beginning with } \\
\text { a letter or single underscore followed by letters, digits, and } \\
\text { single underscores. The last character for an array name must } \\
\text { be a letter or digit; array names cannot end with an underscore } \\
\text { character. Do not use reserved words, defined words, or data } \\
\text { types (elementary, structures, or arrays) for names. Names } \\
\text { must be unique. }\end{array} \\
\hline \text { Structure names } & \begin{array}{l}\text { Structure names have a maximum of 128 characters beginning } \\
\text { with a letter or single underscore followed by letters, digits, and } \\
\text { single underscores. The last character for a structure name } \\
\text { must be a letter or digit; structure names cannot end with an }\end{array}
$$ <br>

\hline underscore character. Do not use reserved words, defined\end{array}\right]\)| words, or data types (elementary, structures, or arrays) for |
| :--- | :--- |
| names. Names must be unique. |

## See also

Working with AADvance Applications on page 103
The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software uses these default file system paths:

- AADvance projects: \%USERPROFILE\%\Documents\SIS Workstation <version>\Projects
- Version control repository: \%USERPROFILE\%\AppData\Local\Rockwell Automation\SIS Workstation <version>\ACP\Templates\DocumentGenerator
- Documentation Generation templates: \%PUBLIC\%\Public Documents\SIS Workstation <version>\Templates\DocumentGenerator
- OPC configuration: \%ALLUSERSPROFILE\%\Rockwell Automation\SIS Workstation <version>\AADvance Gateway
- AADvance-Trusted SIS Workstation software and OPC logs: \%USERPROFILE\%\AppData\Local\Rockwell Automation\SIS Workstation <version>
- Application binary files: \%PROGRAMFILES(x86)\%\Rockwell Automation\SIS Workstation <version>
- Environment settings file (when exporting): \%USERPROFILE\%\Documents\SIS Workstation <version>\Settings\SIS Workstation


## See also

Working with AADvance Applications on page 103
Manage the changing versions of AADvance elements including projects, libraries, controllers, programs, functions, and function blocks by saving these to a version control repository. Saving these elements to a control repository enables multiple users to work on the same solutions and project elements at the same time and retrieve older versions of elements at a later time.

The version control automatically checks in elements to the version control repository upon performing the following operations:

- Creating a project
- Downloading code to targets
- Updating targets

Such automatic check-ins ensure that the code located on a target always has a corresponding version stored in the version control repository. When a user performs one of these operations for a project or an element while others also have elements checked out, the user performing the operation steals the locks from the other users. Modifications performed on such elements by the original lock holder are no longer available for committing.

When modifying elements, automatically check out elements then check in these modifications.

From the repository, check out elements for exclusive access when making modifications. Also, upon modifying checked-in elements, these elements become checked out for exclusive access. Afterwards, check in modifications performed on these elements into the repository. When checking in elements, choose to recursively check in all sub-elements to the repository.

When deleting, renaming, and adding elements, check out the parent of that element.

| Element | Parent |
| :--- | :--- |
| Program, Function, Function Block | Controller, Library |
| Controller | Project |
| Library | Project |

When retrieving, for example updating, an AADvance element from the repository, choose to update this element to the latest version or any specific version previously checked in to the repository. When getting the latest version, recursively replace all sub-elements from the repository. When retrieving a specific version from the version control history, the local element and its sub-elements revert to those from the version control. For controllers having been renamed, retrieve the required version from the controller
history; the required controller version from the project history cannot be retrieved.

Tip: When a controller has been renamed, execute the View History command on the most recent version of the controller.

From the Application View, view the version control status for all elements. From the Communication View and Equipment View, view the version control status for projects and controllers.

| $\square$ | Available element where local version is the same as the version in the repository |
| :---: | :---: |
| $\square$ | Available element where local version differs from the modified version in the repository |
| $\square$ | Available element where locally modified version differs from the version in the repository |
| ¢ | Locked element currently checked out by another user where local version is the same as the version in the repository |
| $\stackrel{\square}{*}$ | Locked element currently checked out by another user where local version differs from the modified version in the repository |
| 苗 | Locked element currently checked out by another user where locally modified version differs from the version in the repository |
| $\stackrel{\sim}{\sim}$ | Checked out element where local version is the same as the version in the repository |
| $\underline{=}$ | Checked out element where locally modified version differs from the version in the repository |
| 米 | New element not yet checked in to the repository |

When working with version control, perform these tasks:

- Define a Repository
- Check In Elements to Version Control
- Check Out Elements from Version Control
- Undo Check Outs of Elements
- Review Pending Checkins
- Get the Latest Versions of Elements
- Label a Version
- Archive and Restore Repository Projects
- Access Repository Projects
- Publish Libraries
- Work Offline and Reconnecting to a Repository
- View the History of Elements
- Refresh the Version Control Status of Elements

This product includes software developed by * CollabNet (http://www.Collab.Net/) based on the Subversion source control plug-in for Visual Studio.

## See also

Define a Repository on page 148
Check In Elements to Version Control on page 149
Checking Out Elements from Version Control on page 150
Labeling a Version on page 153
Refreshing the Version Control Status of Elements on page 159

## Define a Repository

AADvance uses a repository where it stores version control files containing the history for projects and elements. Define a repository at any local or
server (URI) location. AADvance creates all necessary files in which to store the changes for projects and elements for the repository. The repository folder name must be unique and cannot contain a project name. The default installation for the repository is the following:
\%PUBLIC\%\Public Documents\SIS Workstation <version>\Repository

## Tip: AADvance-Trusted SIS Workstation software has issues when using version control with a Subversion server. Use folders shared between computers instead of the Subversion server.

When opening projects, the available projects are those from the current repository.

The information stored in a repository does not include customizations to the AADvance-Trusted SIS Workstation software options.

Tip: When defining a control repository location, specify an empty folder. AADvance
automatically creates all necessary files for the repository in the folder.
Considerations when sharing a repository:

- Before connecting an application, get the latest version of the project from version control.
- Use a string type global variable to enter the user name when taking the restricted access on the controller.
- For a project containing a POU that was renamed or deleted on a different workstation, check out the modified POU and controller, and then check in all pending changes.


## To define a repository

1. From the File menu, point to Version Control, and then click Select Repository.
2. In the Select Repository dialog box, enter the path for the repository or browse for the repository folder location.

Check In Elements to Version Control

## See also

Version Control on page 147
Check in AADvance elements like projects, libraries, controllers, programs, functions, and function blocks to a repository. Check in elements to manage the changing versions of elements and retrieve previous versions. Also add elements to previously checked in projects.

## Prerequisites:

- The project contains no empty structure types, array types, userdefined function block POU bodies, or variables.
- The project contains no child SFC program organization units (POUs) for Sequential SFC Chart (SFC). Use only SFC programs and function blocks.

When creating a project, all elements are automatically checked in to the repository.

Use a recursive check in to check in elements with sub-elements. Also choose to keep elements checked out following a check-in. Recursive operations do not modify elements locked while checked out by other users.

> | Important: | Immediately check in modifications after deleting or renaming elements |
| :--- | :--- |
|  | to ensure that subsequent deleting and renaming are possible. |

Perform version control operations for all elements from the Application View. Perform version control operations for projects and controllers from the Communication View and Equipment View.

## To add or check in elements to version control

1. From the required view, select the element to add or check in to the repository.
2. From the File menu, point to Version Control, and then click Check In.
3. In the Check In dialog box, indicate whether to perform a recursive check in and whether to keep the elements checked out following the check-in, and then click OK.

## See also

Define a Repository on page 148
Checking Out Elements from Version Control on page 150
Check out AADvance elements like projects, libraries, controllers, programs, functions, and function blocks from a repository. Checking out provides exclusive access when modifying these elements. When modifying elements that are checked in, these elements are automatically checked out when not checked out by another other user

When creating a project, all elements are automatically checked in to the repository.

When checking out elements with sub-elements, perform a recursive check out which includes checking out sub-elements. When performing recursive operations, elements locked while checked out by other users are not modified.

From the Application View, perform version control operations for all elements. From the Communication View and Equipment View, perform version control operations for projects and controllers.

## To check out elements from version control

1. From the Application View, select the project or element to check out from the repository.
2. From the File menu, point to Version Control, and then click Check Out.
3. In the Check Out dialog box, indicate whether to perform a recursive check out including all sub-elements, and then click OK.
The elements are checked out from the repository and display with the unlocked icon in the Application View.

## See also

Version Control on page 147
Check In Elements to Version Control on page 149

## Undoing Check Outs of Elements

Undo check outs of projects, libraries, controllers, programs, functions, and function blocks from a repository. Undoing a check out undoes all changes to the elements definitions since these were checked out. The elements are no longer checked out, but instead of updating the repository with the edited version, Undo Check Out reverts to the local version to the latest version from the repository.

When performing recursive operations, elements locked or checked out by other users are not modified.

From the Application View, perform version control operations for all elements. From the Communication View and Equipment View, perform version control operations for projects and controllers.

| Important: | The Undo Check Out operation may take longer than expected for <br> larger applications. |
| :--- | :--- |

## To undo a checkout of an element or project

1. From the required view, select the element or project for which to undo the check out from the repository.
2. From the File menu, point to Version Control, and then click Undo Check Out.
3. In the Undo Check Out dialog box, indicate whether to perform a recursive undo check out including all sub-elements, and then click OK.
The elements are replaced in the repository and display with the checked in icon in the Application View.

## See also

Checking Out Elements from Version Control on page 150
Check In Elements to Version Control on page 149

## Reviewing Pending <br> Checkins

To access the list of pending checkins
From the View menu, click Pending Checkins.
The Pending Checkins window provides a global view of all checked out and locally modified elements for a specific user. In this window, perform these version control operations:

| Operation | Procedure |
| :---: | :---: |
| Check in individual or all checked out and locally modified elements | 1．In the Pending Checkins window，select the elements to check in to the repository． <br> 2．On the Pending Checkins toolbar，click 眿。 |
| Undo check outs for all checked out and locally modified elements | 1．In the Pending Checkins window，select the elements for which to undo check outs． <br> 2．On the Pending Checkins toolbar，click ${ }^{1}$ ． |
| Refresh the display of checked out and locally modified elements | In the Pending Checkins toolbar，click 且． |

Files for pending checkins have the following action types：locked，deleted， modified，added，and renamed．

When performing check ins and check outs，choose to keep files checked out．

## See also

Check In Elements to Version Control on page 149
Checking Out Elements from Version Control on page 150

## Getting the Latest Versions of Elements

Get the latest version for elements like projects，libraries，controllers， programs，functions，and function blocks．When getting the latest version， recursively replace all sub－elements from the repository．When performing recursive operations，elements locked or checked out by other users are not modified．

Important：Before getting a previous version for a controller with binding changes，check out all controllers affected by these changes．

From the Application View，perform version control operations for all elements．From the Communication View and Equipment View，perform version control operations for projects and controllers．

Tip：The Get Latest operation may take longer than expected for larger applications．

## To get the latest version of elements

1．From the required view，select the element for which to get the latest version from the repository．
2．From the File menu，point to Version Control，and then click Get Latest．
3．In the Get Latest dialog box，indicate whether to perform a recursive get latest version including all sub－elements and whether to keep or replace locally modified elements，and then click OK．

## See also

Check In Elements to Version Control on page 149
Checking Out Elements from Version Control on page 150

Labeling a Version

## Archive and Restore Repository Projects

Label a version of a project or an element for reference in the repository. The label is also applied to all sub-elements of the latest version checked in to the repository.

From the Application View, perform version control operations for all elements. From the Communication View and Equipment View, perform version control operations for projects and controllers.

## To label a version

1. From the required view, select the project or element for which to apply a label.
2. From the File menu, point to Version Control, and then click Label.
3. In the Add Label dialog box, enter the text to display as the label in the version control history, and then click OK.

## See also

Check In Elements to Version Control on page 149
View the History of Elements on page 159
Version Control on page 147
Archive projects contained in a repository and restore projects that were previously archived. Archiving a repository project means compressing the content of a project into a *.vsc file. Either archive the complete version of a project, including the modification history and all comments, or archive the latest version of a project checked into a repository. Then restore the archived file in another repository. The archived file can then be restored in another repository. Performing updates or connecting to controllers is unavailable after restoring a project archived at the latest version.

> Tip: To reduce the time taken to perform archive and restore operations, have a different repository folder for each project.

Restore AADvance projects archived with AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software version 1.1 or AADvance ${ }^{\oplus}$ Workbench version 2.1. Projects archived from AADvance-Trusted SIS Workstation software version 1.1 cannot be restored in previous versions of the AADvance Workbench or AADvance-Trusted SIS Workstation software.

Tip: Before archiving a project at the latest version, it is recommended to label the version of the project, ensuring the version control history displays the expected description after project restoration.

## To archive and restore repository projects

To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required for AADvance projects, right-click the SIS Workstation Windows desktop shortcut and select Run as administrator. Then, continue with one of the procedures in this table.

| Task | Procedure |
| :--- | :--- |
| Archive a repository | 1. From the required view, select the project to archive. <br> 2. From the File menu, point to Version Control, and then click Archive. <br> 3. In the Archive Project dialog box, indicate whether to archive the complete project history or the latest <br> version, and then click Archive. <br> 4. In the Save As dialog box, specify a name and storage location for the archive file, and then click Save. |
| Restore a repository | Restore AADvance projects archived with AADvance-Trusted SIS Workstation software version 1.1 or AADvance <br> Workbench version 2.1. <br> Tip: Before restoring the project, limit the Projects location path length to 70 <br> characters. Setting a longer path may exceed the maximum length of 260 <br> characters supported by Windows operating systems. <br> When setting the project location, specify a local path; network paths may <br> cause unexpected results. |
|  | 1. From the File menu, click New Project. <br> 2. In the New Project dialog box, from the AADvance tab, click Restore in the Project Types. <br> 3. In the Templates list, click the Restore Project from an Archive template, and then click Browse. <br> 4. In the Select Archive File dialog box, locate the project archive, and then click Open. <br> 5. The default name and storage location for archive files is the project name and folder. <br> 6. In the New Project dialog box, click OK. <br> 7. (Optional) To update the project, perform a Get version on the download or update actions that correspond <br> to the application version running on the controller. |
| running on the controller. |  |
| 11. Perform an update on the controllers. |  |

From the Application View, perform version control operations for all elements. From the Communication View and Equipment View, perform version control operations for projects and controllers.

## See also

## Labeling a Version on page 153

Accessing Repository Projects on page 154

## Specify Project Options on page 75

## View the History of Elements on page 159

## Accessing Repository Projects

While working with AADvance controllers, access projects stored in a repository. When multiple computers need to access the same project, it is recommended to use a repository in a shared folder. This preferred method enables different users to check out and modify the project while ensuring a
complete modification history. Only one user can check out a project element at any given time.

When only using a locally stored repository, the may require transferring the storage location of the repository to another computer. Transferring a repository also transfers all of the projects contained within the repository folder. Needing to transfer the storage location of the repository occurs when upgrading to a new computer or permanently changing workstation. Keep only one instance of the repository folder.

When needing to access a specific project on a computer other than the one storing the local repository folder, it is recommended to archive the project and then restore the *.vsc file in the local repository folder of the other computer.

Tip: Modification history is retained when restoring the project.

## See also

To access a repository in a shared folder on page 155
To transfer a repository to another computer on page 156
To access a project stored in the local repository of another computer on page 156

Archive and Restore Repository Projects on page 153

To access a repository in a shared folder

Accessing a repository in a shared folder enables different users to check out and modify the project while ensuring a complete modification history. Only one user can check out a project element at any given time.

| Important: | A project with the same name as a project saved in the projects <br> directory cannot be opened; delete the local copy before opening the <br> project. |
| :--- | :--- |

## To access a repository in a shared folder

1. From the File menu, point to Version Control, and then click Select Repository.
2. In the Select Repository dialog box, click Browse to locate the shared folder, and then click OK.
3. For all required computers, ensure the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software uses the repository located in the shared folder.

## See also

Accessing Repository Projects on page 154
To transfer a repository to another computer on page 156
To access a project stored in the local repository of another computer on page 156

To transfer a repository to another computer

Transfer the storage location of the repository to another computer.
Transferring a repository also transfers all of the projects contained within the repository folder. Keep only one instance of the repository folder.

| Important: | A project with the same name as a project saved in the projects <br> directory cannot be opened; delete the local copy before opening the <br> project. |
| :--- | :--- |

## To transfer to a repository to another computer

1. Transfer the repository folder to another computer.

Tip: Make sure to remove the repository folder located on the original computer.
2. On the other computer, open the AADvance ${ }^{\circledR}$ - Trusted ${ }^{\circledR}$ SIS

Workstation software and then from the File menu, point to Version Control, and then click Select Repository.
3. In the Select Repository dialog box, click Browse to locate the transferred repository folder, and then click OK.

## See also

Accessing Repository Projects on page 154
To access a repository in a shared folder on page 155
To access a project stored in the local repository of another computer on page 156

Access a project stored in the local repository of another computer by archiving a project and then restoring the *.vsc file in the local repository folder of the other computer.

Tip: Modification history is retained when restoring the project.

| Important: | A project with the same name as a project saved in the projects <br> directory cannot be restored; delete the local copy before restoring the <br> project. If the project is already stored in the repository, restore the <br> project in another repository folder. Replacing the project currently <br> stored in the repository folder may corrupt the project. |
| :--- | :--- |

## To access a project stored in the local repository of another computer

1. From the Application View, right-click the project, point to Version Control, and then click Archive.
2. In the Archive Project dialog box, select the Complete archive mode, and then click Archive.
3. In the Save As dialog box, specify a name and storage location for the archive file, and then click Save.
4. Transfer the archived ${ }^{*}$.vsc file to another computer.
5. On the other computer, open the AADvance ${ }^{\oplus}$-Trusted ${ }^{\otimes}$ SIS Workstation software and then from the File menu, click New Project.
6. In the New Project dialog box, select the AADvance tab, and then click Restore in the Project Types list.
7. In the Templates list, click the Restore Project from an Archive template, and then click Browse.
8. In the Select Archive File dialog box, locate the project archive, and then click Open.
9. In the New Project dialog box, click OK.

The project is checked into to the repository located on the other computer.
10.Perform a Get version on the download or update actions that correspond to the application version running on the controller.

## See also

Accessing Repository Projects on page 154
To access a repository in a shared folder on page 155
To transfer a repository to another computer on page 156
Publishing libraries makes these available from the repository for use in other projects. Publish a library after checking in the library. Publish libraries that are created in projects.

When publishing a library, enter this information:

| Name | The name of the library |
| :--- | :--- |
| Version | The version number of the library. Enter numerical values. |
| Comment | Free-format text |

## To publish a library

1. From the Application View, select the library to publish.
2. From the File menu, point to Version Control, and then click Check In.
3. In the Check In dialog box, select whether to keep the library checked out, and then click OK.
4. Right-click the library, and then click Publish Library.
5. In the Publish dialog box, enter a name, version, and comment for the library, and then click Publish.
Tip: Use only numerical values for the library version. Using a non-numerical value causes the Add Published Library dialog box to display a blank library version.
A copy of the library appears in the repository.

## See also

Check In Elements to Version Control on page 149

## Define a Repository on page 148

## Working Offline and Reconnecting to a Repository

## AADvance Libraries on page 118

While working with files from a repository, a local working copy may become disconnected from the repository for reasons like a disconnected network cable. In this case, continue to work offline with a local working copy. However, after reconnecting to the repository, the working copy may differ from the copy located in the repository if other users modified that version.

## To work offline and reconnect to a repository

1. When a local working copy becomes disconnected from the repository, a message appears informing of the disconnection from the repository.

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software attempts to reconnect with the repository.
2. When unable to reestablish the connection, choose to work offline with the local file by creating a temporary local repository:
a. Save and close the current project.
b. To set a local folder to a new or existing repository, from the File menu, point to Version Control, and then click Select Repository.
c. To point to the previously saved project, from the File menu, click New Project, and from the New Project dialog box import the required AADvance project.
d. Proceed to making the necessary modifications to the project.

> | Important: | Downloading this local copy prevents other team members from |
| :--- | :--- |
|  | connecting to the respective controller. Also, the history for the |
| local file is not transferred to the temporary local repository. |  |

3. When the connection is restored with the original repository, return your local working copy to the original repository:
a. For each element to merge back, save the element as its original name in the original repository.

The element is saved as a *.pxf file.
b. Open the original project file, and then import the element *.pxf file.
c. Check in the new version of the element.

> | Important: | $\begin{array}{l}\text { When checking in elements that were modified locally, } \\ \text { modifications other users performed for the element may be } \\ \text { overwritten. }\end{array}$ |
| :--- | :--- |

4. To enable connecting to the physical controllers, download the most current code to the required controllers.

## See also

## View the History of Elements

View the history of elements such as projects, libraries, controllers, programs, functions, and function blocks. The history of an element is available from the time of creation.

When viewing the history of an element, a list displays all available versions of the element with their assigned version numbers. Each version entry includes the date, the user, the action, and a description from the check-in. The project version number pertains to the revision number in the repository currently used. Switching the project to another repository might cause the revision number to change.

Tip: When a controller has been renamed, execute the View History command on the most recent version of the controller.

When viewing the history of elements, perform these tasks:

| Task | Procedure |
| :--- | :--- |
| Retrieve a specific version of an <br> element | Before getting a previous version for a controller with binding changes, check out all controllers affected by these changes. <br> When retrieving a specific version from the version control history, the local element and its sub-elements revert to those from <br> the version control. <br> The Get Version command only gets application binaries on update or download actions for the selected controller version. <br> Updating the controller requires these application binaries. |
| 1. From the Application View, select the element for which to get the latest version from the repository. <br> 2. From the File menu, point to Version Control, and then click View History. <br> 3. In the Version History dialog box, select the required version from the list of available versions, and then click Get Version. |  |
| Refreshing the list of versions <br> available for an element | To update the list of displayed versions available for an element, click Refresh. |
| Clearing the list of versions for an |  |
| element | When clearing the list of versions for an element, the history for the element remains in the repository. Therefore, subsequently <br> retrieving the history for the element produces the versions listed prior to clearing the list. |

## See also

## Getting the Latest Versions of Elements on page 152

## Refreshing the Version Control Status of Elements

Version control status information for elements is available from many sources.

- From the Application View, refresh the displayed control status for all elements.
- From the Communication View and Equipment View, refresh the displayed control status for projects and controllers.
- From the Application View, Communication View, and Equipment View, obtain version control status details by mousing over the version control icons.
- In the Output window, display complete status details for an element including its version control status, locked by, locked on, path, user, version time, latest revision, and local revision.

| Task | Procedure |
| :--- | :--- |
| Refresh the control status of elements | From the required view, right-click any element in the structure, point to Version Control, and then click <br> Refresh Status. |
| Display complete status information for an element | From the required view, right-click any element in the structure, point to Version Control, and then click <br> Refresh Status. |

## See also

View the History of Elements on page 159

## Dictionary

To access a Dictionary grid instance

1. From the Application View, expand the project and controller nodes.
2. For the variables of a controller, expand the required controller node, then expand the Variables node, and then double-click Controller Variables.

The Dictionary instance displays containing the variables belonging to the controller.
3. For the variables of a program, expand the required program node, and then double-click Local Variables.

The Dictionary instance displays containing the variables belonging to the program.
4. For the data types of a project, double-click the Global Data Types element.

The data types Dictionary instance displays with the Arrays, Structures, and Defined Words tabs.
5. For the data types of a library, expand the Libraries node, then expand the required library, and then double-click Data Types.

Use the Dictionary, for example tag editor, environment to manage variables, arrays, structures, defined words, and edit the contents of the grid. The Dictionary consists of multiple grids with different purposes.

- Arrays Grid, enables managing the arrays for a project
- Structures Grid, enables managing the structures for a project
- Defined Words Grid, enables managing the defined words for a project
- Variables Grid, enables managing the variables for controllers and programs. Each controller and program has its instance of the grid. For controllers, the grid displays global variables. For programs, the grid displays local variables.

The grids each display the properties for the type of element. Open multiple grid instances simultaneously. When working in a grid, navigate the cells using the mouse controls. For complex data types, expand fields using Ctrl+PLUS SIGN on numeric keypad (+) and collapse fields using Ctrl+MINUS SIGN on numeric keypad (-).

Access Dictionary grids from the Application View.
Customize the Dictionary environment by arranging the columns to display and setting the display colors.

[^3]
## To arrange the columns to display

1. To move a column, drag the column header to another location. When dragging a column header, arrows indicate the current position of the header.
2. To hide a column, right-click a column header, then click Hide Column.
3. To show a column, right-click on any column header, click Show Column, and then select the desired column name.

## See also

## Arrays Grid on page 161

Structures Grid on page 162
Defined Words Grid on page 163
Variables Grid on page 164
Cross Reference Browser on page 168

## Arrays Grid

Use the Arrays grid of the Dictionary to manage the arrays for a project.
Perform these tasks from the Arrays grid:

| Task | Procedure |
| :--- | :--- |
| Create an array | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Arrays tab. <br> 3. In an empty row of the Arrays grid, define the required properties for the array, and then press Enter. |
| Edit an existing array | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Arrays tab. <br> 3. In the Arrays grid, make the required changes. |
| Delete an array | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Arrays tab. <br> 3. In the Arrays grid, right-click the array to remove, and then click Delete. |
| Sort arrays in the grid | 1. From the Application View, expand the project, controller, and lib nodes, then double-click the Data Types item. <br> 2. From the Data Types instance, click the Arrays tab. <br> 3. In the Arrays grid, select the required column header. <br> 4. An arrow showing the current order is displays on the column header. <br> 5. Toggle the column header to switch between ascending and descending order. |
| Filter arrays in the grid | Filter arrays display on the Arrays tab of Data Types instance. When filtering, create a view displaying only the arrays containing <br> specified characters. <br> The filter row is the top row of the grid. Filter arrays by typing alphabetical and numerical characters in the cells of the filter row. <br> Also select from the drop-down combo box. Matching arrays automatically display. <br> 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Arrays tab. <br> 3. In the filter row of the Arrays grid, click the required cell, and then do one of these: <br> - Type the characters to use in the filtering operation <br> - Select the required array from the drop-down combo box |

Properties for arrays are:

| Property | Description | Possible Values |
| :--- | :--- | :--- |
| Name | Name of the array | Limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single <br> underscores. The last character for an array name must be a letter or digit; array names cannot end with an <br> underscore character. Reserved words, defined words, or data types such as, elementary, structures, or arrays <br> are not valid names. Use unique names for the same type of elements within a scope. |

Chapter 2 Working with AADvance Applications

| Property | Description | Possible Values |
| :--- | :--- | :--- |
| Data Type | Type of the array | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, <br> STRING, User arrays, Structures |
| Dimension | The dimension of the <br> array | Example: [1.10] for a one dimensional array, [1.4, ,...7], for a two dimensional array. The dimension is defined <br> as a positive double integer (DINT) value. <br> Tip:Dimension does not apply to function block instances. Avoid setting <br> Dimension for function block instances to help prevent unexpected build <br> results. <br> Comment <br> String SizeComment for the array <br> If Data Type is STRING, <br> represents the length |
| Maximum of 255 characters when using only ASCII characters. <br> Maximum of 127 characters when using at least one non-ASCII character. |  |  |
| String capacity is limited to 255 characters excluding the terminating null character (0), a byte for the current <br> length of the string, and a byte for the maximum length of the string |  |  |

Customize the Dictionary environment by arranging the columns to display.

## See also

Structures Grid on page 162
Defined Words Grid on page 163
Variables Grid on page 164
Dictionary on page 160

## Structures Grid

The Structures grid of the Dictionary enables managing the structures for a project. Tasks to perform from the Structures grid are:

| Task | Procedure |
| :--- | :--- |
| Create a structure | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Structures tab. <br> 3. In an empty row of the Structures grid, define the required properties for the structure, and then press Enter. <br> Tip: Structures are only tested to three levels. |
| Delete a structure | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Structures tab. <br> 3. In the Structures grid, right-click the structure to remove, and then click Delete. |
| Edit an existing structure | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Structures tab. <br> 3. In the Structures grid, make the required changes, and then press Enter. <br> Tip: Changing the data type of a structure member may prevent expanding <br> variables to display structure members. Close and then reopen the <br> project to expand variables. |
|  | - Changing the data type of a structure member may cause build errors. <br> Initial values are not reset. Change the data type of the variable and then <br> change the data type back to the original data type. |


| Task | Procedure |
| :--- | :--- |
| Filter structures in the grid | Filter structures in Structures grid. When filtering, create a view displaying only the structures containing specified characters. <br> The filter row is the top row of the grid. Filter structures by typing alphabetical and numerical characters in the cells of the filter <br> row. You can also select from the drop-down combo box. Matching structures are automatically displayed. <br> 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Structures tab. <br> 3. In the filter row of the Structures grid, click the required cell, then do one of these: <br> - Type the characters to use in the filtering operation <br> - Select the required structure from the drop-down combo box |

Properties for structures are:

| Property | Description | Possible Values |
| :--- | :--- | :--- |
| Name | Name of the structure | Limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single <br> underscores. The last character for a structure name must be a letter or digit; structure names cannot end with <br> an underscore character. Reserved words, defined words, or data types such as, elementary, structures, or <br> arrays, are not valid names. The same type of elements within a scope must have unique names. |
| Data Type | Type of the structure | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, <br> STRING, User arrays, Structures |
| Comment | Comment for the structure | Maximum of 255 characters when using only ASCII characters. <br> Maximum of 127 characters when using at least one non-ASCII character. |
| String Size | If Data Type is STRING, <br> represents the length | String capacity is limited to 255 characters excluding the terminating null character (0), a byte for the current <br> length of the string, and a byte for the maximum length of the string |

Customize the Dictionary environment by arranging the columns to display.

## See also

## Arrays Grid on page 161 <br> Defined Words Grid on page 163 <br> Variables Grid on page 164 <br> Dictionary on page 160

## Defined Words Grid

Use the Defined Words grid of the Dictionary to manage the defined words for a project. Perform these tasks from the defined words grid:

| Task | Procedure |
| :--- | :--- |
| Create a defined word | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Defined Words tab. <br> 3. In the Defined Words grid, define the required properties, and then press Enter. |
| Edit an existing defined word | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Defined Words tab. <br> 3. In the Defined Words grid, make the required changes. <br> Tip:The compiler does not detect modifications made to existing defined <br> words. For example, changing the equivalent value. Create a new global <br> variable to force the incremental build to assess all changes. <br> Delete a defined word1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Defined Words tab. <br> 3. In the Defined Words grid, right-click the defined word to remove, and then click Delete. |


| Task | Procedure |
| :--- | :--- |
| Sort defined words in the grid | 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Defined Words tab. <br> 3. In the Defined Words grid, select the required column header. <br> 4. An arrow showing the current order is displayed on the column header. <br> 5. Toggle the column header to switch between ascending and descending order. |
| Filter defined words in the grid | Filter defined words in Defined Words grid. When filtering, create a view displaying only the defined words containing specified <br> characters. <br> The filter row is the top row of the grid. Filter defined words by typing alphabetical and numerical characters in the cells of the <br> filter row. Also select from the drop-down combo box. Matching defined words are automatically displayed. <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br> 1. From the Application View, access the Dictionary instance for the data types of the project. <br> 2. From the Data Types instance, click the Defined Words tab. <br> 3. In the filter row of the Defined Words grid, click the required cell, and then do one of these: <br> - Type the characters to use in the filtering operation <br> - Select the required defined word from the drop-down combo box |

Properties for defined words are:

| Property | Description | Possible Values |
| :--- | :--- | :--- |
| Word | Name of the defined word | Limited to 128 characters beginning with a letter or single underscore followed by <br> letters, digits, and single underscores. The last character for a defined word must <br> be a letter or digit; defined words cannot end with an underscore character. <br> Reserved words, defined words, or data types, such as elementary, structures, or <br> arrays are not valid names. Use unique names for the same type of elements <br> within a scope. |
| Equivalent | String replacing the defined word during <br> compilation. Example: the defined word <br> "Pl" is replaced by its equivalent "3.14159" | Limited to 128 characters <br> Comment Comment for the defined word | | Maximum of 255 characters when using only ASCII characters. |
| :--- |
| Maximum of 127 characters when using at least one non-ASCII character. |

Customize the Dictionary environment by arranging the columns to display.

## See also

Variables Grid on page 164
Structures Grid on page 162
Arrays Grid on page 161
Dictionary on page 160

## Variables Grid

Use the variables grid of the Dictionary to manage the variables for a controller or program. Each controller and program has its instance of the grid. The grid displays global variables for controllers. The grid displays the local variables for programs. Perform these tasks from the variables grid:

| Task | Procedure |
| :--- | :--- |
| Create a variable | 1. From the Application View, access the Dictionary instance for the required controller or program. |
|  | 2. In an empty row of the variables grid, define the required properties for the variable, and then press Enter. |
| Edit an existing variable | 1. From the Application View, access the Dictionary instance for the required controller or program. <br>  |


| Task | Procedure |
| :---: | :---: |
| Drag a variable | Drag variables from a Dictionary instance to multiple locations within a project. These locations include other Dictionary instances and elements within a language container. <br> Drag variables to other locations individually. When dragging a variable to another Dictionary instance, place the variable anywhere in the grid. When dragging a variable into a language container, place the variable anywhere in the language container. To retain changes made to Dictionary instances and language containers, save the respective instance or POU before closing. <br> 1. From the Application View, access the Dictionary instance containing the required variable and the destination for the variable. <br> 2. From the variables grid of the Dictionary instance containing the required variable, select the variable by clicking the cell in the left-most column. <br> 3. The selection indicator $($ is displays in the leftmost column. <br> 4. Drag , placing the variable in the grid or open language container. <br> The variable displays at the destination. |
| Delete a variable | Deleting variables from an instance opened for a program element removes the variables from the instance only. <br> 1. From the Application View, access the Dictionary instance for the required controller or program. <br> 2. Right-click the variable to remove in the variables grid, and then click Delete. |
| Cut, copy, and paste variables | When selecting variables, an indicator arrow () displays in the leftmost column of the grid. Cut, copy, and paste variables from one Dictionary instance to another. Also cut, copy, and paste variables from the Variable Selector to the Dictionary. <br> 1. In the grid of the required Dictionary instance, cut or copy the required variables. <br> - To remove variables, select the required variable or variables, right-click the selection, and then click Cut. <br> - To copy variables, select the required variable or variables, right-click the selection, and then click Copy. <br> 2. In the grid of the required Variable Selector tab or Dictionary instance, right-click the required location, and then click Paste. |
| Sort variables in the grid | Sort the variables in the grid using an ascending or descending order for the individual columns. <br> 1. From the Application View, access the Dictionary instance for the required controller or program. <br> 2. In the variables grid, select the required column header. <br> An arrow showing the current order displays on the column header. <br> 3. Toggle the column header to switch between ascending and descending order. |
| Filter variables in the grid | When filtering, create a view displaying only the variables containing specified characters. <br> The filter row is the top row of the grid. Filter variables by typing alphabetical and numerical characters in the cells of the filter row. Also select from the drop-down combo box. Variables containing matching characters automatically display in the grid. <br> 1. From the Application View, access the Dictionary instance for the required controller or program. <br> 2. In the filter row of the variables grid, click the required cell, and then do one of these: <br> - Type the characters to use in the filtering operation <br> - Select the required defined word from the drop-down combo box |
| Add a variable to a variable group | In the variable grid, right-click the variable, point to Add to Variable Group, then click the required group. |
| Tip: Update each instance of a function or function block after adding, removing, renaming, changing data type, or changing direction of parameters in all POUs using the instance. Identify these POUs with the Cross Reference Browser, then open each POU and reselect the modified instance using the Block Selector. |  |

For variables of controllers or programs, the properties are:

| Column | Description | Possible Values |
| :--- | :--- | :--- |
| Name | Name of the variable | Limited to 128 characters beginning with a letter or single <br> underscore character followed by letters, digits, and single <br> underscore characters. The last character for a variable name <br> must be a letter or digit; variable names cannot end with an <br> underscore character. Do not use reserved words, defined <br> words, or data types (elementary, structures, or arrays) for <br> names. Names must be unique for the same type of elements <br> within a scope. |
| Logical Value | Available when online. The display value depends on the <br> direction of the variable. In the dictionary view, the columns <br> display values for the variables used in controllers where <br> debugging is enabled. | Input: Locked <br> Output: Updated by the running TIC code <br> Internal: Locked |

Chapter 2 Working with AADvance Applications

| Column | Description | Possible Values |
| :---: | :---: | :---: |
| Physical Value | Available when online. The display value depends on the direction of the variable. In the dictionary view, the columns display values for the variables used in controllers where debugging is enabled. | Input: Updated by the field value <br> Output: Locked <br> Internal: Updated by the running TIC code |
| Lock | Available when the application is connected. The value of the variable is locked. Enables locking only I/O variables. Locking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked directly. For structure and array elements, locking an element locks only that element of the structure or array. | Yes or No |
| Data Type | Data type of the variable. Avoid using library data types. | BOOL, BYTE, DATE, DINT, DWORD, INT, LINT, LREAL, LWORD, REAL, SINT, STRING, TIME, UDINT, UINT, ULINT, USINT, WORD, user arrays, and structures |
| Dimension | Size (number of elements) of an array | For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements. <br> Tip: Dimension does not apply to function block instances. Avoid setting Dimension for function block instances to help prevent unexpected build results. |
| String Size | Maximum length for String-type variables | String capacity has a maximum of 255 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. The default value is 80 . |
| Initial Value | Value held by a variable when the virtual machine starts the execution of the controller | The initial value of a variable. Possible values are the default value, a value assigned by the user when defining the variable, or the value of the retain variable after the virtual machine has stopped. <br> Tip: Use a decimal value only. Hexadecimal values are not supported. |
| Direction | For local variables of a program organization unit (POU), indicates whether the variable is internal, an input, or an output. | Possible values are Var, Varlnput, or VarOutput. When wiring the local variable of a program, the direction changes to match the direction required by the I/O device. |
|  | For global variables of a controller, indicates whether the variable is internal, an input, an output, or a system variable. | Possible values are Var, VarInput, VarOutput, VarGlobal, or VarDirectlyRepresented. When wiring a variable, the direction changes to match the direction required by the I/O device. |
| Attribute | Read and write access rights of a variable | Possible values are Read, Write, or Read/Write. The possible values depend on the direction of the variable. When wiring a variable, the attribute changes to match the direction required by the I/O device. |
| Retained | The controller saves the value of the variable in non-volatile memory at each cycle. <br> When setting the Retained property for a variable with a structure data type, the property is set for the entire structure; individual structure members cannot be retained. <br> Limitations: <br> - Do not use the Retained property for safety-related settings. <br> - Do not set the Retained property on wired T9K_xx_FULL structures. | True or False When the processor battery voltage is low or no battery is fitted, setting Retained to True has no effect. |
| Comment | User-defined text | Maximum of 255 characters when using only ASCII characters. Maximum of 127 characters when using at least one non-ASCII character. |
| Alias | Any name (for use in POUs) | Limited to 128 characters consisting of letters, digits, and these <br>  l, and ~. |
| Wiring | (Read-only) Generated by the I/O wiring tool indicating the I/O channel wired to the variable | Uses the syntax of directly represented variables |
| Groups | Group name or "None" |  |
| Allow OPC Write | An external client can write to the variable. Only available for user-defined variables or user-defined instances of function blocks. | The default value is FALSE |


| Column | Description | Possible Values |
| :--- | :--- | :--- |
| Message True | Message defined for the TRUE value message | Limited to 65535 characters |
| Message False | Message defined for the FALSE value message | Limited to 65535 characters |

Customize the Dictionary environment by arranging the columns to display.

## See also

Quick Declaration of Variables on page 167
Arrays Grid on page 161
Structures Grid on page 162
Defined Words Grid on page 163
Dictionary on page 160

## Quick Declaration of Variables

The Quick Declaration dialog box enables simultaneously creating multiple local or global variables. Quick Declaration can be accessed using the Dictionary. A preview of the variable is available on the top-right of the Quick Declaration dialog box.


Attributes to configure for variables in Quick Declaration are:

| Property | Description |
| :--- | :--- |
| Numbering | The range of values for the variables. The digits option is set to auto by <br> default and can be changed to alter the quantity of displayed digits. |
| Name | The variable name is separated into a prefix and suffix. The prefix <br> appears before the number value and can contain letters, digits, and <br> single underscores. The suffix appears after the number value and can <br> contain letters, digits, and single underscores. Neither can contain two <br> consecutive underscores. |
| Attributes | These attributes are available: <br> Data type: Drop down combo box displaying the variable types. <br> Possible values are elementary IEC 61131-3 types (BOOL, BYTE, DATE, |
|  | DINT, DWORD, INT, LINT, LREAL, LWORD, REAL, SINT, STRING, TIME, |
| UDINT, UINT, ULINT, USINT, or WORD) or derived types (arrays, |  |
| structures, or function blocks). |  |
| Direction: Indicates whether the variable is internal, input, or output. |  |
| Possible values are Var, Varlnput, or VarOutput. |  |
| String Length: Defined length only applying to the STRING variable. |  |
| Possible values are 1 to 255. |  |

## Cross Reference Browser

## To create multiple variables using Quick Declaration

1. In the required dictionary instance, right click an empty row, and then click Quick Declaration.
2. Configure the variable attributes in the Quick Declaration dialog box, and then click OK.

## To access the Cross Reference Browser

From the View menu, click Cross Reference Browser (or press Ctrl+W, Ctrl+C).
Use the Cross Reference Browser for an overview of the variables, programs, functions, function blocks, and defined words existing in a project including information like names, various properties, location of usage, and comments. When locating items in the browser, select a context view for the type of elements to locate. Find specific elements by name or filter the element list. In the browser, some columns from the different context views reflect the respective properties of the items. Sort the items in the list according to the different column headings in ascending or descending order.

The Cross Reference Browser toolbar contains:

| Icon |  |  | Description <br> Locates elements within the element list. Type the element name in the field or select a previous search from the drop-down combo-box. |
| :---: | :---: | :---: | :---: |
| Find |  |  |  |
| $\begin{array}{\|l\|} \hline \text { 瞫 } \\ \hline \end{array}$ |  |  | Locates the element specified in the Find field within the element list |
| <Type Filter keyword> |  | - | Filters the element list using text. Type element name in the field or select a previous search from the dropdown combo-box. To remove a filter, select <Remove Filter> from the drop-down combo-box. |
|  | $\checkmark$ |  | Selects a context view for the element list. Possible views include Variables, Programs, Functions, Function Blocks, Defined Words, and Operators. <br> Tip: - The Function Blocks context view displays references of non-user-declared instances of function blocks. <br> - The Variables context view displays references of user-declared instances of function blocks, program variables, global variables, and local variables of functions for which the Direction value is Var. <br> - The Operators context view is always empty. |
| $Q$ |  |  | Refreshes the element list to include the latest elements from the project. |
|  |  |  | Displays the previous instance of the selected element within the project |
| $\begin{array}{ll} \hline-9 \\ \hline \end{array}$ |  |  | Displays the next instance of the selected element within the project |

In the Cross Reference Browser, navigate using keyboard and mouse controls.

| Keyboard/Mouse Controls | Description |
| :--- | :--- |
| Arrow keys | Moves up, down, left and right within the rows of the elements list |
| Tab key | Moves left and right within the fields and commands of the toolbar. <br> Also moves from the elements list to the instances list. |
| Esc key | Moves from the Cross Reference Browser to the workspace |

Tip: When locating elements in the Cross Reference Browser, refresh the list of cross references by clicking (or pressing Ctrl+T, Ctrl+R). Rebuild the project before refreshing the list of cross references.

## To locate an element in the Cross Reference Browser

1. From the Context View drop-down list, select the type of elements to display.
2. (optional) To refine the element list, do one of the following:

- To sort the elements list, click the column heading by which to sort in an ascending order. Clicking twice sorts in descending order.
- To filter the elements list, type in the filter field or select a previous filter from the drop-down combo-box.

3. Select the required element by performing one of these:

- Type the name of the element in the Find field or select a previous search from the drop-down combo-box, then click 路。
- Scroll through the element list.

4. (optional) To jump to an instance of an element, double-click the required instance of the element (or press F8 or Shift+F8).
The program containing the instance of the element is displayed with the instance selected.

## See also

Variables Grid on page 164
Variable Dependencies on page 171

Defined Word Cross
References

In the element list, color is used to identify elements used within the project. Elements displayed as blue are used at least once within the project.
Elements displayed as red are not in use. For elements other than programs, select elements displayed as blue to view the location information for each instance of the element within the project. Jump to the location of individual instances by double-click in the instance list.

The properties for the Defined Words view are:

| Property | Description |
| :--- | :--- |
| Name | Name of the defined word |
| Equivalent | String replacing the defined word during compilation. For example, <br> the defined word "PI" is replaced by its equivalent "3.1459" |
| Project | Project using the defined word |
| Comment | Comment for the defined word |

## See also

## Cross Reference Browser on page 168

Function and Function Block Cross References on page 170
Program Cross References on page 170
Variable Cross References on page 171

Function and Function
Block Cross References

In the element list, color is used to identify elements used within the project. Elements displayed as blue are used at least once within the project.
Elements displayed as red are not in use. For elements other than programs, select elements displayed as blue to view the location information for each instance of the element within the project. Jump to the location of individual instances by double-click in the instance list.

Properties for the Functions and Function Blocks views are:

| Property | Description |
| :--- | :--- |
| Name | Name of the function or function block |
| Category | Type of function or function block. Possible types are standard, user- <br> defined, and native. |
| Language | Programming language of the function or function block |
| Project | Project using the function or function block |
| Controller | Controller using the function or function block |
| Comment | Comment of the function or function block |
| From Library | Library containing the function or function block |

## See also

## Cross Reference Browser on page 168 <br> Defined Word Cross References on page 169 <br> Program Cross References on page 170 <br> Variable Cross References on page 171

## Program Cross References

In the element list, color is used to identify elements used within the project. Elements displayed as blue are used at least once within the project. Elements displayed as red are not in use. For elements other than programs, select elements displayed as blue to view the location information for each instance of the element within the project. Jump to the location of individual instances by double-click in the instance list.

Properties for the Programs view are:

| Property | Description |
| :--- | :--- |
| Name | Name of the program |
| Language | Programming language of the program |
| Project | Project using the program |
| Controller | Controller using the program |
| Comment | Comment of the program |

## See also

## Cross Reference Browser on page 168

Defined Word Cross References on page 169

## Function and Function Block Cross References on page 170

Variable Cross References on page 171

## Variable Cross References

In the element list, color is used to identify elements used within the project. Elements displayed as blue are used at least once within the project. Elements displayed as red are not in use. For elements other than programs, select elements displayed as blue to view the location information for each instance of the element within the project. Jump to the location of individual instances by double-click in the instance list.

Properties for the Variables view are:

| Property | Description |
| :--- | :--- |
| Name | Name of the variable |
| Scope | Range of accessibility of a variable in relation to program <br> organization units (POUs) of a controller |
| Alias | Alias name of the variable |
| Type | Data type of the variable |
| Project | Project using the variable |
| Controller | Controller using the variable |
| Comment | Comment of the variable |
| Group | Group containing the variable |

## See also

Cross Reference Browser on page 168
Defined Word Cross References on page 169
Function and Function Block Cross References on page 170
Program Cross References on page 170
View the dependencies of a variable in both ascending and descending directions. Ascending dependencies display the variables affecting the variable while descending dependencies display the variables affected by the variable. These dependencies display as structures leading to the right for ascending dependencies and to the left for descending dependencies. When viewing the dependencies of a variable, the variable identification indicates its source: ProjectName.ControllerName.VariableName.

View dependencies of variables while editing or connected. While connected, monitor and force the values of variables from the dependencies.

This example shows the dependencies of the Alarm_Memo variable where the variables on the right, ascending, affect its value while the variable affects the values of the variables to the left, descending.


## To view variable dependencies

- From the language editor, dictionary instance, or Cross Reference Browser, right-click the variable, and then select Dependencies.

Variable dependencies display in the dependencies window.
Tip: Using declared function block instances or structure members may not display all dependencies in the dependencies window.

Perform these tasks in the dependencies window:

| Task | Procedure |
| :--- | :--- |
| Display the dependencies of a <br> variable within the dependencies <br> structure | - From the dependencies structure for a variable, double-click the variable from the ascending or descending structure for <br> which to display the dependencies. |
| Add a variable to a spy list | • From the dependencies structure for a variable, right-click the variable to add to the spy list, and then click Add to Spy List. |
| Force the value of a variable | 1. From the dependencies structure for a variable, select the variable for which to force the value, right-click, and then click <br> Write Variable. |
| 2. In the Write Logical Value dialog box, write the value for the variable. <br> 3. To lock the value for the variable, click Lock. <br> 4. Click Write. |  |
| Access the dictionary instance <br> containing a variable | From the dependencies structure for a variable, select the variable for which to access the dictionary instance, right-click, <br> and then click Variables. |
| Set the zoom of a dependencies <br> window | From the dependencies window for a variable, slide the zoom scale to the required magnification factor. |

## See also

Cross Reference Browser on page 168
Controller Status on page 172
Language Editor on page 26
Dictionary on page 160

## Controller Status

Access real-time status information for all controllers in a project:

| Name | Name of the controller. |
| :--- | :--- |
| Status | Status of the controller: <br> $\bullet$ - Building, indicates that the project or controller build is in progress <br> - Unable to Connect, indicates that the AADvance-Trusted SIS Workstation software <br> is unable to connect with the controller <br> - Need Password, indicates that the AADvance-Trusted SIS Workstation software <br> requires the controller password to connect and provide status information <br> - Need Save Status, indicates that the project requires saving <br> - Unavailable, indicates switching from design to online or simulation mode <br> - Simulator Running, indicates that the AADvance-Trusted SIS Workstation software <br> is unable to connect with the controller since it is in simulation mode |
|  | - Connecting, indicates that the AADvance-Trusted SIS Workstation software is <br> retrieving status information or is unable to communicate with the controller <br> - Running, indicates that the controller is running an application while displaying <br> the application version and date |
| • Stopped, indicates that the controller is not running an application |  |
| Controller statuses also display with a system health condition: Healthy, Faulted, or |  |
| Warning. |  |

When viewing the Controller Status, the Name column displays icons that indicate the connection status of the controllers.

|  | Indicates a healthy, running controller having a good system health |
| :--- | :--- |
|  | Indicates a faulted controller having a bad system health |
| D | Indicates any other condition |

## To access controller status information

- From the View menu, click Controller Status.


## See also

Front Panel Status Indicators on page 173

## Front Panel Status Indicators



The front panel of the $T 9110$ processor module displays these status indicators:

| Indicator | Status | Description |
| :--- | :--- | :--- |
| Healthy |  | Indicates the fault status of the module and the power-on or operational status. |
|  | OFF | No power |
|  | RED | The module has a fault or the controller is offline. <br> The Healthy LED briefly flashes RED while booting up after installation or when a reset is in progress. |
|  | GREEN | During installation, the Healthy LED briefly flashes RED when receiving power, and then turns GREEN as the module <br> boots up. The LED remains GREEN indicating the module is operational and contains no hardware faults. <br> When in recovery mode and no faults are present, the LED is GREEN. <br> Tip: If the Healthy LED is GREEN and all other status indicators are OFF, the <br> module has failed to boot up. <br> If the Healthy LED is GREEN and the Ready and Run LEDs are RED, the <br> module is in the shutdown state. |
| Ready |  | Indicates the education and synchronization status of the module. |

Chapter 2 Working with AADvance Applications

| Indicator | Status | Description |
| :---: | :---: | :---: |
|  | OFF | No power |
|  | RED | The module is booting up, not educated, or not synchronized with partners. |
|  | GREEN | The module is educated and synchronized. |
|  | FLASHING GREEN | Education or synchronization of the module is in progress. |
|  | AMBER | The module is in recovery mode. |
| Run |  | Indicates the application status of the module. The application status should be identical for all educated and synchronized processors. |
|  | OFF | No power. The LED also remains OFF while the module boots up. |
|  | RED | The module is not educated, not synchronized, or no application is loaded. The LED is also RED when the processor module is in recovery mode and the base-level firmware is running. |
|  | GREEN | The application is running. |
|  | AMBER | The module is in recovery mode or the application data may be present but not running; press Fault Reset to run the application. |
| System Healthy |  | Indicates the global health of the system, including all processors and I/0 modules. All educated and synchronized processors must indicate the same status. |
|  | OFF | No power. The LED also remains OFF while the module boots up. |
|  | RED | System or module faults are present. The LED is also RED when the application stops running due to the module being in recovery mode. |
|  | GREEN | No system or module faults are present during normal operations or when in recovery mode. |
| Force |  | Indicates variables are being locked or forced by the application. All educated and synchronized processors indicate the same status. |
|  | OFF | No power. The LED also remains OFF while the module boots up. |
|  | GREEN | No variables are being locked or forced. |
|  | AMBER | The module is in recovery mode or at least one variable of an operating controller is being locked or forced. |
| Aux |  | The application controls the Aux indicator. The application can display any status except when in recovery mode and the application is stopped. |
|  | OFF | No power. The LED also remains OFF while the module boots up or when the application is in control of the indicator. |
|  | GREEN | The application is in control of the indicator. |
|  | AMBER | The module is in recovery mode or the application controls the indicator. |
| Serial 1 and 2 |  | Indicates the serial port activity. |
|  | OFF | No power. The LED also remains OFF while the module boots up. |
|  | RED | Pulse stretched transfer (Tx) |
|  | GREEN | Pulse stretched receive (Rx) |
|  | AMBER | The transfer ( $T x$ ) and receive ( Rx ) activity is in close proximity. |
| Ethernet 1 and 2 |  | Indicates the Ethernet port activity. |
|  | OFF | No power. The LED also remains OFF while the module boots up. |
|  | GREEN | An Ethernet link is present. |
|  | AMBER | Transfer ( Tx ) or receive (Rx) activity. |

## See also

## Controller Status on page 172

## Communication View

To access the Communication View
From the View menu, click Communication View.
The Communication View is a graphical environment that displays aspects of communication information related to communication events. The Communication View displays a tree-like structure with nodes that expand to show the available communication functions and configurations for the series 9000 equipment.

From the Communication View, use these protocols with Ethernet networks and serial ports:

- Modbus RTU/TCP
- OPC
- SOE
- SNTP
- Serial Tunneling using TCP/IP
- SNCP bindings
- CIP
- Peer-to-Peer
- Quality of Service

In the Communication View, select tree nodes to access related communication options for the equipment series. These options display in the property pages. When working in the Communication View, use the mouse controls or keyboard arrows to navigate. View additional properties for selected elements in the Properties window.

## To refresh the Communication View

Refresh the Communication View to display the current version control and debug states.

- Select the Communication View, then from the View menu, click Refresh.


## See also

Application View on page 24
Equipment View on page 231

## Defining Ethernet Ports

The AADvance controller provides up to six Ethernet communication ports, two for each 9110 processor module present. Each pair of Ethernet ports is identified as $\mathrm{E} n-1$ and $\mathrm{E} n-2$ where $n$ indicates the processor module.

- [E1-1 and E1-2]
- [E2-1 and E2-2]
- [E3-1 and E3-2]

The En-1 Ethernet ports are all on one network and the En-2 ports are on a second network.

Tip: To avoid unexpected ARP requests, it is recommended to configure IP addresses for En-2 ports (even if they are unused).

When defining Ethernet ports, indicate an IP address and optionally a comment. Each controller on a local area network must have a unique IP address. Set the IP address when creating a system or fitting a new processor base unit. The IP address must be configured in the physical controller by using AADvance Discover and must then be configured in the AADvance controller.

Tip: Change an IP address in the Communication property page only. Changing an IP address in the Properties window does not update the IP address in Communication View.

## To define Ethernet ports

1. From the AADvance-Trusted SIS Workstation software, open or create the required project.
2. From the View menu, click Communication View.
3. In the Communication View, right-click Ethernet, and then click Open.


The Communication property page displays with the six Ethernet Ports.
4. In the Communication property page, type the IP address for each required port.
Tip: $\quad$ Ports that are Not Configured can be configured individually by selecting the required port in the Communication View.

## See also

Specifying the Workbench and OPC Communications on page 191
The AADvance controller supports Modbus ${ }^{\circledR}$ communications and can act as a Modbus master or Modbus slave. A Modbus master can have Modbus master slaves. The AADvance controller can use serial or Ethernet ports for Modbus communications.

Tip: The AADvance controller does not support the Modbus ASCII protocol.

The Modbus functionality implemented by the AADvance controller meets this standard: Modbus Application Protocol Specification, version 1.1b. December 2006. The Modbus Organization.

To use Modbus communications, Modbus variables for the controller must be mapped. These variables are then used by Modbus slaves, Modbus masters, and Modbus master slaves. When mapping variables, these are mapped as the following types:

- Coils
- Discrete Inputs
- Input Registers
- Holding Registers

Tip: $\quad 64$-bit variables are not supported in Modbus communications. Use 32-bit variables.

The Modbus mapping includes the following properties:

| Name | The name of the variable. |
| :--- | :--- |
| Data Type | The data type of the variable. |
| Direction | The direction of the variable. |
| Base Address | The base address of the variable. Possible values range from 1 to 65536; the <br> default starting address value is 1. |
| Allow Modbus Write | For Coils and Holding Registers, indicates whether data can be copied from the <br> Modbus master to the slave device. |

Modbus uses a numeric addressing scheme to move data between devices. AADvance controllers provide a dedicated area for each of the four variable types. The original Modbus standard defines the address field as a four-digit field with a prefix relating to the variable type. The crosshatch areas in the following example display how the original-style, five-digit Modbus addresses
(for example, a holding register at 40,001 ) are related to the AADvance memory map.


The base addresses used for Modbus data transfer listings start at one. The first variable network address of the AADvance controller is 1 and the first coil is 00001 .

When managing Modbus variables, perform these tasks in the Modbus element:

| Task | Procedure |
| :---: | :---: |
| Add a Modbus variable | 1. In the Communication View, double-click the Modbus element. <br> The Communication property page displays. <br> 2. In the property page, select the tab in which to map the variable, and then click Add Variable. <br> 3. From the Variable Selector, locate the variable to use for the Modbus ${ }^{\ominus}$ mapping, and then click OK. <br> 4. In the property page, enter the required value for the Base Address property. <br> 5. For Coils and Holding Registers, set the Allow Modbus Write property to the required value. <br> Tip: If the project contains two or more controllers, simulating the project and then stopping the simulation causes the Communication View to be readonly. To edit Modbus variables after stopping the simulation, close and then reopen the project. |
| Renumber base addresses of Modbus variables | Renumber the base addresses of Modbus mapped variables. Base address values are assigned in the order of appearance in the list of mapped variables. Base addresses range from 1 to 65536; the default starting address value is 1 . <br> Tip: Manually modify the base address of Modbus variables to the required values. <br> In the Communication View, double-click the Modbus element. <br> The Communication property page displays. <br> 1. In the property page, select the tab having the required mapped variables. <br> 2. In the list of mapped variables, select and right-click the variables to renumber, and then click Renumber. <br> 3. In the Renumber Addresses dialog box, specify the value to use for the base address of the first selected variable, and then click OK. |
| Allow the Modbus write attribute for Modbus variables | Allow the Modbus write attribute for coils and holding registers mapped variables. <br> 1. In the Communication View, double-click the Modbus element. <br> 2. In the Communication property page, select the tab having the required mapped variables. <br> 3. In the list of mapped variables, select and right-click the variables for which to allow the Modbus write attribute, and then click Allow Modbus Write. |


| Task | Procedure |
| :--- | :--- |
| Disable the Modbus write attribute on <br> Modbus variables | Disable the Modbus ${ }^{\circledR}$ write attribute for mapped variables. The Modbus write attribute is only available for coils and holding <br> registers variables. <br> 1. In the Communication View, double-click the Modbus element. <br> 2. In the Communication property page, select the tab having the required mapped variables. <br> 3. In the list of mapped variables, select and right-click the variables for which to disable the Modbus write attribute, and <br> then click Disable Modbus Write. |
| Remove Modbus variables | 1. In the Communication View, double-click the Modbus element. <br> 2. In the Communication property page, select the tab from which to remove mapped variables. <br> 3. In the list of mapped variables, select the variables to remove, and then click Remove Selected. |
| Delete a Modbus object | • In the Communication View, right-click the Modbus slave, master, or master slave, and then click Delete. |

Also configure Modbus Master and Slave protocols for use in the communication services for controllers.

## See also

Defining a Modbus Slave on page 180
Defining a Modbus Master on page 181
Defining a Modbus Master Slave on page 184
Defining Modbus Master Slave Messages on page 188
Simulate an application on page 137

Defining a Modbus Slave

The AADvance controller can operate as a Modbus ${ }^{\circledR}$ slave, supporting up to a maximum of 10 Modbus slaves on each processor module. A controller with three processor modules supports up to 30 Modbus slaves.

Tip: As a Modbus slave device, the controller only transmits data upon a request
from a Modbus master and does not communicate with other slaves.
When configuring a controller as a Modbus slave, select whether the slave uses a serial or Ethernet connection. The connection type affects the slave properties like the available protocols and ports.

## To define a Modbus slave

1. In the Communication View, expand the required controller, and then expand the Modbus element.
The MODBUS Slave and MODBUS Master elements are displayed.
2. Right-click the MODBUS Slave element, and then click Add New Slave.
3. In the Communication View, double-click the slave instance, then in the Communication property page, select the connection type and define the required properties.

## See also <br> Using Modbus Communications on page 178 <br> Defining a Modbus Master on page 181

## Serial Slave Connection

A processor has a maximum of two Modbus slaves using a serial connection, one per serial port. A controller with three processor modules supports up to six Modbus slaves using serial ports.

| Serial Port | The serial port used for Modbus ${ }^{\oplus}$ communications. Possible value is one of the <br> following: S1-1, S1-2, S2-1, S2-2, S3-1, or S3-2. |
| :--- | :--- |
| Name | Name of the Modbus slave. |
| Protocol | When using a serial port, the controller supports Modbus RTU. |
| ID | Represents the Slave ID for a serial connection. Possible values range from 1 to 247; <br> the default value is 1. |
| Port | For serial connections, the Port field is disabled. |
| Comment | Free-formatted text |

## See also

Defining a Modbus Slave on page 180
Ethernet Slave Connection on page 181

## Ethernet Slave Connection

A processor has a maximum of 10 Modbus slaves using an Ethernet connection. A controller with three processor modules supports up to 30 Modbus slaves using Ethernet ports.

| Ethernet Port | The Ethernet ports used for Modbus® communications. Possible value is one of these <br> slices: E1-1 and E1-2, E2-1, and E2-2, or E3-1 and EJ-2. |
| :--- | :--- |
| Name | Name of the Modbus slave. |
| Protocol | When using an Ethernet connection, the controller supports Modbus RTU and TCP; the <br> default value is RTU. |
| ID | Ethernet connections require a port. The Slave ID is 255. |
| Port | Possible values range from 1 to 65535. For the RTU protocol, the default value is 2000. <br> For the TCP protocol, the default value is 502. Make sure the Port field matches the port <br> expected by the Modbus master. <br> Modbus slaves having the same Modbus master must use unique port numbers. |
| Comment | Free-formatted text |

## See also

## Defining a Modbus Slave on page 180

## Serial Slave Connection on page 181

The AADvance controller can fulfill the role of a Modbus ${ }^{\circledR}$ master to one or more Modbus slave devices. Slave devices can include programmable logic controllers, remote devices (typically with little or no processing capability) and, more rarely, other functional safety controllers (Trusted ${ }^{\circledR}$ or AADvance). A controller can have a maximum of 32 Modbus masters.

The AADvance controller supports the Modbus TCP protocol, the Modbus RTU protocol, and a subset of Modbus commands. Each Modbus master can connect to only one Modbus TCP slave device or to one or more Modbus RTU slave devices. Use Modbus TCP with Ethernet ports and Modbus RTU with point-to-point or multi-drop serial links. Serial links can use a full-duplex, 4wire connection, or a half-duplex, 2-wire connection.

To use the AADvance controller as a Modbus master, perform the following:

- Define the physical connections from the controller to the slave devices
- Configure the characteristics for serial ports
- Set up the project for Modbus master and configure the application

The Modbus master functionality is built into the 9110 processor module and the physical communication ports are located on the 9100 processor base unit.

This example shows possible arrangements for Modbus master connections:


In this example, the Modbus RTU slave devices are connected to one or more serial ports on the controller; a common arrangement uses multi-drop (RS485). In the example, the engineering workstation and the Modbus TCP slave devices are connected to Ethernet ports on different networks. These devices can also be combined to use the same network.

Set up an individual list of messages (commands) for each slave device. Modbus read commands cause data to be read from the slave device to the Modbus master, while Modbus write commands cause data to be copied from the Modbus master to the slave device. Also define a sequence of broadcast write commands, which a Modbus master can send to multiple Modbus RTU slaves without requiring an acknowledgment. The AADvance controller can control and monitor individual Modbus master objects and their slave links.

When selecting a serial or Ethernet connection type, select the required port. A Modbus master uses one of two serial ports related to a processor module. When choosing an Ethernet connection, the master uses both ports related to the processor module (for example, the processor slice). Different Modbus masters can use different connection types and ports.

A processor module failure also causes the failure of the Modbus communications allocated to that processor. Communication with slaves allocated to different processor modules are unaffected.

Define these properties to configure a Modbus Master:

| Property | Description |
| :---: | :---: |
| Master Connection | Indication of whether the master uses a Serial or Ethernet connection. The default connection type is Serial. |
| Port | The serial or Ethernet port used for Modbus communications. <br> For a serial connection, the possible value is one of the following: S1-1, S1-2, S2-1, S2-2, S3-1, or S3-2. <br> For an Ethernet connection, the possible value is one of the following slices: E1-1 and E1-2, E2-1, and E2-2, or E3-1 and E3-2. |
| Name | Name of the Modbus master. Possible values are any combination of alphanumeric characters and punctuation including spaces. |
| Protocol | When using a serial port, the controller supports Modbus RTU. <br> When using an Ethernet connection, the controller supports Modbus TCP. |
| Timeout (ms) | The period of time the Modbus master waits for a response from its slave devices before retrying or assuming that the slave is unavailable. The timeout is common to all slaves configured to a particular master. The Modbus master uses the timeout for each of its configured slaves to determine whether they are still communicating. If a slave does not respond within this period (and a specific number of retries), it is deemed to have failed. The number of retries is defined for individual slave devices. <br> Possible values are 0 to 60000 milliseconds; the default value is 1000 ms . To avoid unexpected behaviors in the processor when using Modbus TCP communications, wait at least 100 ms before attempting to reconnect to the slave device. |
| Control Address | The address of a control register. When using a control variable, the Modbus address of the holding register. The application may write the control variable to switch the state of the master. <br> 0 = inactive, the Modbus master is disabled; the link is inactive, there is no communication activity, and all slaves are inactive. <br> 1 = standby, the Modbus master is forced to operate in standby mode; the link is active but no data is transferred. <br> 2 = active, the Modbus master is forced to operate in active mode; the link is active and transfers data. <br> Possible values are 1 to 65536 , or 0 to disable; the default value is 0 . |
| Status Address | The address of a status register (a holding register). The status register, is a UINT returned by the Modbus driver to let the application monitor and act on Modbus master faults. <br> $0=$ healthy, the Modbus master is operating normally; the link is active and no errors are reported. $1 \text { = initializing }$ <br> $2=$ error, the Modbus master has an error and is disabled, or is unable to make a link. <br> Possible values are 1 to 65536 , or 0 to disable; the default value is 0 . |
| Message Wait Interval (ms) | Enables AADvance to support legacy Modbus slave devices having slow communication response times. Only serial RTU links are supported. Possible values range from 0 to 65535 milliseconds; the default value is 100 ms. |
| Statistics Mode | Indicates the statistics reporting mode for the master. Possible values are: Disabled, the default reporting mode. <br> Last Rate, reports the length of the most recent scan time. <br> Maximum Rate, reports the longest scan time. <br> Average Rate, reports the average scan time. <br> Statistics are reported in hundredths of a second. <br> For the Last Rate, Maximum Rate, and Average Rate reporting modes, specify the Data Address and Reset Address properties. |


| Data Address | The holding register of the data variable. The data variable is reported in <br> hundredths of a second. <br> Possible values are 1 to 65536, or 0 to disable; the default value is 0. |
| :--- | :--- |
| Reset Address | The holding register of the reset variable. Any non-zero value in the reset <br> variable resets the data variable to zero. <br> Possible values are 1 to 65536, or 0 to disable; the default value is 0. |
| Comment | Free-formatted text |

All Modbus masters defined for a project (Modbus RTU or Modbus TCP) function independent of each other. Each master polls its associated slave devices, to send the messages scheduled for the slaves.

If a Modbus master encounters communication errors to a slave device, the master suspends polling the slave for a length of time. The suspension enables polling other slaves on the same master without pausing for the communications timeout on each cycle. At regular intervals, the controller pings the non-communicating slave. When the slave responds to a ping, the polling of the slave restarts on the next cycle.

> Tip: If the controller receives error messages with the replies from a slave, the slave remains in the polling cycle since the controller does not treat error messages as a communication failure.

## To define a Modbus Master

1. In the Communication View, expand the required controller, and then expand the MODBUS element.

The MODBUS Slave and MODBUS Master elements are displayed.
2. Right-click the MODBUS Master element, and then click Add New Master.
3. Double-click the MODBUS Master instance.

The Communication property page is displayed with the Master properties.
4. In the property page, select the required Master Connection and Port, and then define the required properties.

## See also

## Using Modbus Communications on page 178

Defining a Modbus Slave on page 180
Defining a Modbus Master Slave on page 184
Defining Modbus Master Slave Messages on page 188

Defining a Modbus Master Slave

Define Modbus ${ }^{\circledR}$ master slaves for existing Modbus masters. A controller and a Modbus master can have a maximum of 64 master slaves. When adding a master slave, this creates a link between the Modbus master and slave device. Each master slave device represents a server. When configuring a Modbus master, define general and messages properties.

Important: If the Modbus master is configured for Modbus TCP, only create one master slave link. If reconfiguring Modbus RTU master with multiple slave links to Modbus TCP, keep only one master slave and delete the other links.

Every Modbus master slave link contains a pair of holding registers: a control register and a slave register. The control register enables the application to control the link and port, while the status register retrieves the status information. Each register is found at a unique address and is defined using the Control Address and Status Address. Registers are optional. To use a register, declare a variable for the register in the application variables and the address must also be specified in the holding registers.

General properties are:

| Property | Description |
| :--- | :--- |
| Name | Name of the Modbus master slave. Possible values are any <br> combination of alphanumeric characters and punctuation <br> including spaces. |
| Slave ID | Unique identifier for the Modbus master slave. <br> When the protocol of the master is RTU, possible values range <br> from 1 to 247 ; the default value is 1. <br> When the protocol of the master is TCP, possible values range <br> from 1 to 255; the default value is 255. |
| Control Address | The address of a control register. When using a control variable, <br> the Modbus address of the holding register. If a control variable <br> address is unspecified, the link is automatically enabled once the <br> controller starts. The application may write the control variable to <br> switch the state of the master slave. <br> $0=$ inactive, the slave link is disabled and the slave is not polled or <br> pinged. <br> $1=$ standby, the slave is forced to operate in standby mode. <br>  <br> Modbus continues to ping the slave to ensure communications are <br> possible. <br> $2=$ active, the slave is forced to operate in active mode and is <br> polled for data. <br> Possible values are 1 to 65536, or 0 to disable; the default value is <br> 0. <br> The address of a status register la holding register). The status <br> register is a UlNT returned by the Modbus driver to let the <br> application monitor and act on faults. If a status register address |
| is not specified, the master slave link does not report the status |  |
| information to the application. |  |
| $0=$ healthy, the Modbus slave is operating normally; the link is |  |
| active and no errors are reported. |  |
| $1=$ initializing, the slave link is initializing. This value is also |  |
| reported if the slave device has failed to respond to a ping on a |  |
| standby link, if the Modbus master is in error, or if the master is |  |
| set to inactive. |  |
| $2=$ error, the last message to the slave resulted in an exception |  |
| response. |  |
| Possible values are 1 to 65536, or 0 to disable; the default value is |  |
| 0. | The minimum amount of time to wait from the start of one <br> message to the beginning of the next message. Possible values <br> range from 0 to $65535 ~ m i l l i s e c o n d s ; ~ t h e ~ d e f a u l t ~ v a l u e ~ i s ~$ <br> 100 |
| Wait Interval (ms) |  |


| Retries | The number of times the master tries to retransmit a message when receiving no response from the slave. This property is used in conjunction with the Modbus master Message Wait Interval value. <br> Possible values range from 0 to 10 ; the default value is 3 . |
| :---: | :---: |
| Slave IP | For TCP connections, the network address used by master slave. Possible values range from 0.0.0.0 to 255.255.255.255.255; the default value is 0.0 .0 .0 . |
| Slave Port | For TCP connections, the port number on which the slave device is offering services. <br> Possible values range from 0 to 65535; the default value is 502 . <br> Modbus slaves having the same Modbus master must use unique port numbers. |
| Ping Mode | Tests the communications between the master and slave and determines the action the master takes if no response is received from the slave after the specified number of Retries. <br> The AADvance controller supports a Function Code 08 ping mode for Modbus RTU slaves. Function Code 08 sends a diagnostic message to the operational slave. If supported by the slave, select Function Code 08 since it is a short message. Alternatively, the Read Holding Register mode reads data from the holding register. Also set a Do Not Ping setting for both protocols. <br> For Modbus RTU, the possible values are Do Not Ping, Function Code 08, and Read Holding Register. <br> For Modbus TCP, the possible values are Do Not Ping or Read Holding Register. Read Holding Register is the only method available to ping a Modbus TCP slave. <br> When the Ping Mode is Do Not Ping or Function Code 08, the Ping Address property is disabled. <br> It is not recommended to use the Do Not Ping mode since the master does not look for the slave after disconnection. When using Do Not Ping, communications only restart when there is a change to the control register of the master or slave. |
| Ping Interval | The maximum time between each initiation of a Read Holding Register or Function Code 08 diagnostic message and specified individually for each master slave. <br> Possible values range from 0 to 65535 milliseconds; the default value is 5000 ms . |
| Ping Address | For use with the Read Holding Register ping mode and specified individually for each master slave; it is the Modbus address of the slave holding register. <br> Possible values range from 1 to 65536 , or 0 to disable; the default value is 0 . |
| Statistics Mode | Indicates the statistics reporting mode for the master slave. Possible values are: <br> - Disabled, the default reporting mode. <br> - Last Rate, reports the length of the most recent scan time. <br> - Maximum Rate, reports the longest scan time. <br> - Average Rate, reports the average scan time. <br> Statistics are reported in hundredths of a second. <br> For the Last Rate, Maximum Rate, and Average Rate reporting <br> modes, specify the Data Address and Reset Address properties. |
| Data Address | The holding register of the data variable. The data variable is reported in hundredths of a second. <br> Possible values are 1 to 65536 , or 0 to disable; the default value is 0. |
| Reset Address | The holding register of the reset variable. Any non-zero value in the reset variable resets the data variable to zero. <br> Possible values are 1 to 65536 , or 0 to disable; the default value is 0. |

## To define a Modbus master slave

1. In the Communication View, expand the required controller, and then expand the MODBUS element.
The MODBUS Slave and MODBUS Master elements are displayed.
2. Expand the MODBUS Master element, right-click the existing Modbus master, and then click Add New Master Slave.
3. Double-click the master slave instance, and in the Communication property page, define the required properties and messages:
a. On the General tab, define the required general properties.
b. On the Messages tab, specify the messages for the slave.

## See also

Using Modbus Communications on page 178
Defining a Modbus Master on page 181
Defining a Modbus Slave on page 180

Modbus Statistics

The AADvance controller has a cycle timer and counters to generate statistics for each slave. The statistics are available to the application, enabling to react accordingly or report that a link is operating correctly. These statistics are also used by diagnostic commands. The application reads the statistics as variables (the address of each variable must be specified) and also resets the statistics to zero. Statistics are only available for point-to-point message transfers.

During operation, the statistics are updated on the first message of each slave and on the first slave linked to each master to provide the slave and master cycle times. The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software also verifies the application interface to see if a statistics reset is required.

Meanwhile, the Modbus ${ }^{\circledR}$ master measures the overall scan time. A scan measures the time from the start of the first slave until looping back to the first slave, and then a new scan starts. The statistics are updated on the first message of each slave and on the first slave linked to each master to provide the slave and master scan times. The AADvance-Trusted SIS Workstation software also verifies the application interface to see if a statistics reset is required.

These application variables also supply diagnostic information, including the status of the communication link and the Modbus slave devices connected to the link.

## See also

Defining a Modbus Master Slave on page 184
Modbus Exception Responses on page 188

Modbus Exception The AADvance controller uses the absence of a response from a slave device Responses to identify an error on a link between a master and slave. When the AADvance controller operates as a slave, in some cases the controller reports the following exception codes through the application:

| Code | Name | Description |
| :--- | :--- | :--- |
| 01 | Illegal Function | The function code received in the query is not a permitted action for the slave. Code 01 represents an unknown or <br> unsupported function in the AADvance controller. <br> If a Poll Program Complete command was issued, this code indicates that no program function preceded the <br> command. |
| 02 | Illegal Data Address | The data address received in the query is not a permitted address for the slave. The AADvance controller raises <br> Code 02 when a request specifies an address outside of the 0 to 65536 <br> 16-bit range. |
| 03 | Illegal Data Value | A value contained in the query data field is not a permitted value for the slave. Code 03 is only raised for boolean <br> coil writes. |
| 04 | Slave Device Failure | An unrecoverable error occurred while the slave was attempting to complete instructions. Code 04 represents an <br> internal error in the AADvance controller. |
| 06 | Slave Device Busy | The slave is processing a long-duration program command. The master must retransmit the message once the <br> slave is free. When the AADvance controller is busy, Code 06 is raised while the controller waits for the application <br> to download or start. The controller can report itself to be busy. |

The Modbus ${ }^{\circledR}$ protocol allows these errors to pass by returning an error frame to the master. The error frame consists of the originally requested function code with the high bit set and a data field with the error code. The Modbus master locates these codes and considers them general errors to the slave device. The slave status variable is set to Slave Error, enabling the application to respond to the exception.

## See also

## Modbus Statistics on page 187

Defining a Modbus Master Slave on page 184

## Defining Modbus Master Slave Messages

Set up an individual list of messages (commands) for each slave device.
A Modbus ${ }^{\circledR}$ master supports a maximum of 400 messages from slave devices. A Modbus master Modbus read commands cause data to be read from the slave device to the Modbus master, while Modbus write commands cause data to be copied from the Modbus master to the slave device. Define a sequence of broadcast write commands, which a Modbus master can send to multiple Modbus RTU slaves without requiring an acknowledgment. The AADvance controller can control and monitor individual Modbus master objects and their slave links.

| Task | Procedure |
| :--- | :--- |
| Add messages for a Modbus master <br> slave | 1. In the Communication View, expand the required Modbus master, and then double-click the required Modbus master slave. <br> 2. The Communication property page is displayed with the General and Messages tabs. <br> 3. In the property page, click the Messages tab. <br> 4. To add a message, click Add Message, and then specify the message properties. <br> 5. To change message order, select the message to move, and then click Move Up or Move Down. |
| Remove messages from a Modbus <br> master slave | 1. In the Communication View, expand the required Modbus® master, and then double-click the required Modbus master slave. <br> The Communication property page is displayed with the General and Messages tabs. |
|  | 2. In the property page, click the Messages tab. <br> 3. To delete messages, select the required messages in the grid, and then click Remove Selected. |

Chapter 2 Working with AADvance Applications
Properties for adding messages are:

| Command | Read Coils | Reads the coil status in a remote slave device. Each message has a maximum Count of 2000 coils. |
| :---: | :---: | :---: |
|  | Read Discrete Inputs | Reads the discrete input (coil) status in a remote slave device. Each message has a maximum Count of 2000 inputs. |
|  | Read Holding Registers | Reads the holding registers in a remote slave device. Each message has a maximum Count of 125 registers. |
|  | Read Input Registers | Reads the input registers in a remote slave device. Each message has a maximum Count of 125 registers. |
|  | Write Coils | If the Master Space is Coils, forces each coil in a sequence of coils on and off in a remote slave device. <br> If the Master Space is Discrete Inputs, writes to each discrete input in a remote slave device. <br> Each message has a maximum Count of 2000 coils or inputs. |
|  | Write Holding Registers | Writes to the holding or input registers. Each message has a maximum Count of 125 registers. |
| Master Space | Depending on the Command, possible values are Coils, Discrete Inputs, Input Registers, or Holding Registers. <br> - For Read Coils or Read Discrete Inputs, the value is Coils. <br> - For Write Coils, the value is either Coils or Discrete Inputs. <br> - For Read Input Registers or Read Holding Registers, the value is Holding Registers. <br> - For Write Holding Registers, the value is either Input Registers or Holding Registers. |  |
| Slave Address | Possible values are 1 to 65536; the default value is 1 . |  |
| Master Address | Possible values are 1 to 65536; the default value is 1 . |  |
| Count | When the Command is Read Coils, Read Discrete Inputs, or Write Coils, the value ranges from 1 to 2000; the default value is 1 . When the Command is Read Input Registers, Read Holding Registers, or Write Holding Registers, the value ranges from 1 to 125 ; the default value is 1 . <br> Tip: Use up to 31 messages. Using more than 31 messages does not send any messages to the controller after downloading code to the controller. |  |
| Control | The address of a control register. When using a control variable, the Modbus address of the coil register. Possible values range from 1 to 65536 , or 0 to disable; the default value is 0 . When the address of a control variable is unspecified, the link is automatically enabled once the controller starts. The application may write to the control variable to switch the state of the master slave. <br> True $=$ The slave is not polled <br> False = The slave is polled for data |  |
| Comment | Free-format text |  |

All Modbus master objects defined for a project (Modbus RTU or Modbus TCP) function independent of each other. Each master polls its associated slave devices, to send the messages scheduled for the slaves.

For Modbus RTU, a master polls its slave devices one at a time, executing one message for each slave in turn. Different slaves attached to the same master usually need a different numbers of messages. To accommodate this, the master proceeds to work through the messages listed for the slaves, one message for each pass, returning to the first message for a slave after it has sent the last one. This means that some messages (for slaves with fewer scheduled messages) are sent more than once during a polling cycle. The polling cycle completes when the Modbus master has made an equal number of polls to each slave and has sent every message; the cycle then starts again.

## Example

This example displays the message schedule for a master with three slaves that require a different number of messages:


The total number of messages that the master must send in a complete polling cycle is equal to the largest number of messages scheduled for any slave, multiplied by the total number of slaves. In the previous example, the total number of messages is $4 \times 3=12$. Nine messages are specified for the three slaves. The message scheduling mechanism sends three of these messages twice, making a total of 12 messages sent. This table lists the sequence of messages for the serial communications link in the previous example:

| Sequence Number | Slave | Message | Remarks |
| :--- | :--- | :--- | :--- |
| 1 | 1 | 1 |  |
| 2 | 2 | 1 |  |
| 3 | 3 | 1 | End of first pass through all the slaves |
| 4 | 1 | 2 |  |
| 5 | 2 | 2 | End of pass through slave 2 |
| 6 | 3 | 2 |  |
| 7 | 1 | 3 |  |
| 8 | 2 | 1 | Repeat message |
| 9 | 3 | 3 | End of pass through slave 3 |
| 10 | 1 | 4 | End of pass through slave 1 |
| 11 | 2 | 2 | Repeat message |
| 12 | 3 | 1 | Repeat message; end of poll |

## See also

Defining a Modbus Master Slave on page 184
Defining a Modbus Slave on page 180
Defining a Modbus Master on page 181
Download code to controllers on page 129
Connect with controllers on page 134

## Specifying the Workbench and OPC Communications

For communication with the OPC ${ }^{\circledR}$ portal, the AADvance ${ }^{\circledR}$ - Trusted ${ }^{\circledR}$ SIS Workstation software uses an Ethernet port from those defined for the controller processors. By default, the first configured Ethernet port is selected for communications. Also select different ports. To enable Workbench and OPC communications, map variables for:

- OPC
- SOE


## To specify the Workbench and OPC communications

1. In the Communication View, expand the controller, and double-click Workbench \& OPC.

The Communication property page for the Workbench and OPC displays.
2. In the Communication property page, select a configured Ethernet port.

Workbench and OPC use the defined port for communications.

## See also

Mapping OPC Variables on page 191
Map SOE Variables on page 193
Defining Ethernet Ports on page 177
The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software enables the write access from the Workbench OPC portal without restricting access. OPC write for variables can only be allowed from the OPC-specific Communication property page. The Allow OPC Write command is only available for user-defined variables or user-defined instances of function blocks.

By default, the following options to map OPC variables are selected (True):

- Include Server Constants
- Include Server Functions
- Auto-include All Variables

When Auto-include All Variables is selected, all of the controller's variables are added to the list of mapped variables.


## To map OPC variables

1. In the Communication View, expand the required controller, and then expand the Workbench \& OPC element.
2. Double-click OPC.

The Communication property page displays the variables mapped for OPC ${ }^{\circledR}$ communication.
3. In the Communication property page, select whether to include server constants and include server functions.

- To automatically include all variables for the OPC service, select Autoinclude All Variables. Also choose to add all of a controller's variables by clicking Add All Variables.
- To add individual variables to the list, click Add Variable, and then choose the required variable from the Variable Selector.
Tip: After modifying a variable, close and then reopen the OPC Communication property page.
- To remove a variable, select a variable from the list, and then click Remove Selected.
- To toggle the OPC write property for variables, in the Allow OPC Write column of the visible window, select the required variables.


## See also

Specifying the Workbench and OPC Communications on page 191
Mapping OPC Variables on page 191

## Map SOE Variables

The AADvance controller supports a Sequence of Events (SOE) service for Boolean variables. When an SOE service is defined, define SOE properties for variables for:

| Property | Definition | Possible values |
| :--- | :--- | :--- |
| Falling-edge Severity | The indication of whether the service detects a fall from TRUE to FALSE. <br> When selected, the service detects a falling edge; when cleared, the <br> service ignores the transition. | Possible values are 0 to 1,$000 ;$ the default value is 1. |
| Rising-edge Severity | The indication of whether the service detects a rise from FALSE to TRUE. <br> When selected, the service detects a rising edge; when cleared, the <br> service ignores the transition. | Possible values are 0 to 1,000 ; the default value is 1. |
| Filter time | The minimum time lapse between two events. | Possible values range from 0 to $32,767 \mathrm{~ms} ;$ <br> value is 0 ms. default |
| Reference Variable | Variable of elementary data type | Possible values are BOOL, SINT, USINT, INT, UINT, DINT, <br> UDINT, and REAL. |

SOE is dependent on the scan time. When a wired input variable to SOE and the state changes, the variable uses the timestamp local to the I/O module. When an unwired variable is mapped to SOE and the state changes, the variable uses the timestamp voted by the processors at the beginning of the application cycle.

## To map SOE variables

1. In the Communication View, expand the required controller, and then expand the Workbench \& OPC element.
2. Double-click SOE.

The Communication property page displays the variables mapped for SOE communication.
3. In the Communication property page, add the required variables by clicking Add Variable.
4. For each variable, specify properties for:

- Falling-Edge Severity
- Rising-Edge Severity
- Filter Time (ms)
- Reference Variable

5. To remove a variable, select a variable from the list, and then click Remove Selected.

## See also

Specifying the Workbench and OPC Communications on page 191
Mapping OPC Variables on page 191

The AADvance controller provides up to six serial communication ports, two for each 9110 processor module present. Each pair of serial ports is identified as $S n-1$ and $S n-2$ where $n$ indicates the processor module:

- [S1-1 and S1-2]
- [S2-1 and S2-2]
- [S3-1 and S3-2]

The serial port settings define the protocol type and the data characteristics of the individual serial ports.

| Setting | Description |
| :--- | :--- |
| Name | Name of the serial port. |
| Baud Rate | The baud data transfer rate, in milliseconds. Possible values are $1200,2400,4800,9600$, <br> $19200,38400,57600,76800$ or $115200 ;$ The default value is 19200. |
| Data Bits | The number of bits representing one character of data in a transmission. Possible values <br> are $5,6,7$, and 8. The default value is 8. |
| Parity | The type of parity used. Possible values are None, Even, and Odd; the default value is <br> None. |
| Stop Bits | The number of stop bits used to indicate the end of a transmission. Possible values are 1 <br> or 2; The default value is 1. |
| Type | The protocol to use for the serial port. Possible protocols are the following: <br> - RS485fd - Full-duplex, 4-wire connection with separate buses for transmit and receive <br> - RS485fdmux - Full-duplex, 4-wire connection with separate buses for transmit and <br> receive and tri-state outputs on the transmit connections <br> - RS485hdmux - Half duplex, 2-wire connection <br> The default property is RS485hdmux. fd indicates full duplex and hd indicates half duplex. |
| Comment | Free-formatted text for the serial port. |

Tip: Most systems use two bits after each data byte. These bits are either a parity bit (odd or even) and one stop bit, or no parity and two stop bits.

## To configure serial ports

1. In the Communication View, expand the required controller, and then double-click the Serial element.

The Communication property page displays the serial ports.
2. In the Communication property page, specify the properties for the individual serial ports.

## See also

Specifying the Workbench and OPC Communications on page 191
Defining Ethernet Ports on page 177

## Configuring the Controller for SNTP

Use the SNTP server to automatically set the real-time clock (RTC), ensuring each controller uses the same date and time for time-related services, events, and $\log$ files.

Tip: It is strongly recommended to use SNTP to synchronize controllers.

The AADvance controller supports a fault-tolerant Simple Network Time Protocol (SNTP) service, synchronizing the date and time used by the networks of the system. The NTP protocol enables time to be communicated using networks. Synchronizing the date and time used by controllers facilitates using the Sequence of Events (SOE) service for Boolean variables, log-based troubleshooting, and increasing the dependability of other investigations.

Configure a controller in one of two modes for SNTP:

- SNTP client
- SNTP server

Controllers are usually configured as SNTP clients, receiving the time from a dedicated time server. When no dedicated time server is available, the AADvance controller can be used as an SNTP server. Using a local time server enables broadcasting. Broadcasting is considered the most efficient and predictable way to circulate the time to many clients.


Using an NTP time server from the Internet is not recommended for industrial
applications since the network transmission delays are unpredictable.
Use one of these methods to provide a network time server:

| Hardware time server | Retrieves the time of day from multiple sources such as GPS satellites, <br> radio-based international time standards (for example, DCF 77 in <br> Germany), and the internal clock of the hardware. The hardware time <br> server is the most adaptable option. |
| :--- | :--- |
| Software time server | A computer-based NTP server. The program enables controlling and <br> configuring the NTP services for Microsoft® Windows. Certain programs <br> also enable switching between configurations, changing service settings, <br> extracting related log entries, and displaying the current status of local <br> and external NTP services. |
| Processor module | The AADvance controller uses the real-time clock of one of the 9110 <br> processor modules. The default value for a new processor module is <br> January 1, 1900. <br> Configure any of the three processor modules as time servers. When one <br> processor module is configured as an SNTP server, the other processor <br> modules, including the module set as the server, must not be configured <br> as SNTP clients. <br> The processor modules of the controller automatically agree on the time. |

Whether configuring SNTP as a client, server, or both, always select both Ethernet ports of a processor slice (even when only a primary server is required). The first Ethernet port selected in the list is considered the primary network time server. The secondary time server is only used when no primary network time server is visible.

## See also

Configure a Controller as an SNTP Client on page 195
Configure a Controller as an SNTP Server on page 197
Using a Controller as an SNTP Client and Server on page 199

## Configure a Controller as an SNTP Client

As an SNTP client, the AADvance controller accepts the current time from an external Network Time Protocol (NTP) or SNTP network time server.

> Tip: $\quad$ Always select both Ethernet ports of a processor slice whether configuring
> SNTP as a client, server, or both. Also when only requires a primary server.

The SNTP client receives the real time clock data from the primary network time server.


The SNTP client settings provide this information to the controller:

| Setting | Description |
| :--- | :--- |
| Enable SNTP Client | The controller is configured as an SNTP client |
| SNTP Client Mode | Operating mode for the time synchronization signal used by the <br> processors for setting their real time clock: <br> - Unicast: The SNTP client actively polls the configured servers and <br> uses their responses. The polling rate (19s) is based on the drift rate of <br> the real time clock and cannot be configured. <br> - Broadcast: The SNTP client passively waits for regular broadcasts <br> from the server. This reduces network traffic and the load on the <br> servers. |
| Ethernet Port | Ethernet port of the controller |
| SNTP Server IP Address | IP address of the network time server. Non-fault tolerant operations <br> require one SNTP server for a processor. Other processors will <br> automatically synchronize to the server and inherit the time. Fault- <br> tolerant SNTP client operations require more than one server address. <br> If only a primary time server is required, set the secondary time server <br> IP address to the null value of 0.0.0.0. |
| Required SNTP Version | The version of the SNTP server. Possible versions are SNTPvl, SNTPv2, <br> SNTPv3, SNTPv4, or Any. When the version of the NTP/SNTP that the <br> server offers is unknown, choose Any. This latest option disables some <br> validation aspects of the incoming signal. |

## To configure a controller as an SNTP client

1. In the Communication View, expand the required controller, and then double-click SNTP.

The Communication property page displays.
2. From the Communication property page, in the SNTP Client section, select Enable SNTP Client.
3. Specify the SNTP Client Mode for each server:

- To use the broadcast mode where the SNTP client passively waits for regular broadcasts from the server, from the drop-down combo box, click Broadcast.
- To use the unicast mode where the SNTP client actively polls the configured servers and uses their responses, from the drop-down combo box, click Unicast.

4. Select the required Ethernet Ports and then define the SNTP Server IP Address property for the required servers.

- For a non-fault tolerant operation, define one SNTP server for only one processor. The other processors will automatically synchronize to the first processor and inherit the time.
- For a fault-tolerant SNTP client operation, define more than one server address.
Tip: The first address represents the primary server and the second address represents the secondary server for each processor module. At start up, the SNTP client chooses the primary server of the lowest slice; if no primary signal is valid, the SNTP client looks for an active secondary server signal.

5. Define the Required SNTP Version property for each required pair of servers for the processor modules.
Tip: When the NTP/SNTP server version is unknown, use the Any option, disabling some validation aspects of the incoming signal.

## See also

Configure a Controller as an SNTP Server on page 197
Using a Controller as an SNTP Client and Server on page 199
Configuring the Controller for SNTP on page 194

## Configure a Controller as an SNTP Server

The AADvance controller can fulfill the role of a primary or secondary SNTP server to provide a network time signal throughout the network.

Tip: Always select both Ethernet ports of a processor slice whether configuring
SNTP as a client, server, or both. Also when only requires a primary server.
SNTP servers operate in broadcast and unicast time synchronization signal modes. Specify a directed broadcast address for an interface enables serving time on the interface for both modes. This method of configuring is derived from the NTP configuration command language. The directed broadcast address for an interface:
= ( (IP address for interface) bitwise-and (subnet-mask) ) bitwise-or (bitwisenot subnet-mask)

Example: if the IP address for an interface is 10.10.1.240 and its subnet-mask is 255.255 .255 .0 , then the directed broadcast address is:
$=((10.10 .1 .240)$ bitwise-and (255.255.255.0)) bitwise-or (bitwise-not
255.255.255.0)
$=$ (10.10.1.0) bitwise-or (0.0.0.255)
$=(10.10 .1 .255)$

Important: Modifications to the SNTP settings are active after the power has cycled.


The SNTP server settings provide this information to the controller:

| Setting | Description |
| :--- | :--- |
| Enable SNTP Server | The controller is configured as an SNTP server |
| SNTP Server Mode | Operating mode for the time synchronization signal used by the <br> processors for setting their real-time clock. Configure the 9110 processor <br> module as one of these: <br> - Unicast, where the server waits to be polled by a client and then <br> responds with a time signal; it will not broadcast any time signals. <br> - Broadcast, where the controller regularly transmits a time signal: it will <br> also respond to unicast polling requests from clients. <br> Broadcasting is the most efficient mode when several devices rely on the <br> server for the time of day. |
| Ethernet Port | Ethernet port of the controller |
| SNTP Broadcast Address Mask |  |
| Broadcast Poll (seconds) | The broadcast polling time, in seconds. Possible values range from 4 to to <br> (0024; The default value is 10. |

## To configure a controller as an SNTP server

1. In the Communication View, expand the required controller, and then double-click SNTP.
The SNTP property page displays.
2. From the SNTP property page, in the SNTP Server section, select Enable SNTP Server.
3. Select the required Ethernet Ports, and then define the SNTP Server Broadcast Address Mask and Broadcast Poll (seconds) properties for the required servers.

- To disable the server, set the IP address to 0.0.0.0
- For a non-fault tolerant operation, define one SNTP server for only one processor. The other processors will automatically synchronize to the first processor and inherit the time.
- For a system requiring a fault-tolerant operation, define more than one server IP address.

4. Specify the SNTP Server Mode for each server:

- To use the unicast mode where the controller waits to be polled by a client and responds with a time signal without broadcasting any time signals, from the drop-down combo box, click Unicast.
- To use the broadcast mode where the controller regularly broadcasts and responds to unicast polling requests on the network, from the drop-down combo box, click Broadcast.


## See also

Configure a Controller as an SNTP Client on page 195
Using a Controller as an SNTP Client and Server on page 199
Configuring the Controller for SNTP on page 194

# Using a Controller as an SNTP Client and Server 

## Configuring Serial Tunneling

In a system with many AADvance controllers, configuring one controller as a client and server is helpful. The controller becomes the client of a hardware or software network time server and transmits the date and time to the other controllers configured as clients. A controller being both a client and server can reduce the load on the server when the clients are using unicast.

## See also

Configure a Controller as an SNTP Client on page 195
Configure a Controller as an SNTP Server on page 197
Configuring the Controller for SNTP on page 194
Serial tunneling enables running a controller in transparent communication (TCI) mode where the controller supports direct routing of traffic between Ethernet ports and serial ports, providing a terminal server interface to elements attached to the serial ports. Transparent communication mode is only available when the controller is not executing an application. While in grandparents communication mode, TCP/IP port address indicates the serial port to which is routed the Ethernet connection. Each serial port must have a unique TCP/IP port address. The TCP/IP port addresses for the serial ports are the following:

- 10001 for S1-1
- 10002 for S1-2
- 10003 for S2-1
- 10004 for S2-2
- 10005 for S3-1
- 10006 for S3-2

Each available serial port supports a single TCP/IP connection. A TCI TCP/IP connection cannot be established to an unavailable serial port. For example, where the corresponding processor module is not present or is inactive. Serial ports retain their configured settings for TCI mode.

Establish TCI links between any serial port and any Ethernet port (for example, the Ethernet and serial ports may be for different processor modules).

When a TCI link is established, any incoming data from the TCP/IP connection is transmitted to the serial port and any incoming data from the serial port is transmitted to the TCP/IP connection. Incoming data from the TCP/IP connection is immediately transmitted to the serial port. Incoming data from the serial port is transmitted to the TCP/IP connection when one of the following conditions occurs:

- The elapsed time between received characters exceeds the specified Idle Time value
- The total number of characters received reaches 256

Connections only transmit valid characters; Connections ignore characters with errors like parity or framing.

When configuring serial tunneling, individual serial ports are assigned to a TCP port. For each port, specify these properties:

- Inactivity Time - The period of time where no data flows through the TCP connection (in either direction) before the TCI function closes down the connection, in seconds. Closing the connection prevents a dormant connection being kept open. In any case, if the controller is re-enabled any TCI connections will be closed. Possible values are 1 to 65535 ; the default value is 600 .
- Idle Time - The period of time waiting for incoming characters from a serial device, in milliseconds. When a complete packet is received, the AADvance ${ }^{\oplus}$-Trusted ${ }^{\circledR}$ SIS Workstation software forwards the data through the Ethernet interface. If idle time is 0 , individual characters are forwarded in their own TCP segment, wasting much bandwidth and processing time. Possible values are 1 to 1000; the default value is 3 .

For online changes, while the system is operational, enable or disable transparent communications operations and modify the Inactivity Time and Idle Time properties.

## To configure serial tunneling

1. In the Communication View, expand the required controller, and then double-click Serial Tunneling.
The Serial Tunneling property page displays.
2. In the Serial Tunneling property page, select Enable Serial Tunneling.
3. For each port, define the Inactivity Time and Idle Time properties.

Bindings are links, for example access paths, between variables located in different controllers belonging to the same project. Define bindings between variables of the same type. From a controller, bindings are defined as having a producer variable sending values to a consumer variable.

The controller consuming the data establishes a binding link with the controller producing the data. The controller also schedules sending and receiving data, sending diagnostic data, managing the safety response when faults occur, and managing communications redundancy.

When defining bindings, controllers must be connected via the SNCP (Safety Network Control Protocol) network. The SNCP network is a SIL 3 certified protocol supplying a safety layer for the Ethernet network. This diagram displays an SNCP network:


The controllers requiring a binding link must be physically connected to each other. The SIL rating of the communications interface is unaffected by the design of the Ethernet network, but the network reliability and spurious trip rate are impacted. SNCP network data can be combined on a common network resulting in safety and non-safety data sharing a physical network. The SIL rating is unaffected by using a common network, but introduces failure modes and possible security risks increasing the spurious trip rate.

SNCP networks can be configured as simplex (fail-safe) or redundant (fault tolerant). The network configuration depends on the application safety and availability requirements. Since the connection between controllers is treated as a logical network, sending and receiving data occurs independently from the physical network.

For individual controllers, variable binding definitions are available from both the producer mapping and consumer mapping perspectives. Example: View the C1_Var1 (Producer) to C2_Var1 (Consumer) binding between Controller1
and Controller2 from the Produce Mapping view for Controller1 or the Consume Mapping section for Controller2.


Tip: When importing or adding existing controllers with bindings, these controllers must keep the same number and name as when exported to retain binding definitions. Before importing or adding such controllers, renumber and rename conflicting controllers.

Bindings use the parameters defined for the producer controller; parameters defined for the consumer controller are ignored. Multiple consumers can connect to a single producer. A single consumer can connect to multiple producers. A producer can also be a consumer and vice versa.

| Important: | For SNCP bindings, a producer variable supports one link to a <br> consumer variable. |
| :--- | :--- |

Variables are not updated in the consuming controller until the producing controller sends the values through the binding media:


The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software does not impose the readonly accessibility for consumed variables. To avoid conflicts between bindings and the execution of POUs, it is highly recommended to declare consumed variables with the read-only attribute.

Before defining bindings between controllers, create a link enabling the transfer of variable values between these controllers. For each controller, specify the Ethernet port allowing communication and the SNCP link properties.

## See also

Consume Status Variables for Bindings on page 203
Binding Process on page 203
Creating Links between Controllers on page 205
Mapping Variable Bindings on page 208

## Consume Status Variables for Bindings

To test values for the binding status, create these defined words in the dictionary for a project:

| Status Message | Value | Description |
| :---: | :---: | :---: |
| SNCP_KVB_ERR_BINDING_READY | 0 | Normal steady state data exchange. The binding link is healthy. |
| SNCP_KVB_ERR_BINDING_IN_PROCESS | 1 | Producer and consumer are in the process of connecting |
| SNCP_KVB_ERR_UPD_IN_PROCESS | 2 | Producer and consumer are in the process of reconnecting following an online update |
| SNCP_KVB_ERR_UPD_TIMEOUT | 3 | Producer and consumer did not reconnect within Update Timeout milliseconds following an online update |
| SNCP_KVB_ERR_NO_PRODUCER | 4 | At the beginning of a connection or following Bind Response Timeout milliseconds during data exchange |
| SNCP_KVB_ERR_BAD_CRC | 5 | Producer and consumer binding table CRC mismatch during connection |
| SNCP_KVB_ERR_IMPOSSIBLE_TO_BIND | 6 | Conversion of structures not supported on heterogeneous binding link |
| SNCP_KVB_ERR_IP_DENIED | 7 | Producer only error, indicating a consumer with an unknown or unexpected IP address is requesting binding links. The link is denied. |
| SNCP_KVB_ERR_BAD_PRODUCER_ID | 8 | Producer only error, indicating the producer has received a binding link request from a consumer containing an unexpected producer identity. The link is denied. |
| SNCP_KVB_ERR_BAD_GROUP_ID | 9 | Producer only error, indicating the producer has received a binding link request from a consumer but the request does not contain a known bound variable group identity. The link is denied. |
| SNCP_KVB_ERR_BAD_BUFF_SIZE | 10 | Producer only error, indicating that either the producer or consumer has insufficient buffer space to translate the bound variables. Unable to establish the link. |
| SNCP_ERR_BAD_CONFIG | 11 | Producer only error, indicating the producer is unable to obtain the IP address of the consumer from the network configuration file. The link is denied. |

To monitor the health of the bindings network, use the consume status variable and the following function blocks: KvbConsNetStatus and KvbProdNetStatus.

Define bindings between controllers in the Bindings section of the Communication View.

## See also

KvbConsNetStatus on page 482
KvbProdNetStatus on page 483
Creating Links between Controllers on page 205
Mapping Variable Bindings on page 208
At the start-up of a binding process, the consume status variable is set to the SNCP_KVB_ERR_NO_PRODUCER value and the consumer sends a Request Connect message to the producer. When the producer receives the Request Connect message, it validates the message, enters the consumer in a list, and sends a Reply Connect message if accepted. If no response is received within Connect Timeout milliseconds, the consumer sends another Request Connect message and this process continues until a Reply Connect message is received from the producer. During the connection process, the consume status variable value is set to the SNCP_KVB_ERR_BINDING_IN_PROCESS for the duration of the application cycle while sending the Request Connect message.

When the consumer receives the Reply Connect message, it also validates the message and may set the consume status variable value to SNCP_KVB_ERR_BAD_CRC if a mismatch exists between the producer and the
consumer CRC. If the connection is accepted, the consume status variable is set to SNCP_KVB_ERR_BINDING_READY.

Following this connection process, the consumer sends a Request Bindings message to the producer at each application cycle. If the consumer receives a Reply Binding message with the requested data in less than Max Age milliseconds, the bound variables are updated. The consumer may sets its status variable value to SNCP_KVB_ERR_IMPOSSIBLE_TO_BIND if the data contains a heterogeneous link impossible to convert. If no response is received within Bind Response Timeout milliseconds, the consume status variable value is set to SNCP_KVB_ERR_NO_PRODUCER, then the consumer sends a Disconnect message to the producer and restarts the binding process. If the producer does not receive a Request Bindings message within Bind Request Timeout milliseconds, the consumer is removed from the producer's list.

During an online change update, the following possibilities can occur:

- The target is a producer or a consumer for the first time, the connection process is started and proceeds as during start-up.
- The target was and remains a producer, the producer clears the list of connected consumers, loads new binding data, restarts the binding process and waits for a consumer to request a connection.
- The target was and remains a consumer, the consumer sends a Disconnect message to the producer, loads new binding data and enters in a reconnection state. In the reconnection state, the consume status variable is set to the SNCP_KVB_ERR_UPD_IN_PROCESS value and the consumer sends a Request Connect message to the producer. If an update behavior default value is defined, the bound variable takes this value. The reconnection process following an online change is the same as the start-up connection process except CRC errors are ignored since the producer may not yet be updated. A reconnection attempt will also occur once every Connect Timeout time lapse.
- If the re-connection is successfully completed within Update Timeout milliseconds, the consume status variable is set to the SNCP_KVB_ERR_BINDING_READY value and begins sending Request Bindings messages. The online change update is over and successful.
- If the reconnection is not successfully completed within Update Timeout milliseconds, the consume status variable is set to the SNCP_KVB_ERR_UPD_TIMEOUT value and sends a Disconnect message to the producer, then restarts the process as during start-up. The online change update is over and has failed.


## See also

Consume Status Variables for Bindings on page 203
Creating Links between Controllers on page 205
Mapping Variable Bindings on page 208

## Creating Links between Controllers

Links between controllers enable the transfer of values between producer and consumer variables defined in bindings. Before defining variable bindings, create links between controllers with producer variables and consumer variables to use in these bindings.

To use bindings, define at least one Ethernet port to be used by SNCP bindings. When multiple ports are defined, each IP address is pinged to locate accessible links. Communication occurs on the first Ethernet port (slice) selected in the list of ports; the other addresses continue to be pinged. If a timeout occurs on the active link, communications switch to another IP address. To use bindings, configure the SNCP link properties.

Tip: To correct the port configuration when reopening a project after changing the port selection:

1. Export the variables used in bindings between both controllers.
2. Delete the incorrect binding link and save changes.
3. Recreate the binding link and save changes.
4. Import the exported variables.


Binding contains these SNCP link properties:

| Property | Description |
| :--- | :--- |
| Consume Status Variable | The wired variable used to report the health status of the link. The data <br> type of the variable must be DINT. A healthy binding link returns a value <br> of 0, while a link containing a fault returns a non-zero value. <br> Tip: <br> To obtain the correct status, ensure the first mapped <br> variable in the Bindings list does not use the LINT, ULINT, <br> LWORD, or LREAL data type. |


| Bind Request Timeout | The time period the producer waits to receive a binding data request from the consumer, in milliseconds. In the event of a timeout, the consumer is disconnected. Subsequent requests from the consumer are ignored until the consumer establishes a new connection. <br> Tip: The network binding parameters, for example timeout values, of the consumer are found in the producing controller. <br> Possible values range from 0 to 65535 milliseconds. The default value is 10000 milliseconds. |
| :---: | :---: |
| Bind Response Timeout | The time period during which the consumer awaits binding values from the producer, in milliseconds. In the event of a timeout, the consumer can send another request (dependent on the value of Max Age) or disconnects from the producer. <br> The number of requests sent is calculated as: <br> Max Age/Bind Response Timeout <br> Once disconnected, the consumer attempts to reestablish the connection by sending a Connect request at Connect Timeout intervals. <br> Possible values range from 0 to 65535 milliseconds. The default value is 1200 milliseconds. |
| Connect Timeout | The time period during which a consumer awaits a connection response from a producer, in milliseconds. In the event of a timeout, the consumer sends another Connect request. The consumer continues to send requests until a connection is established. Possible values range from 0 to 65535 milliseconds. The default value is 10000 milliseconds. |
| Max Age | The total time, in milliseconds, during which the consumer awaits a valid response from the producer before considering the physical network unhealthy. Stale, corrupt, and out of sequence responses are considered invalid. Continuing to receive these types of responses can indicate underlying network issues, demonstrated by the unhealthy network status. The unhealthy status is reported by the KvbConsNetStatus and KvbProdNetStatus function blocks. <br> The number of requests is calculated as: <br> Max Age/Bind Response Timeout <br> The Max Age is also used to verify the age of the binding response received by the consumer. If a response contains data older than Max Age, the response is discarded. Discrepancies may occur when a response is delayed due to network issues. <br> Possible values range from 0 to 65535 milliseconds. The default value is 2500 milliseconds. |
| Update Timeout | The timeout period during which a consumer and producer must reconnect after an update has modified the binding information, in milliseconds. <br> Important: Perform updates as quickly as possible to avoid timeouts. |
|  | During the Update Timeout period, the configured consume status variable continues to indicate a healthy status even though no connection exists between the consumer and producer. In the event of a timeout, the consume status variable becomes unhealthy. Possible values range from 0 to 65535 milliseconds. The default value is 60000 milliseconds. |
| Comment | Free-format text |

Important: Linked controllers must have the same SNCP Link Properties.

## Timeout Example

This example displays the timeout behavior when Max Age is set to 2500 ms and Bind Response Timeout is set to 1000 ms :


## To create a link between controllers

1. In the Communication View, expand the required controller to link, then right-click Bindings, point to Add Controller Link, and then click the controller with which to link.
2. Double-click the Bindings link.


The Communication property page is displayed with the Allow Communication and SNCP Link Properties sections.
3. From the Communication property page, in the Allow Communication section, select the required Ethernet ports. The available Ethernet ports are those defined for the individual controllers.
4. In the SNCP Link Properties section, enter the required values for the following properties:

- Bind Request Timeout (ms)
- Bind Response Timeout (ms)
- Connect Timeout (ms)
- Max Age (ms)
- Update Timeout (ms)

5. If required, click $\quad$ to wire a Consume Status Variable.

The Variable Selector is displayed.
6. In the Variable Selector, select the required variable of type DINT, and then click OK.
7. (optional) To delete a link between controllers, do the following:
a. In the Communication View, expand the required controller, and then expand Bindings.
b. Right-click the controller link to remove, and then click Delete.

## See also

Consume Status Variables for Bindings on page 203
Binding Process on page 203
Mapping Variable Bindings on page 208

## Mapping Variable Bindings

Map variable bindings from the producer or consumer mapping perspective for a controller. When mapping bindings, view all available variables for both linked controllers. Create a binding by selecting a variable from each controller, then adding it to the bindings list. The Bindings list displays the variables and data type of defined bindings. For the consumption behavior of individual variable bindings, specify a value to use for error behavior and update behavior; the default value is the last value for the variable binding.

[^4]Variable binding from controller produce mapping perspective:


Variable binding from controller consume mapping perspective:


When selecting a variable from a list of available variables, details for the variable display below the list. Limit the number of variables that display for a controller by typing search characters.
Delete or rename variables used as producer variables in bindings but these modifications cause the bound variables to display errors.

The consumption behavior indicates the values to use when communication for the binding is in error, for example, the update timeout period is exceeded, and during an online update. The update behavior period begins when the producing or consuming variables are being updated and ends when the binding definition between the producing and consuming variables match and the consuming variable receives valid production values.

## To map a variable binding

1. In the Communication View, select the controller having the producer variable to include in the binding.
2. Create a link with the controller having the consumer variable to include in the binding.
The link between the controllers is created and has a Produce Mapping and Consume Mapping item.

3. Double-click Produce Mapping.

The Communication property page displays and lists the producer controller's variables in the list on the left and the linked consumer controller's variables in the list on the right.
4. From the Communication property page, select the producer variable from the list of producer variables on the left and the consumer variable from the list of consumer variables on the right.
5. Verify the variable information displayed below the lists of producer variables and consumer variables.
6. Specify an Error Behavior and Update Behavior for the binding, and then click Add.
The variable binding displays in the Bindings list.

## See also

Creating Links between Controllers on page 205
Binding Process on page 203
Consume Status Variables for Bindings on page 203

## Use CIP over EtherNet/IP

The Common Industrial Protocol CIP ${ }^{\text {TM }}$ over EtherNet/IP ${ }^{\text {TM }}$ protocol enables AADvance controllers to exchange data with controllers supporting the CIP protocol, such as ControlLogix ${ }^{\circledR}$ controllers. The CIP network exchanges data using producers and consumers; this network cannot download to a target or monitor a target. The CIP data sharing mechanism is similar to the variable bindings mechanism used in AADvance. For each controller, CIP communication enables defining a produce link and multiple consume links.

The controller supports produce and consume communications to redundancy systems. The support for produce/consume variables is non-interfering; a failure of the EtherNet/IP stack does not interfere with the safe operation of the system.

To use CIP over EtherNet/IP, define the required consume and produce links, specify an Ethernet port to connect to the CIP network having the remote controller, then define producer and consumer variables (or structures) for the exchange of data. For CIP consume links, also specify the producer and the CIP path to the producer. At runtime, the controller having producer variables sends values to the remote controller, whereas, the controller having consumer variables pulls values from the remote controller having producer variables.

> | Important: | When connecting a controller not supporting CIP to a CIP network, a |
| :--- | :--- |
|  | warning occurs and the connection is not implemented. The default |
|  | IP address for a connection between a controller and a CIP network |
| is blank. When a valid IP address is not specified, an error when |  |
|  | building occurs: 'Network CIP: invalid connection properties for (the |
|  | particular configuration).' |

## Communications Cycle

Before executing the logic at the start of an application scan, the AADvance controller sets the consumer variables to the most recently received values. The controller updates the producer variables after executing the logic at the end of the application scan. When sending a packet, the AADvance controller uses the most recent value of a producer variable, ensuring the variable uses the expected Requested Packet Interval (RPI).

A consumer variable retains the most recently received value; a default value cannot be set.

To use CIP over EtherNet/IP, configure controllers to use the Unicast mode since AADvance controllers use unicast for CIP.

## To configure an AADvance controller to communicate with another controller

1. Set the first AADvance controller as a CIP provider.
2. Set the second controller as a CIP consumer.

Tip: After building the project, if the Output window displays a message indicating Multiple CIP network instances found, see Update Ethernet ports of CIP produce and CIP consume links.

## See also

Creating CIP Consume Links on page 213
Update Ethernet ports of CIP produce and CIP consume links on page $\underline{216}$

## Creating the CIP Produce Link

For a controller, create one CIP $^{\text {TM }}$ produce link for sending data to a remote controller across a CIP network. When defining a produce link, specify the Ethernet port connecting to the CIP network with the remote controller. Then map producer variables.

| Important: | If the IP address of the producer is modified, update the path to the <br> producer in the consume link. |
| :--- | :--- |

## To create the CIP produce link

1. In the Communication View, expand the required controller, right-click CIP, and then click Add CIP Produce Link.

The With Any link and CIP Produce item are added to the Communication View tree structure.
2. Double-click the With Any link, then in the Communication property page, select the Ethernet port to use for communication with the CIP network.

Tip: Your specified Ethernet ports also apply to all CIP Consume Links of this controller.

## See also

Map CIP Producer Variables on page 212
Creating CIP Consume Links on page 213

## Map CIP Producer Variables

Each AADvance controller supports 128 mapped producer variables and a maximum of 255 CIP $^{\text {TM }}$ variables. The maximum size of a CIP variable is 500 bytes. A variable is configured either as a producer or as a consumer; a variable cannot be both. Only global variables are available for use with controllers supporting a connection to a CIP Network. Use only BOOL, DINT, INT, LINT, LREAL, REAL, and SINT data types with CIP; library data types are unsupported.

The Communication property page displays these properties for CIP producer variables:

| Property | Description |
| :--- | :--- |
| Name | Name of the variable. For CIP producers, variable names have a maximum of 40 <br> characters. |
| Remote Tag Name | The name of the CIP producer, which is a specific identifier for CIP, is exposed <br> on CIP protocol. To allow data exchange, the Remote Tag Name of the CIP <br> consumer must match the Remote Tag Name of the producer. For projects <br> imported from AADvance Workbench 1.1, 1.2, 1.3, and 1.4, the displayed Remote <br> Tag Name and Name for CIP Producer are the same. |


| Property | Description |
| :--- | :--- |
| Data Type | Data type of the global variable. <br> Supports BOOL, DINT, INT, LINT, LREAL, REAL, and SINT data types when the data <br> type is 4 bytes ( 32 bits) or larger. Also supports array or structure variables but <br> not member elements of arrays or structures. To transfer data types smaller <br> than four bytes, create a user-defined data type and pad it accordingly. <br> When the variable is a structure that contains members of supported data <br> types, each member starts on a new byte or word depending on the size. For <br> example, a DINT starts a new 4-byte word even though the last word only had a <br> one-bit BOOL. |
| Firection | For l/0 wiring, indicates if a variable is an input, output, or internal. |
| Max Connections maximum number of simultaneous consumers of the variable. Possible |  |
| values range from 1 to 10. The default value is 5. |  |
| The AADvance controller supports a maximum of 255 CIP connections. |  |
| For AADvance® Workbench prior to version 1.40, CIP producer variables support |  |
| one Max Connection to a consumer. Configuring the number of consumers |  |
| allowed by a produced value to any value higher than 1 treats all higher values |  |
| as 1. |  |

## To map CIP producer variables

1. In the Communication View, expand the required controller, right-click CIP, and then click Add CIP Produce Link.
The With Any link and CIP Produce item are added to the Communication View tree structure.
2. Double-click the With Any link, then in the Communication property page, select the Ethernet port to use for communication with the CIP network.

## See also

Creating the CIP Produce Link on page 212
Creating CIP Consume Links on page 213
Map CIP Consumer Variables on page 215
Creating CIP Consume Links
For a controller, define up to 128 CIP ${ }^{\text {mM }}$ consume links for receiving data from remote controllers across a CIP network. Add one consume link per controller requiring CIP communications. When defining a CIP consume link, specify the Ethernet port connecting to the CIP network with the remote controller.

[^5]

Then map consumer variables.

## To create CIP consume links

1. In the Communication View, expand the required controller, right-click CIP, and then click Add CIP Consume Link.
The With Unnamedx (Not configured) link and CIP Consume item are added to the Communication View tree structure.
2. Configure the CIP consume link:
a. Double-click the With Unnamedx (Not configured) link to display the Communication property page.
b. From the property page, select the Ethernet port to use for communication with the CIP network.

Tip: Your specified Ethernet ports also apply to the CIP Produce Link and to all other CIP Consume Links of this controller.
c. In the Producer Name field, indicate the name of the CIP producer from which the consumer receives data.
d. In the CIP Path to Producer field, type the CIP path to the controller. The path uses the following format:

IP_address,port,slot
where
IP_address is the address of the CIP Ethernet module in the system. port is the communications route from the remote controller. For ControlLogix ${ }^{\circledR}$ controllers, the port should always be 1 specifying the ControlLogix backplane. Other numbers specify different communications routes from ControlLogix.
slot is the number of the slot where the CPU module is installed. These slots start from 0 and are numbered from left to right.

## See also

Map CIP Consumer Variables on page 215
Creating the CIP Produce Link on page 212
Map CIP Producer Variables on page 212

## Map CIP Consumer

Variables

Each AADvance controller supports 128 mapped consumer variables and a maximum of $255 \mathrm{CIP}^{T M}$ variables. The maximum size of a CIP variable is 500 bytes. A variable is configured either as a producer or as a consumer; a variable cannot be both. Only global variables are available for use with controllers supporting a connection to a CIP Network. Only BOOL, DINT, INT, LINT, LREAL, REAL, and SINT data types can be used with CIP; library data types are unsupported.

Tip: When adding a consumer variable, define the Remote Tag Name and Requested Packet Interval (RPI) properties.

The Communication property page displays these properties for CIP consumer variables:

| Property | Description |
| :--- | :--- |
| Name | Name of the variable. For CIP consumers, variable names are limited to 40 <br> characters. |
| Remote Tag Name | The name of the remote CIP producer, which is a specific identifier for CIP, is <br> exposed on the CIP protocol via the Remote Tag Name of the producer. <br> To allow data exchange, the Remote Tag Name of the CIP consumer must match <br> the Remote Tag Name of the producer. |
| RPI | Requested Packet Interval. The frequency in milliseconds that the remote <br> controller offers the variable to the AADvance controller. Possible values range <br> from 1 to 1000 ms. The default value is 20 ms. <br> The RPI should be set to a minimum of 200 ms. |
| Data Type | Data type of the global variable. <br> BoOL, DINT, INT, LINT, LREAL, REAL, and SINT data types are supported as long as <br> the data type is of size 4 bytes (32 bits) or larger. Array or structure variables are <br> also supported, but not member elements of arrays or structures. To transfer <br> data types smaller than four bytes, create a user-defined data type and pad it <br> accordingly. <br> When the variable is a structure containing members of supported data types, <br> each member starts on a new byte or word depending on the size. For example, a <br> DINT starts a new 4-byte word even though the last word only had a one-bit BoOL. |

## To map CIP consumer variables

1. In the Communication View, expand the required $\mathrm{CIP}^{\text {™ }}$ consume link, and then double-click CIP Consume.

The Communication property page displays.
2. In the property page, map a consumer variable by clicking Add Variable to access the Variable Selector.
3. In the Variable Selector, select the required variable, and then click OK.
4. In the property page, define the Remote Tag Name and RPI for the consumer variable.
The consumer variable is added to the CIP Consume list of variables.

## See also

Creating the CIP Produce Link on page 212
Creating CIP Consume Links on page 213
Map CIP Producer Variables on page 212

# Update Ethernet ports of CIP produce and CIP consume links 

After building the project, if the Output window displays a message indicating Multiple CIP network instances found, update the Ethernet ports of CIP produce and CIP consume links.

## To update Ethernet ports of CIP produce and CIP consume links

1. Clear all Ethernet ports of each CIP produce and CIP consume link.
a. In Communication View, double-click the CIP produce or CIP consume link.

The Communication property page displays.
b. Clear all Ethernet ports in the Communication property page.

Tip: Perform Step 1 on all CIP produce and CIP consume links.
2. Save and then close the project.
3. Open the project.
4. Select the required Ethernet ports for each CIP produce and CIP consume link.
a. In Communication View, double-click the CIP consume or CIP produce link.

The Communication property page displays.
b. Select the required Ethernet ports in the Communication property page.

Tip: Perform Step 4 on all CIP produce and CIP consume links.
5. Build the project.

## See also

Using CIP over EtherNet/IP on page 210
Creating the CIP Produce Link on page 212
Creating CIP Consume Links on page 213

## Resolve invalid CIP links

After opening an AADvance project, if the Output window displays invalid CIP produce link or invalid CIP consume link, remove the CIP variable, and then return the deleted CIP variable to the project.

## To resolve invalid CIP links

1. In Communication View, double-click the CIP produce or CIP consume link.
2. Select the CIP producer variable or the CIP consumer variable with the invalid CIP link.
3. Select Remove Selected.
4. Return the deleted CIP variable.
a. Select Add Variable.
b. Select the deleted variable.
c. Select OK.
5. Save the project.


Tip: When you open or build the AADvance project, the Output window no longer displays the invalid CIP links.

## Configuring Peer-to-Peer Communications

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software provides the option of setting up a Peer-to-Peer network to enable communicating application data between AADvance and Trusted ${ }^{\circledR}$ systems. Each peer network supports communication with up to 40 AADvance or Trusted systems. Data can be transferred between individual systems or from one system to multiple systems by using multicast network communication.

Tip: The current version of the AADvance controller only supports two multicast IP addresses. A fixed address can be configured for each Ethernet port.

A Peer-to-Peer network consists of one or more Ethernet networks connecting together a series controllers to enable passing safety application data between them. Peer controllers only communicate with other peer controllers sharing the same network. Connect these types of controllers:

| Trusted controller | The controller has four communication interfaces with eight Ethernet ports. <br> The Peer-to-Peer network can use all eight physical Ethernet networks, <br> referred to as subnets, to provide redundant data paths via up to eight <br> separate physical routes. The eight ports can be assigned to one or multiple <br> peer networks. <br> To participate in peer communication, fit each Trusted system with a T812x <br> processor interface adapter. |
| :--- | :--- |
| AADvance controller | The two network interfaces of the AADvance processor module can be used <br> simultaneously for Peer-to-Peer communication. A triple modular redundant <br> (TMR) controller with three processor modules provides a maximum of six <br> physical connections that can be divided between the different peer networks <br> or assigned to the same network. |

A peer network consists of one or more subnets. Network subnets are assigned to the Ethernet ports of the processor. Each subnet is considered a physical network representing a redundant data path on the network. Subnets of the same redundant network should use different processor modules to achieve the highest level of hardware fault tolerance.

| Important: | To use Peer-to-Peer communications on both ports of a processor, |
| :--- | :--- |
|  | configure IP addresses with different subnets. |

When configuring the AADvance controller to communicate with a Trusted Peer-to-Peer network, use the same parameters as a Trusted configuration.

Tip: The AADvance-Trusted SIS Workstation software also supports external communications using Modbus ${ }^{\circledR}$ over serial and Ethernet links. Performance may be affected when using both Modbus and Peer-to-Peer communications.

The information to transfer over the peer network is defined within the application data blocks. Configure data blocks of 16 or 128 digital points, 16 or 128 analog points, and the network status information.

To configure Peer-to-Peer communications, perform these tasks:

- Create a Peer-to-Peer network
- Configure the Subnet
- Enter the peer IP address information
- Define a Peer List
- Configure the data blocks
- Wire the status variables
- Wire the data variables


## See also

Configuring a Peer-to-Peer Network on page 219
Configuring a Peer-to-Peer Subnet on page 220
Defining the Peer List on page 222
Configuring Peer-to-Peer Data Blocks on page 223

## Peer-to-Peer Communication Cycle

Peer-to-Peer communication functions as a master slave relationship, needing at least one master per network. Configure redundant masters to ensure subnets remain connected if the master peer goes offline. The masters arbitrate the control. Configure any Ethernet port as a master or slave.

The Peer-to-Peer network operates in a cycle. Each controller is asked in turn to send all the configured output data.

At the start of the communications cycle, the peer master issues an inquiry command to the first slave. If the master receives a response, the slave is
registered as active and the master repeats the process with the next slave. The sequence continues until all slaved are polled.


The master then sends a transmit data command (token) instructing the first slave to send the output data to the configured peers. The first slave then returns the token to the master and the master repeats the process with the next slave. Once all slaves have been polled, the master transmits the output data. The transmit data cycle is then repeated with the first slave.


The master continuously repeats the communications cycle.
For redundant networks, data integrity is verified using the CRC of the data packet sent between systems.

## See also

Configuring Peer-to-Peer Communications on page 217

## Configuring a Peer-to-Peer Network

When configuring the Peer-to-Peer network, define these settings:

| Network ID | The number identifying the network for the controller. Possible values range from 1 to 8. |
| :--- | :--- |
| Peer ID | The number identifying the controller IP address in the Peer List. Possible values range <br> from 1 to 40. |

## To configure a peer-to-peer network

1. In the Communication View, expand the required controller, then right-click Peer-to-Peer, and then click Add Network.

The Network Configuration dialog box displays.
2. In the Network Configuration dialog box, set the Network ID and Peer ID, and then click OK.
3. (optional) To delete a peer-to-peer subnet, in the Communication View, right-click the network instance to remove, and then click Delete.

The Network with subnets and data blocks are deleted.

## See also

Defining the Peer List on page 222
Configuring Peer-to-Peer Data Blocks on page 223
Configuring Peer-to-Peer Communications on page 217

## Configuring a Peer-to-Peer Subnet

When configuring the Peer-to-Peer subnet, define these settings:

| Setting | Description |
| :--- | :--- |
| Subnet ID | The number identifying the Ethernet network for the controller. Possible values <br> are 1 to 8. |
| Network Interface | The processor slice that the peer network is physically connected to. Possible <br> values are Processor Slice A, Processor Slice B, Processor Slice C, or No <br> Connection. |
| Response Timeout (ms) | The amount of time for a peer to acknowledge reception of a data packet, in <br> milliseconds. Possible values range from 0 to 10000 ms. The default value is 0 <br> ms. <br> Set the value above 0 to avoid network packet sequence errors, which occur <br> when the propagation delay between two nodes exceeds 1 ms. The value must be <br> greater than the propagation delay (measurable by using the ping command). <br> Set to O when no acknowledgment is required. For example, no acknowledgment <br> is required if the entire Peer-to-Peer network is on a local Ethernet network and <br> contains no routers or gateways. <br> The Response Timeout acts as a watchdog trip for a lost packet and should be <br> the shortest value for the timeout settings. |
| Refresh Timeout (ms) | Note: To provide some tolerance to message loss, a slave is only declared <br> inactive (FALSE) after the occurrence of two consecutive response timeouts. |
| The maximum amount of time, in milliseconds, the peer controller waits during a <br> cycle for the master to signal to the peer to transmit the output data. Possible <br> values range from 0 to 10000 ms. The default value is 2000 ms. <br> The Refresh Timeout value is calculated by multiplying the Slave Transmit <br> Timeout value with the number of peers lost on a worst case network break. <br> When the Refresh Timeout expires, the peer discards any data to transmit and <br> considers the network inoperable. The network must be reactivated by the <br> master. <br> If the network is broken, the Slave Transmit Timeout trips, minimizing the loss of <br> data transfer. |  |


| Slave Transmit Timeout (ms) | The amount of time, in milliseconds, that a subnet master controller waits for a slave to complete transmission of the output data and return control of the network before declaring the slave absent. The Slave Transmit Timeout is only used by the master of a subnet. Possible values range from 1 to 10000 ms (a minimum of 64 ms is recommended). The default value is 500 ms . <br> The Slave Transmit Timeout acts as a watchdog for a lost slave, allowing enough time for the slave to send all the outputs even if the slave is waiting for lost packets from the Response Timeout setting. <br> A slave is only declared inactive (FALSE) after the occurrence of two consecutive timeouts. The slave remains inoperable until reactivated by the subnet master. To calculate the Slave Transmit Timeout value, do the following: <br> - (Response Timeout of the slave) x (Maximum number of output data boards in any controller) + 16 ms <br> - When the Response Timeout value of the slave is 0 ms , use 2 ms in the previous calculation. |
| :---: | :---: |
| Enable Controller | The wired variable starts or stops the peer communications for the controller. Possible values are TRUE or FALSE. <br> - TRUE - Enable Peer-to-Peer communications <br> - FALSE - Disable Peer-to-Peer communications |
| Master/Slave | Sets if the controller is a master or slave. Possible values are TRUE or FALSE. <br> - TRUE - The controller acts as a master <br> - FALSE - The controller acts as a slave <br> Set at least one master to enable Peer-to-Peer communication. |
| $\begin{array}{ll}\text { Important: } & \begin{array}{l}\text { To use Peer-to-Peer communications on both ports of a processor, configure IP } \\ \text { addresses with different subnets. }\end{array}\end{array}$ |  |

## To configure a peer-to-peer subnet

1. In the Communication View, right-click the required Peer-to-Peer network instance, and then click Add Subnet.

The Subnet Configuration dialog box displays.
2. In the Subnet Configuration dialog box, set the Subnet ID and then click OK.
3. In the Communication View, double-click the subnet instance.

The Communication property page displays.
4. In the property page, enter the required values for the following properties:

- Network Interface
- Response Timeout
- Refresh Timeout
- Slave Transmit Timeout

5. Click to wire variables for the following parameters:

- Enable Controller
- Master/Slave

6. (optional) To delete a peer-to-peer subnet, in the Communication View, right-click the subnet instance to remove, and then click Delete.
Tip: $\quad$ Networks must always have at least one subnet; the last remaining subnet under a network cannot be removed.

## See also

## Configuring a Peer-to-Peer Network on page 219

Defining the Peer List on page 222
Configuring Peer-to-Peer Data Blocks on page 223
Configuring Peer-to-Peer Communications on page 217
The Peer List contains the IP addresses of the peer controllers defined for one subnet. Each peer controller has a unique Peer ID, defining the position of the IP address in the Peer List. The Peer List can contain up to 40 Peer IDs. The Peer List must be the same for all controllers on the same subnet. Configure multicast IP addresses in the Peer List.


Unique IDs enable data to pass between one output data block and one input data block, or multicast from one output data block to multiple input data blocks on different controllers. Multicasting is part of TCP/IP communications. IP addresses that are used specifically for multicast are:

- 224.1.2.3
- 224.4.5.6

Each receiving peer is configured to accept data from the multicast addresses and the defined IP address for that peer.

From the Peer List, wire a Boolean input variable to get the peer status. The status variable is TRUE when the peer is active and FALSE when the peer is inactive or faulty. Multicast peers are always set to FALSE (even when active).

## To define the peer list

1. In the Communication View, double-click the required Peer-to-Peer subnet instance.

The Communication property page display with the Subnet and Peer List tabs.
2. In the property page, click the Peer List tab.
3. In the list, enter the IP address for each peer. Enter the IP address in the row with the corresponding Peer ID.
4. To configure a multicast IP address, enter the following information:

- Port 1: 224.1.2.3
- Port 2: 224.4.5.6

5. Click ... to wire status variables for the required peers.
6. (optional) To unwire a status variable, click $x$.

## See also

Configuring a Peer-to-Peer Network on page 219
Configuring a Peer-to-Peer Subnet on page 220
Configuring Peer-to-Peer Data Blocks on page 223
Configuring Peer-to-Peer Communications on page 217

## Configuring Peer-to-Peer Data Blocks

The information to transfer over the peer network is defined in data blocks. Use the data blocks for safety and non-safety data. Data blocks can have either 16 or 128 channels. Add these types of data blocks:

- Digital Inputs
- Analog Inputs
- Digital Outputs
- Analog Outputs

Each data block must have a unique ID on the peer network. To enable communication between an input data block and output data block, the blocks must be on the same network, have the same Block ID, and reference each other's Peer ID in the Peer List.

After adding a data block, wire status variables and data variables.
When adding a data block, define these properties:

| Block ID | The number identifying the data block. Possible values range from 1 to 64. |
| :--- | :--- |
| Direction | Indicates whether a data block is an input or output. |
| Peer ID | For input data blocks, the number identifying the peer sending data. For output data <br> blocks, the number identifying the peer receiving data. Possible values range from 1 to <br> 40. |
| Type | The type of data block. Possible values are digital and analog. |
| Size | The amount of channels associated to a data block. Possible values are 16 and 128. | | The analog data block with 128 channels must only be used for non-safety related |
| :--- |
| data since there are insufficient measures to ensure the transport of data meets |
| the safety integrity level of SIL3. |

## To add a data block

1. In the Communication View, right-click the required peer-to-peer network, and then click Add Data Block.

The Data Block Configuration dialog box displays.
2. In the Data Block Configuration dialog box, enter the required values, and then click OK.

## See also

Peer-to-Peer Digital Inputs on page 224
Peer-to-Peer Analog Inputs on page 226
Peer-to-Peer Digital Outputs on page 227
Peer-to-Peer Analog Outputs on page 228
Data Variables on page 229

## Peer-to-Peer Digital Inputs

When configuring a digital input data block, define these properties:

| Property | Description |
| :--- | :--- |
| Refresh Timeout | The maximum number of milliseconds allowed between successive <br> refreshes of the input data. When the time is exceeded, the input data is <br> declared invalid and either retains the last received value or reverts to the <br> fail-safe condition, depending on the Hold Last Valid Value setting. <br> Possible values range from 1 to 10000 ms. The default value is 5000 ms. |
| Value in Failed State | The control value to be adopted by the input when the input data is not <br> refreshed within the Refresh Timeout value and therefore declared <br> invalid. Possible values are 0 and 1. The default value is 0. |

Wire the following status variables for digital input data blocks:

| Variable | Description |
| :--- | :--- |
| Input Data is Valid | - TRUE - Input data was refreshed within the specified Refresh Timeout <br> value. <br> - FALSE - Input data was not refreshed within the specified time. |
| Refreshed by Subnet 1 | This status variable is used to detect latent faults within a redundant <br> network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 17 within the specified <br> Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 1 within the specified <br> Refresh Timeout milliseconds. |
| Refreshed by Subnet 2 | This status variable is used to detect latent faults within a redundant <br> network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 2 within the specified <br> Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 2 within the specified <br> Refresh Timeout milliseconds. |
| Refreshed by Subnet 3This status variable is used to detect latent faults within a redundant <br> network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 3 within the specified <br> Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 3 within the specified <br> Refresh Timeout milliseconds. |  |


| Variable | Description |
| :---: | :---: |
| Refreshed by Subnet 4 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 4 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 4 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 5 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 5 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 5 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 6 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 6 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 6 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 7 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 7 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 7 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 8 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 8 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 8 within the specified Refresh Timeout milliseconds. |
| Hold Last Valid Value | - When set to FALSE, forces data to the fail-safe state when the input data is invalid. <br> - When set to TRUE, allows previous data to persist when input data is invalid. |

## To configure a digital input data block

1. In the Communication View, double-click the digital input data block.

The Communication property page displays with the Data Block and Data Variables tabs.
2. In the Data Block tab, enter the required values for the Refresh Timeout and Value in Failed State.
3. Click ... to wire the required status variables.
4. (optional) To unwire a status variable, click $x$.

## See also

Peer-to-Peer Analog Inputs on page 226
Peer-to-Peer Digital Outputs on page 227
Peer-to-Peer Analog Outputs on page 228

Configuring Peer-to-Peer Data Blocks on page 223

## Peer-to-Peer Analog Inputs

When configuring an analog input data block, define these properties:

| Property | Description |
| :--- | :--- |
| Refresh Timeout | The maximum number of milliseconds allowed between successive <br> refreshes of the input data. When the time is exceeded, the input data is <br> declared invalid and either retains the last received value or reverts to the <br> fail-safe condition, depending on the Hold Last Valid Value setting. <br> Possible values range from 1 to 10000 ms. The default value is 5000 ms. |
| Value in Failed State | The control value to be adopted by the input when the input data is not <br> refreshed within the Refresh Timeout value and therefore declared <br> invalid. Possible values range from -3.402823E +38 to $3.402823 \mathrm{E}+38$. The <br> default value is -1024. When the data type of the input variable is DINT, <br> the decimals are truncated. |

Wire the following status variables for analog input data blocks:

| Variable | Description |
| :--- | :--- |
| Input Data is Valid | - TRUE - Input data was refreshed within the specified Refresh Timeout <br> value. <br> - FALSE - Input data was not refreshed within the specified time. |
| Refreshed by Subnet 1 | This status variable is used to detect latent faults within a redundant <br> network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 1 within the specified <br> Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 1 within the specified <br> Refresh Timeout milliseconds. |
| Refreshed by Subnet 2 | This status variable is used to detect latent faults within a redundant <br> network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 2 within the specified <br> Refresh Timeout value. |
| - FALSE - The data failed to arrive on Subnet 2 within the specified |  |
| Refresh Timeout milliseconds. |  |


| Variable | Description |
| :---: | :---: |
| Refreshed by Subnet 6 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 6 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 6 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 7 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 7 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 7 within the specified Refresh Timeout milliseconds. |
| Refreshed by Subnet 8 | This status variable is used to detect latent faults within a redundant network. The data is simultaneously delivered over all available subnets. <br> - TRUE - The input data was refreshed by Subnet 8 within the specified Refresh Timeout value. <br> - FALSE - The data failed to arrive on Subnet 8 within the specified Refresh Timeout milliseconds. |
| Hold Last Valid Value | - When set to FALSE, forces data to the fail-safe state when the input data is invalid. <br> - When set to TRUE, allows previous data to persist when input data is invalid. |

## To configure an analog input data block

1. In the Communication View, double-click the analog input data block.

The Communication property page displays with the Data Block and Data Variables tabs.
2. In the Data Block tab, enter the required value for the Refresh Timeout and Value in Failed State.
3. Click .... to wire the required status variables.
4. (optional) To unwire a status variable, click $x$.

## See also

Peer-to-Peer Digital Inputs on page 224
Peer-to-Peer Digital Outputs on page 227
Peer-to-Peer Analog Outputs on page 228
Data Variables on page 229
Configuring Peer-to-Peer Data Blocks on page 223

## Peer-to-Peer Digital Outputs

When configuring a digital output data block, define this property:

| Property | Description |
| :--- | :--- |
| Refresh Interval | The maximum number of milliseconds allowed between successive <br> transmissions of the output data. Note that data is immediately sent <br> following any change in the output state. Possible values range from 0 to <br> 10000 ms. The default value is 2000 ms. Specify a value of 0 to send data <br> during each application cycle. |

## To configure a digital output data block

1. In the Communication View, double-click the digital output data block. The Communication property page displays with the Data Block and Data Variables tabs.
2. In the Data Block tab, enter the required value for the Refresh Interval.

## See also

Peer-to-Peer Analog Outputs on page 228
Peer-to-Peer Digital Inputs on page 224
Peer-to-Peer Analog Inputs on page 226
Data Variables on page 223
Configuring Peer-to-Peer Data Blocks on page 223

## Peer-to-Peer Analog Outputs

When configuring an analog output data block, define these properties:

| Property | Description |
| :--- | :--- |
| Refresh Interval | The maximum number of milliseconds allowed between successive <br> transmissions of the output data. Note that data is immediately sent <br> following any change in the output state. Possible values range from 0 to <br> 10000 ms. The default value is 2000 ms. Specify a value of 0 to send data <br> during each application cycle. |
| Minimum Change before Update | The minimum change in the value of the output variable required before <br> an update is sent to the peer input, regardless of the Refresh Interval. <br> Possible values range from 0 to $3.402823 E+38$. The default value is 0. <br> When the data type of the output variable is DINT, the decimals are <br> truncated. |

## To configure an analog output data block

1. In the Communication View, double-click the analog output data block. The Communication property page displays with the Data Block and Data Variables tabs.
2. In the Data Block tab, enter the required value for the Refresh Interval and Minimum Change before Update.

## See also

Peer-to-Peer Digital Outputs on page 227
Peer-to-Peer Analog Inputs on page 226
Peer-to-Peer Digital Inputs on page 224
Data Variables on page 229
Configuring Peer-to-Peer Data Blocks on page 223

## Data Variables

## To wire data variables for a data block

1. In the Communication View, double-click the required data block.

The Communication property page is displayed with the Data Block and Data Variables tabs.
2. Click the Data Variables tab.
3. For analog input and output data blocks, set the data type of the variable for a specific channel by clicking the type to toggle between DINT and REAL.
4. Click .... to wire the required data variables.
5. (optional) To unwire a data variable, click $x$.

## See also

Peer-to-Peer Digital Inputs on page 224
Peer-to-Peer Analog Inputs on page 226
Peer-to-Peer Digital Outputs on page 227
Peer-to-Peer Analog Outputs on page 228
Configuring Peer-to-Peer Data Blocks on page 223
The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software provides the option of specifying the priority of IP traffic for different applications. The Quality of Service feature ensures high priority services are not delayed during periods
of network congestion. Set the priority for up to 16 services. When using the Quality of Service, also enable TCP negotiation.


Tip: When the Quality of Service feature is enabled, the scan rate of the controller may differ by five milliseconds.

When enabled, the service uses bit patterns in the "DS-byte" of IP, which is the Type-of-Service (ToS) octet for IPv4. When defining the quality of service, set the priority of applications as high, medium, low, or as a value ranging from 0 to 255.

To set the service, arrange routers and switches able to inspect IP headers and prioritize them by the ToS header octet. The network devices then apply the rules to prioritize the IP traffic. The AADvance controller maintains the priority when responding to incoming messages. Outgoing messages are sent out according to the set priority defined in the Quality of Service.

| Important: | When using all 8 bits of the ToS byte, the networking equipment <br> may interpret the packet incorrectly. To avoid issues, it is <br> recommended to use a maximum value of 252 for ToS. |
| :--- | :--- |

## To define the quality of service

For TCP, each service using the same port must have a different peer address. For UDP, each service must use a different port. The same port number may be used for both protocols.

1. In the Communication View, expand the required controller, and then double-click Quality of Service.
The Communication property page displays.
2. In the property page, select Enable Quality of Service to enable the feature.
3. If TCP negotiation is required, select Enable TCP Negotiation.
4. In the table, set the values for the Protocol, Port, and ToS for each required service. If needed, set the Peer Address.

## See also

TCP Negotiation on page 231

## TCP Negotiation

When TCP negotiation is enabled, the TCP/IP stack applies the user-specified ToS priority during the TCP negotiation.

When the controller acts as a server, disable TCP negotiation. The controller uses the default value of 0 for ToS during the 3 -way TCP communication and then applies the defined ToS value when connected.
Whether enabled or disabled, the ToS value may change after the completion of the TCP negotiation if another service has a more specific rule for the port.

Tip: To apply a connection by ToS during the enabled TCP negotiation, define Protocol and Port values; the Peer Address must remain empty.

When setting the Quality of Service, define the following:

| Protocol | The type of protocol. <br> Possible values are Unused, UDP, or TCP. The default value is Unused. |
| :--- | :--- |
| Port | The port number. If the AADvance controller initiates the connection, the service number <br> of the device being connected to. <br> Possible values range from 0 to 65535. The default value is 0. |
| Peer Address | When the defined protocol is TCP, the address of the device being connected to. Setting <br> a peer address is optional and only required when needing to set the priority for traffic <br> from one specific device. The default value is 0.0.0.0. <br> To prioritize traffic coming from devices using the same port to communicate with the <br> AADvance controller, leave the Peer Address at 0.0.0.0. |
| ToS | The value of the priority. <br> Possible values are Low, Medium, High, or a value ranging from 0 to 255. The default <br> value is 0. <br> When setting the value of ToS to 255, the networking equipment may interpret the <br> packet incorrectly. It is recommended to use a maximum value of 254 for ToS. |

## See also

Defining the Quality of Service on page 229

## Equipment View

## To access the Equipment View

From the View menu, click Equipment View.
The Equipment View displays a graphical tree view of the application hardware. From the Equipment View, configure the system type, processor
safety time, and battery alarm for each controller. Also define the Status, Control, and RTC variables for controllers. The Equipment View contains an IO Bus 1 and IO Bus 2 for each Standard or Mixed controller defined in a project. Each IO Bus has 24 slots available for assigning I/O modules. Eurocard controllers only have IO Bus 1, containing 18 slots. The Equipment View displays the digital and analog I/O modules in their defined slots. From the Equipment View, enable the HART protocol and HART Pass-Through features for analog I/O modules.


Actions to perform from the Equipment View are:

- Configure the system type, process safety time, and battery alarm for a controller
- Wire processor variables
- Manually set the real time clock
- Add or edit I/O modules
- Configure I/O modules
- Configure the process safety time for an I/O module
- Wire a status variable to an I/O module
- Wire the channels of an I/O module
- Define threshold values for input modules
- Configure analog I/O channels to support the HART ${ }^{\circledR}$ protocol
- Configure the controller to support the HART Pass-Through feature


## To refresh the content in the Equipment View

- Select the Equipment View, then from the View menu, click Refresh.


## See also

Configuring the Controller on page 234
Configuring Processor Variables on page 236
Configure the Controller I/O on page 259
Configuring I/O Modules for AADvance Applications on page 261

## Hardware Redundancy

Input Voting

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software enables creating hardware redundancy from the Equipment View. When defining I/O modules to empty IO bus slots, there is an option of adding one, two, or three modules. Adding two (duplex arrangement) or three (triplex arrangement) modules enables redundancy, with each module carrying out the same function to improve reliability. When adding an I/O module having a duplex or triplex arrangement, the AADvance-Trusted SIS Workstation software automatically assigns the modules to the adjacent slots. For duplex or triplex groups, only configure one set of channels.

> Tip: The AADvance-Trusted SIS Workstation software automatically connects to each processor after setting the network connection.

During redundancy, the processors read all the module inputs independently through the I/O bus. The processors then share the input data, application variables, and other communication data to agree on the data to use in the application. The processors run their logic independently, then share the outputs to send to the I/O and communications. Once they agree on the data, the processors individually send the output data, and then the next cycle starts.

## See also

Input Voting on page 233
Output Voting on page 234
Digital and analog modules are processed (voted) this way:

| Valid Modules | Description |
| :--- | :--- |
| 3 | The mid-value is used by the application |
| 2 | The lower value is used by the application |
| 1 | The value is used by the application |
| 0 | The previous value is used by the application. A fault will occur after the <br> process safety time (PST). |
| Tip: | For analog output modules, the measured currents are added together <br> since they share the current. |

A valid module means the HART ${ }^{\circledR}$ and process measurements pass the integrity checks.

## See also

Hardware Redundancy on page 233
Output Voting on page 234

## Output Voting

For the commanded output data to digital and analog output modules, the application does this:

| Redundant Configuration | Description |
| :--- | :--- |
| Dual Processors | The output modules processing the integrated circuit (FPGA) <br> disregard discrepant data from the processors and the <br> outputs hold (freeze) at their last state. |
| Triple Processors | When only two of the three sets of data agree, the 2003 data <br> is used and the third set is discarded. When all three sets of <br> data disagree, the FPGAs discard all data and the outputs hold <br> (freeze) at their last state. |
| Tip: $\quad$Frozen outputs are only tolerated for the PST period. When the PST expires <br> and no valid data is received, the outputs enter the shutdown state defined <br> in the application. |  |

## See also

Hardware Redundancy on page 233
Input Voting on page 233

## Configuring the Controller

From the Equipment View, define these properties for controllers: System Type, Process Safety Time, Battery Alarm, and Remote Fault Reset.


## System Type

AADvance controllers support three types of systems: Standard, Eurocard, or Mixed.

| System Type | Description |
| :--- | :--- |
| Standard | IO Bus 1 and 2, each containing 24 slots |
| Eurocard | 10 Bus 1, containing 18 slots |
| Mixed | 10 Bus 1 and 2, each containing 24 slots |

The type of system affects the available I/O modules. The Mixed controller type enables configuring a standard controller with both Standard and Eurocard I/O modules.

[^6]
## Process Safety Time

The process safety time (PST) defines the maximum time in milliseconds that the outputs of a controller remain in the ON state after detecting important diagnostic or application faults. When the defined PST is exceeded, the outputs go into their specified safe states. Set the PST for the controller. The default PST for the controller is 2500 ms . By default, each group of I/O modules inherits the PST from the controller. Also set an alternate PST for each I/O group.

## Battery Alarm

The 9110 processor module contains a battery supporting retained variables and the Real Time Clock (RTC) when the system power is turned off. The system contains a processor battery alarm to alert that the battery voltage is low and the battery must be replaced. The battery alarm is enabled by default for Standard and Mixed controllers.

$$
\text { Tip: } \quad \text { The Battery Alarm must be disabled for Eurocard controllers. }
$$

## Remote Fault Reset

The remote fault reset enables resetting faulty processors in the controller without needing to physically press the Fault Reset button located on the T9110 front panel. As well, use the remote fault reset to join processors to a running system. The remote fault reset is required for sub-sea systems since it is impossible to physically manipulate the processors. By default, Remote Fault Reset is disabled ( $0 \times 00000000$ ). To enable the feature, a non-zero hexadecimal value must be set matching the combined values of the control integers Allow Remote Fault Reset MSB and Allow Remote Fault Reset LSB. Possible values range from $0 \times 00000001$ to 0xFFFFFFFF. When enabling Remote Fault Reset, a warning displays confirming the remote reset of the controller.

> Tip: Set the System Type before adding I/O modules. The defined PST must be larger than the execution time for the application to run. By default, the battery alarm is enabled. Disable the alarm for sub-sea systems or other systems lacking a battery. By default, the remote fault reset is disabled. It is recommended to enable the fault reset for sub-sea systems.

## To configure the controller system type, process safety time, battery alarm, and remote fault reset

1. From the Equipment View, double-click the required controller. The Equipment property page displays.
2. From the Equipment property page, click the Processor tab.
3. From the Processor tab, select the required System Type. Select one of the following:

- Standard
- Eurocard
- Mixed

The Equipment View refreshes to display the IO Buses and slots attributed to the specific system type.
4. From the Processor tab, in the Process Safety Time field, type the required value ranging between 20 ms and 60000 ms .
Tip: For large systems, set the PST for the controller to 1500 ms or more, ensuring sufficient time to educate a second or third processor module.
5. Set the Battery Alarm value to Enabled or Disabled.
6. Set the Remote Fault Reset value to $0 \times 0$ (disabled) or a non-zero hexadecimal value (enabled).

- When enabling the property, a warning displays confirming the action. Click Yes to enable Remote Fault Reset.


## See also

Configuring Processor Variables on page 236
Configure the Controller I/O on page 259
Configuring I/O Modules for AADvance Applications on page 261

## Configuring Processor Variables

The 9110 processor module contains status, control, and RTC variables used by the application. Status variables get data from the processor module while control variables set data in the processor module. RTC variables are used for the real-time clock of the application.

Configure the processor variables from the property page of the controller. Each type of processor variable is wired in a separate tab, as seen in the following:


- Status tab, contains status integers and Booleans supplying data about the controller to the application.
- Control tab, contains control integers and Booleans enabling the application to send data to the controller.
- RTC tab, contains status, program, and control variables for the real time clock. RTC status variables supply the date and time to the application. RTC program variables hold the date and time to be written to the real time clock. RTC control variables set and update the clock.

| Task | Procedure |
| :--- | :--- |
| Wire a processor variable | 1. From the Equipment View, double-click the required controller. <br> The Equipment property page displays. <br> 2. From the property page, click one of the following tabs: <br> - Status <br> - Control <br> - RTC <br> 3. Each tab displays different processor variables performing specific functions. <br> 4. From the processor variable tab, locate the required variable, and in the Wire column click <br> 5. The Variable Selector displays. <br> 6. From the Variable Selector, select the required variable and click OK. |
| Unwire a processor variable | 1. From the Equipment View, double-click the required controller. |
| The Equipment property page displays. |  |
|  | 2. From the property page, click one of the following tabs: |
| • Status |  |
| • Control |  |
| • RTC |  |
| Each tab displays different processor variables performing specific functions. |  |
| 3. From the processor variable tab, locate the wired variable, and in the Unwire column click |  |

## See also

Status Processor Variables on page 237
Control Processor Variables on page 245
Status Processor Variables
Status processor variables supply data about the controller to the application. The AADvance controller has these status processor variables:

- Status Integers
- Number of Locked Input Variables
- Number of Locked Output Variables
- Processor Module A Temperature
- Processor Module B Temperature
- Processor Module C Temperature
- Status Booleans
- System Health
- System Health Reset
- Dongle Detected (Voted)
- Processor Module A Online
- Processor Module B Online
- Processor Module C Online
- Processor Module A Health
- Processor Module B Health
- Processor Module C Health
- Processor Module A 24v1 Power Feed Health
- Processor Module B 24v1 Power Feed Health
- Processor Module C 24v1 Power Feed Health
- Processor Module A 24v2 Power Feed Health
- Processor Module B 24v2 Power Feed Health
- Processor Module C 24v2 Power Feed Health
- Processor Module A Ready
- Processor Module B Ready
- Processor Module C Ready
- Processor Module A NVRAM Battery Health
- Processor Module B NVRAM Battery Health
- Processor Module C NVRAM Battery Health


## See also

Configuring Processor Variables on page 236
Control Processor Variables on page 245

## Number of Locked Input Variables

## See also

Status Processor Variables on page 237

## Number of Locked Output Variables

## See also

Status Processor Variables on page 237
Reports the temperature, in degrees Celsius, of a specific slot on the 9110 processor module. Set to 0 (zero) if no processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 65,535 |
| Wiring | \%IW1.0.2 |

## See also

Status Processor Variables on page 237

## Processor Module B <br> Temperature

## Processor Module C Temperature

Reports the temperature, in degrees Celsius, of a specific slot on the 9110 processor module. Set to 0 (zero) if no processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 65,535 |
| Wiring | $\%$ IW1.0.3 |

## See also

## Status Processor Variables on page 237

Reports the temperature, in degrees Celsius, of a specific slot on the 9110 processor module. Set to 0 (zero) if no processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 65,535 |
| Wiring | $\%$ IW1.0.4 |

## See also

Status Processor Variables on page 237
Displays the system health for processor and I/O modules.

| Direction | Input to the application from the controller |
| :---: | :---: |
| Type | Boolean |
| Values | - TRUE - All installed processor and I/O modules are healthy and the Set System Health Alarm variable is FALSE (See Control Booleans). The System Healthy LED of the processor module is green. <br> - FALSE - One or more of the installed processors or I/0 modules are reporting a module health problem or the Set System Health Alarm variable is TRUE (See Control Booleans). The System Healthy LED of the processor module is red. |
| Wiring | \%\|X1.2.0 |
| Tip: After a fault, reset the Set System Health Alarm in a system containing at least one healthy processor or I/O module in a module group. |  |

## See also

Status Processor Variables on page 237
Reports that the fault reset button on a processor module was pressed. The System Health Reset is triggered by pressing the button, but the value does not change to TRUE until the start of the next application cycle. The value remains TRUE for the cycle and then reverts to FALSE even if the button was pressed throughout the cycle.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |


| Values | $\bullet$ TRUE - The fault reset button on a 9110 processor module was <br> pressed in an earlier cycle. <br> $\bullet$ <br> - FALSE - No fault reset button is active. <br> The default value is FALSE. |
| :--- | :--- |
| Wiring | $\%$ IX1.2.1 |

## See also

## Status Processor Variables on page 237

## Dongle Detected (Voted)

Reports the presence or absence of a program enable key.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - One or more 9110 processor modules detects that <br> the program enable key is inserted in the KEY connector of <br> the 9100 processor base unit. |
|  | - FALSE - The processor modules are unable to detect a <br> program enable key. |
| Wiring | \%IX1.2.2 |

## See also

Status Processor Variables on page 237

## Processor Module A Online

Reports that a processor module in a duplex or triplex redundant configuration is present and is communicating through the inter-processor link to one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The 9110 processor module in the given slot is online |
|  | - FALSE - The processor module is offline <br> The default value is TRUE. |
| Wiring | \%IX1.2.5 |

## See also

## Status Processor Variables on page 237

Reports that a processor module in a duplex or triplex redundant configuration is present and is communicating through the inter-processor link to one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The 9110 processor module in the given slot is online |
|  | - FALSE - The processor module is offline <br> The default value is TRUE. |
| Wiring | \%IX1.2.6 |

## See also

## Status Processor Variables on page 237

## Processor Module C Online

Reports that a processor module in a duplex or triplex redundant configuration is present and is communicating through the inter-processor link to one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The 9110 processor module in the given slot is online |
|  | • FALSE - The processor module is offline |
| The default value is TRUE. |  |
| Wiring | \%IX1.2.7 |

## See also

Status Processor Variables on page 237
Processor Module A Health
Reports the health status of a processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The 9110 processor module in the given slot is healthy <br> and the Healthy LED indicator is green. <br> - FALSE - The processor module is faulty and the Healthy LED <br> indicator is red. |
| Wiring | \%IX1.2.8 |

## See also

## Status Processor Variables on page 237

Reports the health status of a processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The 9110 processor module in the given slot is healthy <br> and the Healthy LED indicator is green. <br> - FALSE - The processor module is faulty and the Healthy LED <br> indicator is red. |
| Wiring | \%IX1.2.9 |

## See also

Status Processor Variables on page 237

## Processor Module C Health

Reports the health status of a processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |


| Values | • TRUE - The 9110 processor module in the given slot is healthy <br> and the Healthy LED indicator is green. <br> $\bullet$ <br> FALSE - The processor module is faulty and the Healthy LED <br> indicator is red. |
| :--- | :--- |
| Wiring | $\%$ IX1.2.10 |

## See also

Status Processor Variables on page 237

## Processor Module A 24v1

 Power Feed HealthReports the health of power feed 1 (nominal 24 Vdc ) in the given slot of the 9110 processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The power feed voltage is within specifications (18 to |
|  | 32 Vdc). |
|  | • FALSE - The power feed is outside specifications. |
| Wiring | \%XX1.2.11 |

## See also

## Status Processor Variables on page 237

Reports the health of power feed 1 (nominal 24 Vdc ) in the given slot of the 9110 processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The power feed voltage is within specifications (18 <br> to 32 Vdc). |
|  | - FALSE - The power feed is outside specifications. |
| Wiring | \%IX1.2.12 |

## See also

## Status Processor Variables on page 237

Reports the health of power feed 1 (nominal 24 Vdc ) in the given slot of the 9110 processor module

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The power feed voltage is within specifications (18 to |
|  | 32 Vdc). |
|  | • FALSE - The power feed is outside specifications. |
| Wiring | \%IX1.2.13 |

## See also

Status Processor Variables on page 237

Processor Module A 24v2 Power Feed Health

Reports the health of power feed 2 (nominal 24 Vdc ) in the given slot of the 9110 processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | $\bullet$ TRUE - The power feed voltage is within specifications (18 to |
|  | 32 Vdc ). <br>  <br>  <br>  <br> Wiring |

## See also

Status Processor Variables on page 237
Reports the health of power feed 2 (nominal 24 Vdc ) in the given slot of the 9110 processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The power feed voltage is within specifications (18 to |
|  | 32 Vdc). |
|  | • FALSE - The power feed is outside specifications. |
| Wiring | \%IX1.2.15 |

## See also

Status Processor Variables on page 237
Reports the health of power feed 2 (nominal 24 Vdc ) in the given slot of the 9110 processor module.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The power feed voltage is within specifications (18 to <br> 32 Vdc). <br>  <br>  <br> - FALSE - The power feed is outside specifications. |
| Wiring | \%IX1.2.16 |

## See also

## Status Processor Variables on page 237

Reports that a processor module in a duplex or triplex redundant configuration is present and is synchronized with one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The 9110 processor module in the given slot is <br> synchronized. <br> $\bullet$ FALSE - The processor module is out of synchronization or <br> missing. |
| Wiring | \%IX1.2.17 |

## See also

Status Processor Variables on page 237

## Processor Module B Ready

## Processor Module C Ready

Processor Module A NVRAM Battery Health

## See also

Status Processor Variables on page 237
Reports that a processor module in a duplex or triplex redundant configuration is present and is synchronized with one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The 9110 processor module in the given slot is <br> synchronized. <br> - FALSE - The processor module is out of synchronization or <br> missing. |
| Wiring | $\%$ IX1.2.19 |

## See also

Status Processor Variables on page 237
Reports that a processor module in a duplex or triplex redundant configuration is present and is synchronized with one or both of its peers. Reports that a simplex processor module is present.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The 9110 processor module in the given slot is <br> synchronized. <br> - FALSE - The processor module is out of synchronization or <br> missing. |
| Wiring | \%IX1.2.18 |

Reports the health status of the back-up battery in a processor module. The battery voltage is verified during start up and then every 24 hours.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The back-up battery in the given slot of the 9110 <br> processor module is present and the voltage is within <br> satisfactory limits. |
|  | -FALSE - The voltage of the back-up battery is low or the <br> battery is missing. <br> Wiring |

## See also

Status Processor Variables on page 237

Processor Module B NVRAM Battery Health

Reports the health status of the back-up battery in a processor module. The battery voltage is verified during start up and then every 24 hours.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The back-up battery in the given slot of the 9110 <br> processor module is present and the voltage is within <br> satisfactory limits. |
|  | -FALSE - The voltage of the back-up battery is low or the <br> battery is missing. <br> Wiring <br> \%IX1.2.27 |

## See also

## Status Processor Variables on page 237

Reports the health status of the back-up battery in a processor module. The battery voltage is verified during start up and then every 24 hours.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The back-up battery in the given slot of the 9110 <br> processor module is present and the voltage is within <br> satisfactory limits. <br> • FALSE - The voltage of the back-up battery is low or the <br> battery is missing. |
| Wiring | \%IX1.2.28 |

## See also

Status Processor Variables on page 237
Control Processor Variables
Control processor variables enable the application to send data to the controller. The AADvance controller has these control processor variables:

- Control Integers
- AUX LED Color
- Allow Remote Fault Reset MSB
- Allow Remote Fault Reset LSB
- Control Booleans
- Unlock All Locked Variables
- Set System Health Alarm
- HART® Pass-Through
- Perform Remote Fault Reset
- Perform Remote Fault Join


## See also

Configuring Processor Variables on page 236
Sets the state of the Aux LED indicator on all 9110 processor modules.

| Direction | Output from the application to the controller |
| :--- | :--- |


| Type | WORD |
| :--- | :--- |
| Values | $\bullet 0=$ off |
|  | $\bullet 1$ = red |
|  | $\bullet 2$ = green |
|  | $\bullet 3=$ amber |
|  | The default value is 0 |
| Wiring | \%QW1.1.0 |

## See also

## Control Processor Variables on page 245

## Allow Remote Fault Reset MSB (Most Significant Bits)

Used in combination with Allow Remote Fault Reset LSB, enables remotely resetting a processor or enables joining a processor to a running system. For a successful remote fault reset or join, the value entered for the Remote Fault Reset must match the combined values set for the control integers Allow Remote Fault Reset MSB and Allow Remote Fault Reset LMB. As well, the control Booleans Perform Remote Fault Reset and Perform Remote Fault Join must be set to TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | WORD |
| Values | Possible values range from 0 to 65535. |
| Wiring | \%QW1.1.1 |

## See also

Control Processor Variables on page 245
Used in combination with Allow Remote Fault Reset MSB, enables remotely resetting a processor or enables joining a processor to a running system. For a successful remote fault reset or join, the value entered for the Remote Fault Reset must match the combined values set for the control integers Allow Remote Fault Reset MSB and Allow Remote Fault Reset LSB. As well, the control booleans Perform Remote Fault Reset and Perform Remote Fault Join must be set to TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | WORD |
| Values | Possible values range from 0 to 65535. |
| Wiring | \%QW1.1.2 |

## See also

Control Processor Variables on page 245
Unlock All Locked Variables
Removes all user locks on input and output variables.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |


| Values | $\bullet$ TRUE - Remove all locks <br> • FALSE - No effect <br> The default value is FALSE |
| :--- | :--- |
| Wiring | \%QX1.3.0 |

## See also

Control Processor Variables on page 245

## Set System Health Alarm

## HART Pass-Through

Sends a system health alarm signal from the application to the controller.

| Direction | Output from the application to the controller |
| :---: | :---: |
| Type | Boolean |
| Values | - TRUE - When the variable transitions from FALSE to TRUE, the system responds as if a system level fault was found. The System Healthy LED of the processor module is set to RED and the System Health processor variable is set to TRUE. When the variable is TRUE, the fault is immediately re-annunciated after pressing the Fault Reset button. <br> - FALSE - No alarm signal is sent to the controller. After the variable transitions from TRUE to FALSE, the System Healthy LED of the processor module turns GREEN once the Fault Reset button is pressed. <br> The default value is FALSE. |
| Wiring | \%QX1.3.1 |

## See also

Control Processor Variables on page 245
Starts the HART ${ }^{\circledR}$ Pass-Through feature enabling HART messages on analog input and output modules. The system allows messages on each channel both independently and together.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - HART Pass-Through is enabled and available for an analog $1 / 0$ module. <br>  <br>  <br> - FALSE - HART Pass-Through is disabled and unavailable. <br> The default value is FALSE. |
| Wiring | \%QX1.3.2 |

## See also

## Control Processor Variables on page 245

Perform Remote Fault Reset
When a rising edge is detected, resets the faulty processors.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - Resets the faulty processors. <br>  <br>  <br>  <br>  <br> - FALSE - Remotely resetting processors is disabled. <br> The default value is FALSE. |
| \%iring | \%X1.3.3 |

## See also

Control Processor Variables on page 245

Perform Remote Fault Join
When a rising edge is detected, the processors join the running system.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The processors join the running system |
|  | • FALSE - Remotely joining processors to a running system is disabled. <br> The default value is FALSE. |
| Wiring | $\%$ QX1.3.4 |

## See also

Control Processor Variables on page 245
RTC processor variables manage information regarding the real time clock. The AADvance controller has the following RTC processor variables:

- RTC Status Variables: Year, Month, Day of Month, Hours, Minutes, Seconds, Milliseconds
- RTC Program Variables: Year, Month, Day of Month, Hours, Minutes, Seconds, Milliseconds
- RTC Control Variables: RTC Write, RTC Read, Year, Month, Day of Month, Hours, Minutes, Seconds, Milliseconds


## See also

Configuring Processor Variables on page 236
The AADvance controller contains a real-time clock (RTC) used to make a record of the time and date of system events, SOE services, and log files. Manually set the real time clock of the processor by using RTC variables or use SNTP from the Communication View.

| Important: | It is recommended to use SNTP to synchronize controllers so all |
| :--- | :--- |
| time-related events, services, and logs use the same time of day. |  |

## To manually set the real time clock

1. From the Build menu, click Build All.
2. From the Project menu, either click Download or Update.
3. From the Online menu, click Connect.
4. Force the following RTC Control Variables to TRUE, therefore requesting IXL restricted access: RTC Read, Year, Month, Day of Month, Hours, Minutes, Seconds, and Milliseconds.
Important: Do not force RTC Write at this point, the value should be FALSE.

The RTC Status variables now display the current date and time in the processor.
5. Set RTC Read to FALSE.
6. Double-click each RTC Program Variable (Year, Month, Day of Month, Hours, Minutes, Seconds, and Milliseconds), specify a value, and then click Write.
7. To enable writing the new date and time to the processor, set RTC Write to TRUE and then FALSE.
8. Lock and force the value of RTC Read to TRUE.

The RTC Status variables now display the set date and time in the processor.

## See also

RTC Processor Variables and the Real Time Clock on page 248

## RTC Status: Year

## RTC Status: Month

Reports the oldest value of the real-time clock (RTC) year as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Year updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Year = TRUE
The value of RTC Status: Year is 0 (zero) when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Year $=$ FALSE
The value of RTC Status: Year is not updated when the variable is set to:
RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 1,900 to 2,299 |
| Wiring | $\%$ IW1.4.0 |

## See also

## RTC Control: RTC Read on page 256

RTC Program: Year on page 253
RTC Control: Year on page 256
Reports the oldest value of the real-time clock (RTC) month as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Month updates when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Month = TRUE
The value of RTC Status: Month is 0 (zero) when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Month = FALSE

The value of RTC Status: Month is not updated when the variable is set to:
RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 1 to 12, or 0 |
| Wiring | $\%$ IW1.4.1 |

## See also

RTC Control: RTC Read on page 256
RTC Program: Month on page 257
RTC Control: Month on page 257

## RTC Status: Day of Month

## See also

RTC Control: RTC Read on page 256
RTC Program: Day of Month on page 253
RTC Control: Day of Month on page 257
Reports the oldest value of the real-time clock (RTC) hours as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Hours updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Hours = TRUE

The value of RTC Status: Hours is 0 (zero) when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Hours = FALSE
The value of RTC Status: Hours is not updated when the variable is set to:
RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 23 |
| Wiring | \%IW1.4.3 |

## See also

RTC Control: RTC Read on page 256
RTC Program: Hours on page 253
RTC Control: Hours on page 257
Reports the oldest value of real-time clock (RTC) minutes as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Minutes updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Minutes $=$ TRUE
The value of RTC Status: Minutes is 0 (zero) when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Minutes = FALSE
The value of RTC Status: Minutes is not updated when the variable is set to:
RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 59 |
| Wiring | \%IW1.4.4 |

## See also

RTC Control: RTC Read on page 256
RTC Program: Minutes on page 254
RTC Control: Minutes on page 258
Reports the oldest value of real-time clock (RTC) seconds as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Seconds updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Seconds = TRUE
The value of RTC Status: Seconds is 0 (zero) when these variables are set to:
RTC Control: RTC Read = TRUE
RTC Control: Seconds = FALSE
The value of RTC Status: Seconds is not updated when the variable is set to:
RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 59 |
| Wiring | $\%$ IW. 1.5 |

## See also

RTC Control: RTC Read on page 256
RTC Program: Seconds on page 254
RTC Control: Seconds on page 259

## RTC Status: Milliseconds

Reports the oldest value of real-time clock (RTC) milliseconds as voted by the presently synchronized 9110 processor modules. The value of the RTC Status variable Milliseconds updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Milliseconds= TRUE
The value of RTC Status: Milliseconds is 0 (zero) when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Milliseconds = FALSE
The value of RTC Status: Milliseconds is not updated when the variable is set to:

RTC Control: RTC Read = FALSE

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 999 |
| Wiring | \%IW1.4.6 |

## See also

RTC Control: RTC Read on page 256
RTC Program: Milliseconds on page 255
RTC Control: Milliseconds on page 259

## RTC Program: Year

## RTC Program: Month

Specifies the year to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Year is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 1,900 to 2,299 <br> The default value is 0 |
| Wiring | \%QW1.5.0 |

## See also

RTC Control: RTC Write on page 255
RTC Status: Year on page 249
RTC Control: Year on page 256
Specifies the number of the month to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Month is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 1 to 12, or 0 <br> The default value is 0 (zero) |
| Wiring | \%@W1.5.1 |

## See also

RTC Control: RTC Write on page 255
RTC Control: Month on page 257
RTC Status: Month on page 249
Specifies the day of the month to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Day is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 1 to 31, or 0 <br> The default value is 0 (zero) |
| Wiring | \%@W1.5.2 |

## See also

RTC Control: RTC Write on page 255
RTC Control: Day of Month on page 257
RTC Status: Day of Month on page 250

## RTC Program: Hours

Specifies the time of day (in hours) to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Hours is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 23 <br> The default value is 0 (zero) |
| Wiring | \%OW1.5.3 |

## See also

RTC Control: RTC Write on page 255
RTC Status: Hours on page 250
RTC Control: Hours on page 257
Specifies the time of day (in minutes) to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Minutes is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 59 <br> The default value is 0 (zero) |
| Wiring | \%OW1.5.4 |

## See also

RTC Control: RTC Write on page 255
RTC Status: Minutes on page 251
RTC Control: Minutes on page 258
Specifies the time of day (in seconds) to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Seconds is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 59 <br> The default value is 0 (zero) |
| Wiring | \%QW1.5.5 |

## See also

RTC Control: RTC Write on page 255
RTC Status: Seconds on page 251
RTC Control: Seconds on page 259

## RTC Program: Milliseconds

Specifies the time of day (in milliseconds) to write to the real-time clock when the RTC Control variable RTC Write is set to TRUE. The value is only written if the RTC Control variable Milliseconds is TRUE.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Word |
| Values | 0 to 999 <br> The default value is 0 (zero) |
| Wiring | \%QW1.5.6 |

## See also

RTC Control: RTC Write on page 255
RTC Status: Milliseconds on page 252
RTC Control: Milliseconds on page 259
Sets new values for the real-time clock. There are seven values, all specified by the following RTC Program variables: Year, Month, Day of Month, Hours, Minutes, Seconds, and Milliseconds. Each value is set only if the related RTC Control Variable (which is a boolean, and similarly named Year, Month, Day of Month, Hours, Minutes, Seconds, or Milliseconds) is TRUE.

The change is started by the transition of the variable from FALSE to TRUE. If the application holds the TRUE state until the end of the application cycle (or longer), the application makes the change at the end of the cycle. There is no time limit on returning the value from TRUE to FALSE after the clock is updated. If the application returns the variable to FALSE before the end of the application cycle, the clock is not updated

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - Applies new values to the real-time clock <br> - FALSE -No effect <br> The default value is FALSE. |
| Wiring | \%QX1.6.0 |

## Example

Consider the following scenario:
The date is 28 th October 2008, 8 hours, 12 minutes and 35 seconds
RTC Control RTC Read is TRUE
RTC Control Year, Month, and Day of Month are TRUE
RTC Control Hours, Minutes, and Seconds are TRUE
The RTC Status variables are returned, and the real-time clock is set as follows:

$$
\text { Year }=2008
$$

$$
\text { Month }=10
$$

Day $=28$
Hours $=8$
Minutes $=12$
Seconds $=35$

## See also

RTC Control: RTC Read on page 256

## RTC Control: RTC Read

Verifies if the RTC Status variables (Year, Month, Day of Month, Hours, Minutes, and Seconds) update in real time.

All RTC Status variables must be set to TRUE when the RTC Read variable is set to TRUE, otherwise the real-time clock is not updated.

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The controller updates the RTC Status values on each application cycle. |
|  | - FALSE - The RTC Status values are static (no update occurs). <br> The default value is FALSE |
| Wiring | \%QX1.6.1 |

## See also

RTC Control: RTC Write on page 255
Verifies if the value of the RTC Program variable Year must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Year updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Year = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The RTC program Year is applied by RTC Write. <br>  <br>  <br> • FALSE - The RTC program Year is ignored. <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%OX1.6.2 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Year on page 249
RTC Program: Year on page 253

## RTC Control: Month

Verifies if the value of the RTC Program variable Month must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Month updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Month = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The RTC Program Month is applied by RTC Write. <br> - FALSE - The RTC Program Month is ignored. <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%QX1.6.3 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Month on page 249
RTC Program: Month on page 253

## RTC Control: Day of Month

Verifies if the value of the RTC Program variable Day of Month must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Day of Month updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Day of Month = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The RTC Program Day of Month is applied by RTC Write. |
|  | • FALSE - The RTC Program Day of Month is ignored. <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%QX1.6.4 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Day of Month on page 250
RTC Program: Day of Month on page 253

## RTC Control: Hours

Verifies if the value of the RTC Program variable Hours must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Hours updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Hours = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The RTC Program Hours is applied by RTC Write. <br> $\bullet$ <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%QX1.6.5 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Hours on page 250
RTC Program: Hours on page 253
Verifies if the value of the RTC Program variable Minutes must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Minutes updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Minutes = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - The RTC Program Minutes is applied by RTC Write. <br>  <br>  <br> • FALSE - The RTC Program Minutes is ignored. <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%OX1.6.6 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Minutes on page 251
RTC Program: Minutes on page 254

## RTC Control: Seconds

## RTC Control: Milliseconds

Verifies if the value of the RTC Program variable Seconds must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Seconds updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Seconds = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - The RTC Program Seconds is applied by RTC Write. |
|  | • FALSE - The RTC Program Seconds is ignored. |
| The default value is FALSE until an initial value is specified in the application. |  |
| Wiring | $\%$ QX1.6.7 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Seconds on page 251
RTC Program: Seconds on page 254
Verifies if the value of the RTC Program variable Milliseconds must be applied to the real-time clock the next time the RTC Control variable RTC Write is set to TRUE.

The value of the RTC Status variable Milliseconds updates when these variables are set to:

RTC Control: RTC Read = TRUE
RTC Control: Milliseconds = TRUE

| Direction | Output from the application to the controller |
| :--- | :--- |
| Type | Boolean |
| Values | • TRUE - RTC Program Milliseconds is applied by RTC Write. <br> • FALSE - RTC Program Milliseconds is ignored. <br> The default value is FALSE until an initial value is specified in the application. |
| Wiring | \%Q̣X1.6.8 |

## See also

RTC Control: RTC Write on page 255
RTC Control: RTC Read on page 256
RTC Status: Milliseconds on page 252
RTC Program: Milliseconds on page 255

## Configure the Controller I/O

Use the Equipment View to add I/O modules to controllers, mimicking the hardware arrangement of the modules. I/O modules are available in a simplex
arrangement or in a redundant group of two or three modules. When selecting a redundant group, modules are automatically assigned to the free adjacent slots.

Tip: In anticipation of modifications to the I/O hardware, add a duplex/triplex group for a simplex arrangement or a triplex group for a duplex arrangement. If required, also create a hot swap.

When configuring the controller, select the required system type. Define the system type before adding any I/O modules. Different I/O modules are available depending on the selected system type. When the system type is Mixed, both Standard and Eurocard I/O modules are available.

Perform the following tasks for I/O modules:

| Task | Procedure |
| :---: | :---: |
| Add an I/O module to a controller | 1. From the Equipment View, expand the required controller and IO Bus to display the slots. <br> 2. Right-click the required slot, and then click Assign I/O Module. <br> - When adding duplex or triplex redundant groups, ensure sufficient free slots are available adjacent to the required slot. Also right-click an occupied slot and then click Insert Empty Slot. <br> 3. From the Select an I/O Module dialog box, select the required module and termination assembly, then click OK. <br> The I/O module is added to the controller and needs to be configured. |
| Move an I/O module to a different slot | - From the Equipment View, expand the IO bus, then click and drag the I/O module to the empty slot. Alternately cut and paste the I/O module to the slot. <br> Tip: Dragging an I/O module to an existing slot may result in wired variables losing their wiring. |
| Remove an I/O module | - From the Equipment View, expand the IO bus, then right-click the required I/O module and click Delete. |
| Tip: In an I/O module having a duplex or triplex arrangement, wire the required I/O variables and add them to a dedicated spy list to see which processor is at fault when a fault occurs. |  |

## See also

Standard or Mixed System Type on page 260
Eurocard or Mixed System Type on page 261

Standard or Mixed System Type

| Module | Channels | Termination Assemblies |
| :--- | :--- | :--- |
| 9401 24Vdc Digital Input Module | 8 | 9801 Digital Input TA - Simplex <br> 9802 Digital Input TA - Duplex <br> 9803 Digital Input TA - Triplex |
| 9402 24Vdc Digital Input Module | 16 | 9801 Digital Input TA - Simplex <br> 9802 Digital Input TA - Duplex <br> 9803 Digital Input TA - Triplex |
| $943124 V$ dc Analog Input Module | 8 | 9831 Analog Input TA - Simplex <br> 9832 Analog Input TA - Duplex <br> 9803 Analog Input TA - Triplex |
| 9432 24Vdc Analog Input Module | 9831 Analog Input TA - Simplex <br> 9832 Analog Input TA - Duplex <br> 9803 Analog Input TA - Triplex |  |
| 9451 24Vdc Digital Output Module | 16 | 9851 Digital Output 24Vdc TA - Simplex <br> 9852 Digital Output 24Vdc TA - Duplex |


| Module | Channels | Termination Assemblies |
| :--- | :--- | :--- |
| 9481 4-20mA Analog Output Module | 3 | 9881 Analog Output TA - Simplex <br> 9882 <br>  |
| 9482 Analog Output TA - Duplex |  |  |

## See also

Configure the Controller I/O on page 259

## Eurocard or Mixed System Type

These I/O modules are available when the controller system type is Eurocard or Mixed:

| Module | Channels | Termination Assemblies |
| :--- | :--- | :--- |
| 9501 24Vdc Digital Input Module | 8 | 9801 Digital Input TA - Simplex <br> 9802 Digital Input TA - Duplex <br> 9803 Digital Input TA - Triplex |
| 9531 24Vdc Analog Input Module | 8 | 9831 Analog Input TA - Simplex <br> 9832 Analog Input TA - Duplex <br> 9803 Analog Input TA - Triplex |
| 9551 24Vdc Digital Output Module | 8 | 9851 Digital Output 24Vdc TA - Simplex <br> 9852 <br> Digital Output 24Vdc TA - Duplex |
| 9581 4-20mA Analog Output Module | 9881 Analog Output TA - Simplex <br> 9882 Analog Output TA - Duplex |  |

## See also

Configure the Controller I/O on page 259

## Configuring I/O Modules for AADvance Applications

Configure the modules after adding the required I/O modules to the I/O bus of a controller. The following are the types of I/O modules in an AADvance controller, each requiring different information to complete their configuration:

- Digital Inputs
- Analog Inputs
- Digital Outputs
- Analog Outputs

For all types of I/O modules, configure the process safety time (PST) and status variable.

When configuring I/O modules, define the process safety time (PST). The process safety time (PST) defines the maximum time in milliseconds that the outputs of a controller remain in the ON state after detecting important diagnostic or application faults. When the defined PST is exceeded, the outputs go into their specified safe states. By default, the PST is inherited from the controller. Also specify a smaller PST for each I/O group.

When configuring an I/O module, use a status variable to get data from the module such as the health, ready, online, run, and shutdown statuses. The status variable must have the structure data type T9K_TA_GROUP_STATUS.

| Task | Procedure |
| :---: | :---: |
| Configure process safety time for an I/O module | 1. From the Equipment View, expand the 10 bus and double-click the required I/O module. <br> The Equipment property page displays. <br> 2. From the Equipment property page, in the module-specific tab, clear the Inherit option. <br> 3. In the Process Safety Time field, type the required value ranging between 20 ms and 60000 ms . The value must be smaller than the PST set for the controller. |
| Wire a status variable to an I/0 module | 1. In the Dictionary, declare a variable with the following properties: <br> - Data Type: TgK_TA_GROUP_STATUS <br> - Direction: Varlnput <br> 2. In the Equipment View, expand the IO bus and double-click the required I/0 module. <br> The Equipment property page displays. <br> 3. From the Equipment property page, in the module-specific tab, click $\square$ in the Status Variable field. The Variable Selector displays. <br> 4. In the Variable Selector, select the required variable and then click OK. |

The following example displays the module-specific tab to define the PST and wire the status variable for an I/O module:


## See also

Configuring the Controller on page 234
Digital Input Modules on page 265
Analog Input Modules on page 269
Digital Output Modules on page 278
Analog Output Modules on page 286

## T9K_TA_GROUP_STATUS

The structure type T9K_TA_GROUP_STATUS contains the following members (fields):

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.EXPC | DINT | Number of Modules Expected | Retrieves the quantity of $1 / 0$ modules defined in the configuration of the group. The possible values are 1,2 or 3 . |
| <tagname>.ACT | DINT | Number of Modules Online | Retrieves the quantity of $1 / 0$ modules that are installed, powered, locked and communicating over the I/O bus. <br> The possible values are 1,2 or 3 . |
| <tagname>.LOC | DINT | Slot location of first module in group | Retrieves the number of the slot allocated to the first I/0 module in a group. <br> For Eurocard systems, the possible values are 1 to 18. For Standard or Mixed systems, the possible values are 1 to 24 . <br> Tip: The I/O bus number is not specified. |
| <tagname>.GH | BOOL | Group Health FALSE if any module in fault | Retrieves the general health status of all $1 / 0$ modules in a group. <br> - TRUE - All modules are healthy <br> - FALSE - One or more modules in the group is online and reporting a fault |
| <tagname>.AONL | B00L | A Module Online Status | Retrieves the online status of module A <br> - TRUE - The I/0 module is installed, powered, locked, and is communicating over the I/0 bus <br> - FALSE - The I/O module is unable to communicate over the I/O bus. |
| <tagname>.AHLY | BOOL | A Module Health Status | Retrieves the general health of module A <br> - TRUE - The module is online and has no faults <br> - FALSE - The module is either offline or reporting a fault |
| <tagname>.ARDY | B00L | A Module Ready Status | Retrieves the ready status of module A. <br> - TRUE - The module is online and ready to send channel values <br> - FALSE - The module is either offline or not ready to send channel values |
| <tagname>.ARUN | BOOL | A Module Run Status | Retrieves the run status of module A <br> - TRUE - The module is online and reporting channel values or requires manual intervention by pressing the Fault Reset button before values can be reported. <br> - FALSE - The module is either offline or online but is not reporting channel values. |
| <tagname>.ASDN | BOOL | A Module Shutdown Status | Retrieves the shutdown status of module A <br> - TRUE - Press the Fault Reset button for the module to send values <br> - FALSE - No action required |
| <tagname>.APOS | DINT | A Module Position | Retrieves the slot number of module A <br> For Eurocard systems, the possible values are 1 to 18. For Standard or Mixed systems, the possible values are 1 to 24 . <br> Tip: The I/O bus number is not specified. |
| <tagname>.BONL | B00L | B Module Online Status | Retrieves the online status of module B <br> - TRUE - The I/O module is installed, powered, locked, and is communicating over the I/0 bus <br> - FALSE - The I/O module is unable to communicate over the I/O bus. |
| <tagname>.BHLY | BOOL | B Module Health Status | Retrieves the general health of module $B$ <br> - TRUE - The module is online and has no faults <br> - FALSE - The module is either offline or reporting a fault |
| <tagname>.BRDY | BOOL | B Module Ready Status | Retrieves the ready status of module B <br> - TRUE - The module is online and ready to send channel values <br> - FALSE - The module is either offline or not ready to send channel values |
| <tagname>.BRUN | BOOL | B Module Run Status | Retrieves the run status of module $B$ <br> - TRUE - The module is online and reporting channel values or requires manual intervention by pressing the Fault Reset button before values can be reported. <br> - FALSE - The module is either offline or online but is not reporting channel values. |
| <tagname>.BSDN | BOOL | B Module Shutdown Status | Retrieves the shutdown status of module $B$ <br> - TRUE - Press the Fault Reset button for the module to send values <br> - FALSE - No action required |
| <tagname>.BPOS | DINT | B Module Position | Retrieves the slot number of module B <br> For Eurocard systems, the possible values are 1 to 18 . For Standard or Mixed systems, the possible values are 1 to 24 . <br> Tip: The I/O bus number is not specified. |

Chapter 2 Working with AADvance Applications

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.CONL | B00L | C Module Online Status | Retrieves the online status of module C <br> - TRUE - The I/O module is installed, powered, locked, and is communicating over the I/O bus <br> - FALSE - The I/O module is unable to communicate over the I/O bus. |
| <tagname>.CHLY | B00L | C Module Health Status | Retrieves the general health of module C <br> - TRUE - The module is online and has no faults <br> - FALSE - The module is either offline or reporting a fault |
| <tagname>.CRDY | B00L | C Module Ready Status | Retrieves the ready status of module C <br> - TRUE - The module is online and ready to send channel values <br> - FALSE - The module is either offline or not ready to send channel values |
| <tagname>.CRUN | B00L | C Module Run Status | Retrieves the run status of module C <br> - TRUE - The module is online and reporting channel values or requires manual intervention by pressing the Fault Reset button before values can be reported. <br> - FALSE - The module is either offline or online but is not reporting channel values. |
| <tagname>.CSDN | B00L | C Module Shutdown Status | Retrieves the shutdown status of module C <br> - TRUE - Press the Fault Reset button for the module to send values <br> - FALSE - No action required |
| <tagname>.CPOS | DINT | C Module Position | Retrieves the slot number of module C <br> For Eurocard systems, the possible values are 1 to 18 . For Standard or Mixed systems, the possible values are 1 to 24 . <br> Tip: The I/0 bus number is not specified. |

Tip: $\quad$ The first module in a termination assembly is defined as $A$, the second as $B$, and the third as C. The T9K_TA_GROUP_STATUS structure type always includes the fields for modules $A, B$, and $C$ even if the termination assembly only contains a simplex or duplex arrangement.

## See also

Configuring I/O Modules for AADvance Applications on page 261

## Series 9000 Structure Data Types

Choose between two types of variables structures when wiring to I/O channels: full and compact. When wiring variables to I/O channels, declare variables using the full or compact structure data types applicable to the I/O module. Also only wire the first element by declaring a variable of the required data type. For example, to only wire the first element of the 9431 24 Vdc Al channel, declare a variable of the REAL data type.

Wiring a variable with the full or compact structure type creates a set of variable structure member with the same <tagname>. The variable members wired to the I/O channel are dependent on the chosen structure type. The syntax for a structure member variable is <tagname>. $X X$, where <tagname> represents the name of the variable and $X X$ represents the structure member. For example, <tagname>.DI is a Boolean reporting the digital input state for a channel.

Structure data types used for wiring I/O variables to I/O channels are:

| Structure Type | Definition |
| :--- | :--- |
| T9K_DI_FULL | Series $9000^{\text {TM }}$ digital input channel |
| T9K_DI_COMPACT | Series 9000 digital input channel |
| T9K_Al_FULL | Series 9000 analog input channel |
| T9K_Al_COMPACT | Series 9000 analog input channel |


| Structure Type | Definition |
| :--- | :--- |
| T9K_Al_HART | Series 9000 HART® data for an analog input channel |
| TgK_Al_HART_FULL | Series 9000 HART data for an analog input channel |
| T9K_DO_FULL | Series 9000 digital output channel |
| T9K_DO_COMPACT | Series 9000 digital output channel |
| TGK_AO_FULL | Series 9000 analog output channel |
| T9K_AO_COMPACT | Series 9000 analog output channel |

## See also

Configuring I/O Modules for AADvance Applications on page 261
T9K TA GROUP STATUS on page 263
For digital input modules, configure the following:

- Process Safety Time (see Configuring I/O Modules)
- Status Variable (see Configuring I/O Modules)
- Channels
- Threshold Values


## See also

Configuring I/O Modules for AADvance Applications on page 261
Analog Input Modules on page 269
Digital Output Modules on page 278
Analog Output Modules on page 286
Use these structure types when wiring channels for digital inputs:

- T9K_DI_COMPACT
- T9K_DI_FULL

To wire only the first element of a digital input channel, declare a variable using the BOOL data type. The element retrieves the input module state.

The T9K_DI_FULL structure type reports state values, including the faulted state, for digital input channels.

Wire channel variables so the controller receives the reported input values for the channels.

## To wire a channel variable to a digital input

1. From the Equipment View, double-click the required digital input module.

The Equipment property page displays with the module-specific tab and the Channels and Thresholds tabs.
2. From the Equipment property page, click the Channels tab. The channels display.

3. Locate the required channel, and in the Wire column click The Variable Selector displays.
4. In the Variable Selector, select the required variable and then click OK.

The digital input channel is wired.

## See also

T9K DI COMPACT on page 266
T9K DI FULL on page 266
Faulted State Value for Digital Inputs on page 267
Threshold Values for Digital Inputs on page 267
T9K_DI_COMPACT
Use the T9K_DI_COMPACT structure type to wire channels for digital inputs.

| Description | Identifier | Type | Remarks |
| :--- | :--- | :--- | :--- |
| Input State | <tagname>.DI | BOOL | • TRUE - Input voltage is above threshold T6 <br> $\bullet$ <br> • FALSE - Input voltage is below threshold T5 5 |
| Line Fault | <tagname>.LF | BOOL | - TRUE - Input voltage is above threshold T8; between T5 and T4; or below T1 <br> - FALSE - Input voltage is between thresholds T2 and T3; or between T6 and T7 |
| Discrepancy | <tagname>.DIS | BOOL | - TRUE - A discrepancy in voltage larger than $20 \%$ exists between the channels of a duplex or triplex <br> redundant configuration |

## See also

Channels for Digital Inputs on page 265
TGK_DI_FULL
Use the T9K_DI_FULL structure type to wire channels for digital inputs.

| Description | Identifier | Type | Remarks |
| :--- | :--- | :--- | :--- |
| Input State | <tagname>.DI | BOOL | • TRUE - Input voltage is above threshold T6 <br> • FALSE - Input voltage is below threshold T5 |


| Description | Identifier | Type | Remarks |
| :---: | :---: | :---: | :---: |
| Line Fault | <tagname>.LF | BOOL | - TRUE - Input voltage is above threshold T ; between T 5 and T 4 ; or below T 1 <br> - FALSE - Input voltage is between thresholds T 2 and T ; or between T 6 and T 7 |
| Discrepancy | <tagname>.DIS | BOOL | - TRUE - A discrepancy in voltage larger than $8 \%$ (of 24 V ) exists between the channels of a duplex or triplex redundant configuration |
| Channel Fault | <tagname>.CF | BOOL | - TRUE - Module diagnostics detect a fault in the channel electronics or firmware (state = 7) |
| Voltage | <tagname>.V | UINT | Reports the channel voltage in units of millivolts with an accuracy of $\pm 500 \mathrm{mV}$. |
| State | <tagname>.STA | USINT | Reports one of the following state value for the channel: <br> - 1 = Open circuit <br> - 2 = De-energized (off) <br> - 3 = Indeterminate <br> - 4 = Energized (on) <br> - 5 = Short circuit <br> - 6 = Over voltage <br> - 7 = Faulted |

Tip: $\quad$ T1-T8 refers to the Threshold Values for Digital Inputs.

## See also

## Channels for Digital Inputs on page 265

## Faulted State Value for Digital Inputs

A digital input channel is faulted when the channel is unable to report a voltage within a safety accuracy specification of $10 \%$ of the full scale of the 24 Vdc supply ( 2.4 V ). When a state is faulted, the state value is 7 . When a faulted state is reported, the following safe values are reported by the other channel variables:

- $\quad$ Input State $=$ FALSE
- Line Fault = TRUE
- Discrepancy = TRUE
- Channel Fault = TRUE
- Voltage $=0 \mathrm{mV}$


## See also

Channels for Digital Inputs on page 265
Threshold Values for Digital Inputs

The channel state and line fault status for the module are determined by comparing the channel input voltage with a set of threshold values. Use the default threshold values or define custom values. When defining threshold values for the input module, each channel inherits the threshold values. Also define individual threshold values for each digital input channel.

| Task | Procedure |
| :--- | :--- |
| Define the custom threshold values <br> for a digital input module | The channels of the digital input module inherit the custom threshold values. <br> 1. From the Equipment View, double-click the required digital input module. <br> The Equipment property page displays with the module-specific tab and the Channels and Thresholds tabs. <br> 2. From the Equipment property page, click the Thresholds tab. The default threshold values display. |
|  | 3. Select the Use Custom Thresholds option, then in the threshold fields, type the required values. |


| Task | Procedure |
| :--- | :--- |
| Define custom threshold values for <br> individual digital input channels | 1. From the Equipment View, expand the digital input module and then double-click the required channel. <br> The Equipment property page displays with the channel-specific tab and Thresholds tab. <br> 2. From the Equipment property page, click the Thresholds tab. The inherited threshold values _display. <br> 3. Select the Use Custom Thresholds option, then in the threshold fields, type the required values. |
| Restore the default threshold values <br> for a digital input module | • In the Equipment property page Thresholds tab, clear the Use Custom Thresholds option. |

An indeterminate region is defined between the closed and open regions to detect marginal faults in either the sensor or external wiring. To prevent chatter, the AADvance controller provides hysteresis on the thresholds for increasing and decreasing values. The reporting values update during each application cycle. The following displays the typical (default) threshold values (T\#), state values (STA), digital input statuses (DI), and line fault statuses for a digital input module:


## See also

Channels for Digital Inputs on page 265
Default Threshold Values for Digital Inputs on page 268

## Default Threshold Values <br> for Digital Inputs

These default threshold values for digital inputs are for a standard (online monitored) 24 Vdc digital input channel:


## See also

Threshold Values for Digital Inputs on page 267

## Analog Input Modules

For analog input modules, configure the following:

- Process Safety Time
- Status Variable
- Channels
- HART ${ }^{\circledR}$
- HART Pass-Through
- Threshold Values


## See also

Configuring I/O Modules for AADvance Applications on page 261
Digital Input Modules on page 265
Digital Output Modules on page 278
Analog Output Modules on page 286

Channels for Analog Inputs
Use these structure types when wiring channels for analog inputs:

- T9K_AI_COMPACT
- T9K_AI_FULL

To only wire the first element of an analog input channel, declare a variable using the REAL data type. The element retrieves a floating-point value representing 4 to 20 mA .

The structure types contain additional information about the input, like the discrepancy status. The T9K_AI_FULL structure type also reports state values, including the faulted state, for analog input channels.

Wire channel variables so the controller receives the reported input values for the channels.

Example

| Raw Count | Engineering Value (\%) | Process Value (mA) |
| :--- | :--- | :--- |
| 1,024 | 0 | 4 |
| 3,072 | 50 | 12 |
| 5,120 | 100 | 20 |

## To wire a channel variable to an analog input

Tip: When importing variables wired to analog $1 / 0$ channels, enter the required Raw and Engineering values for the analog $1 / 0$ channels.

1. From the Equipment View, expand the analog input module, and double-click the required channel.

The Equipment property page displays with the Channel-specific tab and the HART and Thresholds tabs.
2. From the Equipment property page, click the Channel-specific tab.
3. In the Channel Variable field, click

The Variable Selector displays.
4. In the Variable Selector, select the required variable and then click OK.

The analog input channel is wired.

| Channel 01 HART | Thresholds |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | Channel Variable channel_ai_01 |  |  |  | ... $X$ |
|  |  | Low | High |  |  |
|  | Raw | 1024 | 5120 |  |  |
|  | Engineering | 0 | 100 |  |  |
| Description | Wiring | Com | Logical | Physical | Locked |
| PV | channel_ai_0 |  |  |  |  |
| Raw Count | channel_ai_0 |  |  |  |  |
| Line Fault | channel_ai_0 |  |  |  |  |
| Discrepancy | channel_ai_0 |  |  |  |  |
| Channel Fault | channel_ai_0 |  |  |  |  |
| State | channel_ai_0 |  |  |  |  |

5. In the Raw fields, set the Low and High counts for the values received from the field elements.
6. In the Engineering fields, set the Low and High scaling factor in the range of $0 \%$ to $100 \%$.

## See also

Threshold Values for Analog Inputs on page 277
T9K AI COMPACT on page 271
T9K AI FULL on page 271
Faulted State Value for Analog Inputs on page 272
T9K_Al_COMPACT
Use the T9K_AI_COMPACT structure type to wire channels for analog inputs.

| Description | Identifier | Type | Remarks |
| :--- | :--- | :--- | :--- |
| Process Value (PV) | <tagname>.PV | REAL | A scaled, floating-point value representing the analog loop current. <br> The default scaling factor is 0 to $100 \%$, representing 4 to 20 mA. |
| Raw Count | <tagname>.CNT | INT | A count representing the current on the channel in units of $1 / 256 \mathrm{~mA}$. <br> $\bullet 0$ represents 0 mA <br> $\bullet 5,120$ represents 20 mA <br> Accuracy is within $\pm 13$ counts, equivalent to $\pm 0.05 \mathrm{~mA}$ |
| Discrepancy | <tagname>.DIS | BOOL | TRUE - A discrepancy in current larger than $2 \%$ exists between the channels of a duplex or triplex <br> redundant configuration. |

## See also

Channels for Analog Inputs on page 270
T9K_AL_FULL
Use the T9K_AI_FULL structure type to wire channels for analog inputs.

| Description | Identifier | Type | Remarks |
| :---: | :---: | :---: | :---: |
| Process Value (PV) | <tagname>.PV | REAL | A scaled, floating-point value representing the analog loop current. The default scaling factor is 0 to $100 \%$, representing 4 to 20 mA . |
| Raw Count | <tagname>.CNT | INT | A count representing the current on the channel in units of $1 / 256 \mathrm{~mA}$. <br> - 0 represents 0 mA <br> - 5,120 represents 20 mA <br> Accuracy is within $\pm 13$ counts, equivalent to $\pm 0.05 \mathrm{~mA}$ |
| Line Fault | <tagname>.LF | B00L | TRUE - The state value is $1,5,6$ or 7 FALSE - The state value is 2,3 or 4 |
| Discrepancy | <tagname>.DIS | B00L | TRUE - A discrepancy in current larger than $2 \%$ exists between the channels of a duplex or triplex redundant configuration. |
| Channel Fault | <tagname>.CF | BOOL | TRUE - Module diagnostics detect a fault in the channel electronics or firmware (state = 7) |
| State | <tagname>.STA | USINT | Reports one of the following state values for the channel: <br> - 1 = Open circuit <br> - 2 = Transmitter fault (low) <br> - 3 = Normal <br> - 4 = Transmitter fault (high) <br> - 5 = Short circuit <br> - 6 = Over range <br> - 7 = Faulted |

## See also

# Faulted State Value for Analog Inputs 

## Channels for Analog Inputs on page 270

An analog input channel is faulted when the channel is unable to report a count within a safety accuracy specification of $1 \%$ of the full scale measurement range of 5,120 ( 51 counts, 0.2 mA ). When a state is faulted, the state value is 7 . When a faulted state is reported, the following "safe" values are reported by the other channel variables:

- Process Value $=A$ calculated value based on a Count value of 0
- Line Fault = TRUE
- Discrepancy = TRUE
- Channel Fault = TRUE
- Count $=0$


## See also

Channels for Analog Inputs on page 270
Highway Addressable Remote Transducer (HART®) is an open protocol for process control instrumentation. HART combines digital signals with analog signals to supply control and status data for field devices. The AADvance controller supports the use of HART on each analog input and output channel. The AADvance controller also supports HART Pass-Through. The application can use HART data to monitor and respond to device conditions and to supply diagnostic information such as data comparison and error reporting. The HART protocol increases the level of safety system diagnostics.
If using HART in a safety system, follow these precautionary guidelines:

- Do not use HART variables as the primary initiator for a Safety Instrumented
Function (SIF). The HART protocol does not satisfy the required safety integrity
levels for Safety Instrumented Functions.
- Make sure to disable HART for field devices having an unlocked configuration.
This prevents the use of HART from changing a device configuration.
- Ensure the custom data for the device (the data given in response to HART
command 3) is used in accordance with the recommendations of the
manufacturer.

When enabling HART on an AADvance controller, ensure the HART field devices support the HART commands 0 (read unique ID) and 3 (read current and four dynamic variables). The commands are used for communication between the AADvance controller and the HART field devices. The analog input and output modules use the HART command 3 to collect data from the field devices as specified by Revision 5 of the HART specification.

To enable monitoring HART field device information, wire a HART variable to the required channels of analog input and output modules. HART variables must be declared using the following structure types:

- T9K_Al_HART
- T9K_Al_HART_FULL

For AADvance controllers, the T9K_AI_HART_FULL structure type is used for HART Pass-Through communication.

The HART structure types supply the following information:

- Loop current in milliamps (mA)
- Process measurement in engineering units
- Errors on HART communication seen by the field device
- Status of the field device
- Time since the most recent update, in milliseconds (ms)

Use the loop current variable for diagnostic checks in the application, comparing the value of the variable with the value on the 4 to 20 mA loop, and reacting if there is a discrepancy. Also monitor the status of the field device and use this to report diagnostic errors and manual configuration changes.

| Important: | The update rate for HART data from field devices is slower than the <br> update rate for the 4 to 20 mA analog signal itself. HART data can <br> take a maximum of 4 seconds to update, depending on the device <br> type and configuration. |
| :--- | :--- |

## To configure HART for monitoring an analog input field device

1. In the Dictionary, declare a HART variable having the structure type T9K_AI_HART.
2. From the Equipment View, expand the analog input module and then double-click the required channel.
The Equipment property page displays with the Channel-specific tab and the HART and Thresholds tabs.
3. In the Equipment property page, click the HART tab.
4. Select the Enable HART on this Channel property.

5. In the HART Variable field, click The Variable Selector displays.
6. In the Variable Selector, select the required variable and then click OK.

Tip: Enable HART on each channel to use the HART protocol.

## See also

Analog Input Modules on page 269
T9K AI HART on page 274
T9K AI HART FULL on page 274
HART Pass-Through for Analog Inputs on page 275
TGK_Al_HART
The T9K_AI_HART structure type is used when declaring HART variables.

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>. 1 | REAL | Current | Loop current in mA |
| <tagname>.V1 | REAL | First Variable | First loop current variable |
| <tagname>.U1 | BYTE | First Variable Units Code | First loop current variable units code |
| <tagname>.V2 | REAL | Second Variable | Second loop current variable |
| <tagname>.U2 | BYTE | Second Variable Units Code | Second loop current variable units code |
| <tagname>.V3 | REAL | Third Variable | Third loop current variable |
| <tagname>.U3 | BYTE | Third Variable Units Code | Third loop current variable units code |
| <tagname>.V4 | REAL | Fourth Variable | Fourth loop current variable |
| <tagname>. $\mathrm{U4}$ | BYTE | Fourth Variable Units Code | Fourth loop current variable units code |
| <tagname>.COMMS | BOOL | Communication Status | HART communication status <br> - TRUE - Communication OK <br> - FALSE - Communication stopped |
| <tagname>.DEVICE | BYTE | Device Status | Field device status: <br> - Bit 7 - Field device malfunction <br> - Bit 6 - Configuration changed <br> - Bit 5 - Cold start <br> - Bit 4 - More status available <br> - Bit 3 - Analog output current fixed <br> - Bit 2 - Analog output saturated <br> - Bit 1 - Second, third, or fourth variable out of limits <br> - Bit 0 - First variable out of limits |

## See also

HART for Analog Inputs on page 272
T9K_Al_HART_FULL
The T9K_AI_HART_FULL structure type is used for HART Pass-Through communication:

| Identifier | Type | Description | Remarks |
| :--- | :--- | :--- | :--- |
| <tagname>.I | REAL | Current | Loop current in mA |
| <tagname>.V1 | REAL | First Variable | First loop current variable |
| <tagname>.U1 | BYTE | First Variable Units Code | First loop current variable units code |
| <tagname>.V2 | REAL | Second Variable | Second loop current variable |
| <tagname>.U2 | BYTE | Second Variable Units Code | Second loop current variable units code |
| <tagname>.V3 | REAL | Third Variable | Third loop current variable |


| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.U3 | BYTE | Third Variable Units Code | Third loop current variable units code |
| <tagname>.V4 | REAL | Fourth Variable | Fourth loop current variable |
| <tagname>.U4 | BYTE | Fourth Variable Units Code | Fourth loop current variable units code |
| <tagname>.COMMS | BOOL | Communication Status | HART communication status <br> - TRUE - Communication OK <br> - FALSE - Communication stopped |
| <tagname>.DEVICE | BYTE | Device Status | Field device status: <br> - Bit 7 - Field device malfunction <br> - Bit 6 - Configuration changed <br> - Bit 5 - Cold start <br> - Bit 4 - More status available <br> - Bit 3 - Analog output current fixed <br> - Bit 2 - Analog output saturated <br> - Bit 1 - Second, third, or fourth variable out of limits <br> - Bit 0 - First variable out of limits |
| <tagname>.ELAPSED | DINT | Elapsed Time Since Update | Elapsed time (ms) since last non-Pass-Through communication. This parameter is reset to zero when data is received. |
| <tagname>.PASSTHROUGH | BOOL | Pass-Through Communication in Progress | HART Pass-Through communication status: <br> - TRUE: Communication OK <br> - FALSE: Communication stopped |

## See also

## HART Pass-Through for Analog Inputs

HART for Analog Inputs on page 272
The HART® Pass-Through feature enables using an external asset management system to manage HART compatible field devices connected to an AADvance controller.

HART Pass-Through uses the Device Type Manager (DTM) standard to enable using any asset management system having the generic Frame standard. Examples of compatible tools are the Fieldcare application by Endress+Hauser and the Factory Talk $^{\circledR}$ AssetCentre by Rockwell Automation ${ }^{\circledR}$.

To use HART Pass-Through, install the AADvance HART DTM software on the computer running the asset management system. Then enable or disable the HART Pass-Through capability of the controller. Declare HART variables to monitor HART Pass-Through data for analog input and output channels.


[^7]| Task | Procedure |
| :---: | :---: |
| Configure HART for Pass-Through communication monitoring | 1. In the Dictionary, declare a HART variable having the structure type TgK_Al_HART_FULL. <br> 2. From the Equipment View, expand the analog input module and then double-click the required channel. <br> The Equipment property page displays with the Channel-specific tab and the HART and Thresholds tabs. <br> 3. In the Equipment property page, click the HART tab. <br> 4. Select the Enable HART on this Channel property. <br> 5. In the HART Variable field, click <br> The Variable Selector displays. <br> 6. In the Variable Selector, select the required variable having the TTK_Al_HART_FULL type, and then click OK. |
| Install the AADvance HART DTM software | 1. Uninstall the existing version of the DTM installed on the workstation. <br> 2. Restart the system. <br> 3. Locate and install the file named AADvance DTM 1.xxx Setup.exe. <br> 4. Once the installation is complete, restart the system. |
| Enable HART Pass-Through in the controller | 5. In the Dictionary, create a variable having these properties: <br> - Name: HART_CONTROL <br> - Data type: BOOL <br> - Direction: VarOutput <br> The HART_CONTROL variable is used to enable and disable HART Pass-Through. <br> 6. In the Equipment View, double-click the controller requiring HART Pass-Through. <br> 7. The Equipment property page displays with the Processor, Status, Control, and RTC tabs. <br> 8. In the Equipment property page, click the Control tab. <br> 9. In the HART Pass-Through row, click <br> The Variable Selector displays. <br> 10. In the Variable Selector, select the HART_CONTROL variable, and then click OK. |

## HART Pass-Through has these features:

- Pass-Through support for HART standards 5, 6, and 7
- Dedicated Ethernet port for HART Pass-Through communication
- Compatible with the AADvance DTM supplied by Rockwell Automation ${ }^{\circledR}$
Tip: - Do not use HART Pass-Through on a safety application loop but is considered non-interfering on the SIL 3 loop.
- HART Pass-Through data is unmonitored.
- The HART feature in AADvance provides a controller capability to pass through requests and responses.

To use HART Pass-Through, perform the following:

- Declare a variable having the T9K_AI_HART_FULL structure type
- Declare a CIP Produce variable
- Declare a control processor variable for HART Pass-Through
- Enable HART on an analog input channel
- On an enabled channel, wire a HART variable
- Install the AADvance HART DTM software
- Enable HART Pass-Through at the controller level


HART Pass-Through has the following limitation for the maximum size of a HART message:

- For an analog input module, the maximum message size is $0 \times 5 \mathrm{~F}$ (plus between 5 and x14 bytes preamble) for the command to remote HART device and Ox5F for a response (including up to 14 bytes for the preamble).

A typical arrangement of a system using HART Pass-Through follows:


Set up the application to get status data for individual analog channels. These members of the T9K_AI_HART_FULL structure type supply data:

- <tagname>.ELAPSED (DINT), displays the time in milliseconds since the most recent valid non-Pass-Through communication. The value of this element resets to 0 (zero) when the application passes new HART data on the channel.
- <tagname>.PASSTHROUGH (BOOL), displays when the channel is carrying HART Pass-Through data.

The application can use these members to choose when to permit HART PassThrough communications.

## See also

HART for Analog Inputs on page 272

Threshold Values for Analog Inputs

The channel state and line fault status for the module are determined by comparing the channel input current with a set of threshold values. Use the default threshold values or define custom values. When defining threshold values for the input module, each channel inherits the threshold values. Also define individual threshold values for each analog input channel.

| Important: | When the system is operational, threshold values can only be <br> changed using an online update. |
| :--- | :--- | changed using an online update.


| Task | Procedure |
| :--- | :--- |
| Define the custom threshold values <br> for an analog input module | The channels of the analog input module inherit the custom threshold values. <br> 1. From the Equipment View, double-click the required analog input module. <br> The Equipment property page displays with the Module-specific tab and the Channels, HART®, and Thresholds tabs. <br> 2. From the Equipment property page, click the Thresholds tab. The default threshold values display. <br> 3. Select the Use Custom Thresholds option, then in the threshold fields, type the required values. |
| Define custom threshold values for <br> individual analog input channels | 1. From the Equipment View, expand the digital input module and then double-click the required channel. <br> The Equipment property page displays with the Channel-specific, HART, and Thresholds tabs. <br> 2. From the Equipment property page, click the Thresholds tab. The inherited threshold values display. <br> 3. Select the Use Custom Thresholds option, then in the threshold fields, type the required values. |
| Restore the default threshold values <br> for an analog input module | - In the Equipment property page Thresholds tab, clear the Use Custom Thresholds option. |

To prevent chatter, the AADvance controller provides hysteresis on the thresholds for increasing and decreasing values. The reporting values update during each application cycle. The following displays the typical (default)
threshold values (T\#), state values (STA), and line fault statuses for an analog input module:


## See also

Analog Input Modules on page 269

## Default Threshold Values for Analog Inputs

The default threshold values for analog inputs are for a standard (online monitored) 24 Vdc analog input channel. The values agree with the lower limit and higher limits for an analog signal detailed in the NAMUR NE43 standard. The following are the default values:


## See also

## Threshold Values for Analog Inputs on page 277

For digital output modules, configure the following:

- Process Safety Time
- Status Variable
- Channels
- Advanced Settings
- Variables


## See also

Configuring I/O Modules for AADvance Applications on page 261
Digital Input Modules on page 265
Analog Input Modules on page 269
Analog Output Modules on page 286

## Channels for Digital Outputs

Use these structure types when wiring channels for digital outputs:

- T9K_DO_COMPACT
- T9K_DO_FULL

The structure types provide data about the output like the line fault and discrepancy statuses.

To wire only the first element of a digital output channel, declare a variable using the BOOL data type.

Also set the Advanced settings for individual digital output channels.

| Important: | The controller writes to the digital outputs once during each <br> application cycle and the digital output variables are updated once <br> during each application cycle. |
| :--- | :--- |

## To wire a channel variable to a digital output

1. From the Equipment View, double-click the required digital output module.
The Equipment property page displays with the module-specific tab and the Channels and Variables tabs.
2. From the Equipment property page, click the Channels tab. The channels display.

3. Locate the required channel, and in the Wire column click The Variable Selector displays.
4. In the Variable Selector, select the required variable and then click OK. The digital output channel is wired.

## See also

T9K DO COMPACT on page 280
T9K DO FULL on page 281
Faulted State Value for Digital Outputs on page 281
Protection for Digital Outputs on page 281
Advanced Settings for Digital Output Channels on page 282
T9K_DO_COMPACT
Use the T9K_DO_COMPACT structure type to wire channels for digital outputs.

| Identifier | Type | Description | Remarks |
| :--- | :--- | :--- | :--- |
| <tagname>.DOP | BOOL | Command State | The output demand to pass to the output channel <br> - Set to TRUE to energize <br> - Set to FALSE to de-energize |
| <tagname>.LF | BOOL | Line fault | TRUE - The field supply is missing, the load is disconnected, or a short circuit is detected. |
| <tagname>.DIS | BOOL | Discrepancy | TRUE - A discrepancy in current larger than $1 \%$ exists between the channels of two modules in a <br> redundant configuration. Discrepancy can only be reported TRUE when two modules are active in a <br> group. |

## See also

T9K DO FULL on page 281

## T9K_DO_FULL

Use the T9K_DO_FULL structure type to wire channels for digital outputs.

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.DOP | BOOL | Command State | The output demand to pass to the output channel <br> - Set to TRUE to energize <br> - Set to FALSE to de-energize |
| <tagname>.LF | BOOL | Line fault | TRUE - The field supply is missing, the load is disconnected, or a short circuit is detected. |
| <tagname>.DIS | BOOL | Discrepancy | TRUE - A discrepancy in current larger than $1 \%$ exists between the channels of two modules in a redundant configuration. Discrepancy can only be reported TRUE when two modules are active in a group. |
| <tagname>.CF | BOOL | Channel Fault | TRUE - Module diagnostics identify a fault in the channel electronics or firmware (state = 7) |
| <tagname>.V | UINT | Voltage | Reports the channel voltage at the output terminals, in units of millivolts within an accuracy of $\pm$ 500 mV . The voltage structure member is unable to report values below 0 mV . |
| <tagname>.I | INT | Current | Reports the current for the channel in milliamps within an accuracy of $\pm 2 \mathrm{~mA}$ and $\pm 10 \%$ of measurement. |
| <tagname>.STA | USINT | State | Reports a state value for the channel: <br> - 1 = No vfield, the field supply voltage is at or below 18 V . The reported voltage (<tagname>.V) is 0 mV . <br> - 2 = De-energized (off), the command state is FALSE and the channel is de-energized. <br> - 3 = No load, the controller is unable to identify a load connected to the channel field wiring or the load is below the required minimum channel load of 10 mA when the command state is TRUE. <br> - 4 = Energized (on), the command state is TRUE and the channel is energized. <br> - 5 = Short circuit, the controller has detected a short circuit condition, irrespective of the channel drive state. <br> - 6 = Field fault, an external source is driving the channel to an energized state or a voltage is larger than 18 Vdc , irrespective of the channel drive state. <br> - 7 = Faulted |

## See also

T9K DO COMPACT on page 280

## Faulted State Value for Digital Outputs

A digital output channel is faulted when normal operations or diagnostic tests find a known fault condition. When a channel is faulted, the state reports the value 7 , and the other variables report the following 'safe' values:

- Line Fault = TRUE
- Discrepancy = TRUE
- Channel Fault = TRUE
- Voltage $=0 \mathrm{mV}$
- Current $=0 \mathrm{~mA}$


## See also

## Protection for Digital Outputs on page 281

## Advanced Settings for Digital Output Channels on page 282

The AADvance controller has three mechanisms to protect digital output channels:

- Inrush current protection
- Short circuit protection for energized channels
- Short circuit protection for de-energized channels

The controller tolerates inrush currents so the digital outputs can energize capacitive loads without causing the controller to report a short circuit. The following displays the characteristics of the maximum load currents the controller tolerates when a digital output is energized. If the load current enters the region above the curve on the graph, the controller applies inrush current protection.


After allowing for inrush, the controller engages short circuit protection for an energized channel when the loop current reaches 2 A .

- Short circuit detection on an energized channel is immediate and the channel is de-energized. The controller reports the condition until the short circuit is cleared.
- When the short circuit is removed, the channel re-energizes. The short circuit report is then cleared by pressing the fault reset button on the 9110 processor module or by setting the command state to FALSE.

The controller verifies de-energized digital output channels for possible short circuits. Periodically, the controller partially turns on each de-energized output in turn and measures the loop current. If the loop current shows a loop resistance of less than approximately $10 \Omega$, the controller reports a short circuit.

## See also

Faulted State Value for Digital Outputs on page 281
Advanced Settings for Digital Output Channels on page 282

## Advanced Settings for Digital Output Channels

Configure these advanced settings for individual digital output channels:

- Shutdown State
- Disable Line Test


## Shutdown State

The Shutdown State defines how the output behaves when the following occurs to the parent 9451 digital output module:

- The module suffers a loss of communications with the processor
- A failure in the module causes the module to go into shutdown mode


## Disable Line Test

Enable or disable the line test for AADvance controllers. The 9451 digital output module performs a test (Disable Line Test) for a no load condition on
each output. A no load condition occurs when the controller is unable to locate a load connected to the field wiring or the load current is below 20 mA when the output command state is TRUE. Disable the line test to connect a low load to an output or if the output is unused and it preferred not to fit a dummy load.

When the line test is enabled, the module reports a no load condition by setting the state variable (<tagname>.STA) to the value 3 and by setting the channel LED to amber. After disabling the line test, and assuming there are no other faults present, the state variable indicates the value 2 or 4 (depending on the command state value) and the channel LED is off or green instead of amber.

Important: If line test is disabled there are other scenarios, like no field voltage, that set the channel LED to amber.

## To configure the shutdown state and disable line test

1. From the Equipment View, expand the digital output module and then double-click the required channel.
The Equipment property page displays with the Channel-specific and Advanced tabs.
2. In the Equipment property page, click the Advanced tab.
3. Select the Use Advanced Settings property.

4. Set the Shutdown State to one of the following options:

- Off - Disables the output when the module is in the shutdown state
- Hold Last State - Forces the output to stay in the most recent state during a module shutdown.
- Default - Not used by the application

Important: To maintain the safety rating of the system, when using Hold Last State for a safety-related output, restore the communications in less than the MTTR or use other compensating measures during a module failure.
5. Set the Disable Line Test to Yes or No.

## See also

Faulted State Value for Digital Outputs on page 281
Protection for Digital Outputs on page 281

## Variables for Digital Outputs

The 9451 digital output module provides these status variables available to the application:

- Status Booleans
- Group Field Power Health
- Field Power Status Integers
- Group Field Power Current
- A Module Field Power Voltage 1
- A Module Field Power Voltage 2
- B Module Field Power Voltage 1
- B Module Field Power Voltage 2


| Task | Procedure |
| :---: | :---: |
| Wire a status variable to a digital output module | 1. In the Equipment View, double-click the required digital output module. <br> The Equipment property page displays with the Module-specific, Channels, and Variables tabs. <br> 2. In the Equipment property page, click the Variables tab. <br> 3. From the Variables tab, locate the required status variable, and in the Wire column click The Variable Selector displays. <br> 4. From the Variable Selector, select the required variable and click OK. |
| Unwire a processor variable from a digital output module | 1. From the Equipment View, double-click the required digital output module. <br> 2. In the Equipment property page, click the Variables tab. <br> 3. From the Variables tab, locate the wired variable, and in the Unwire column click |

Group Field Power Health
Provides a top level indication of the health of field power supplies to active digital output modules.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | Boolean |
| Values | - TRUE - All field power supplies for all active digital output <br> modules in the group are in the range of 18 V to 32 Vdc. <br> - FALSE - One or more field power supplies to an active <br> module is less than 18 Vdc or more than $32 \mathrm{Vdc}$. |
| Wiring | \%IX115.9.0 |

Important: The controller incorporates a 0.5 V hysteresis on these thresholds to prevent chatter. The controller declares a fault when a supply falls below 18 V but does not clear the fault until the supply rises to 18.5 V. Similarly the controller declares a fault when a supply exceeds 32 V but does not clear the fault until the supply falls below 31.5 V .

Group Field Power Current
Reports the total current drawn from the field power supply by the active digital output modules in a group. Accuracy is $\pm 10 \%$.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | DINT |
| Values | 0 to $8,000 \mathrm{~mA}$ or larger (limited by the capacity of the DINT <br> variable) |

## A Module Field Power Voltage 1

Reports the voltage from the field power supply for the specified module and field power input. Accuracy is $\pm 500 \mathrm{mV}$.

| Direction | Input to the application from the controller. |
| :--- | :--- |
| Type | DINT |
| Values | 0 to $48,000 \mathrm{mV}$ or larger (limited by the capacity of the DINT <br> variable) |

## See also

A Module Field Power Voltage 2 on page 285
B Module Field Power Voltage 1 on page 285
B Module Field Power Voltage 2 on page 286
Reports the voltage from the field power supply for the specified module and field power input. Accuracy is $\pm 500 \mathrm{mV}$.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | DINT |
| Values | 0 to $48,000 \mathrm{mV}$ or larger (limited by the capacity of the DINT <br> variable) |

## See also

A Module Field Power Voltage 1 on page 285
B Module Field Power Voltage 1 on page 285
B Module Field Power Voltage 2 on page 286
Reports the voltage from the field power supply for the specified module and field power input. Accuracy is $\pm 500 \mathrm{mV}$.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | DINT |


| Values | 0 to $48,000 \mathrm{mV}$ or larger (limited by the capacity of the DINT <br> variable) |
| :--- | :--- |

## See also

A Module Field Power Voltage 1 on page 285
B Module Field Power Voltage 1 on page 285
B Module Field Power Voltage 2 on page 286
Reports the voltage from the field power supply for the specified module and field power input. Accuracy is $\pm 500 \mathrm{mV}$.

| Direction | Input to the application from the controller |
| :--- | :--- |
| Type | DINT |
| Values | 0 to $48,000 \mathrm{mV}$ or larger (limited by the capacity of the DINT <br> variable) |

## See also

A Module Field Power Voltage 1 on page 285
A Module Field Power Voltage 2 on page 285
B Module Field Power Voltage 1 on page 285

## Analog Output Modules

Channels for Analog Outputs

For analog output modules, configure the following:

- Process Safety Time
- Status Variable
- Channels
- Advanced Settings
- HART ${ }^{\oplus}$
- HART Pass-Through


## See also

Configuring I/O Modules for AADvance Applications on page 261
Digital Input Modules on page 265
Analog Input Modules on page 269
Digital Output Modules on page 278
Use these structure types when wiring channels for analog outputs:

- T9K_AO_COMPACT
- T9K_AO_FULL

To wire only the first element of an analog output channel, declare a variable using the REAL data type. This variable retrieves the command state.

The structure types provide data about the output, including the line fault and discrepancy statuses.

The controller writes to the analog outputs once during each application cycle. The analog output variables are also updated once during each application cycle.

## Example

| Raw Count | Engineering Value (\%) | Process Value (mA) |
| :--- | :--- | :--- |
| 1,024 | 0 | 4 |
| 3,072 | 50 | 12 |
| 5,120 | 100 | 20 |

## To wire a channel variable to an analog output

1. From the Equipment View, expand the analog output module, and double-click the required channel.

The Equipment property page displays with the Channel-specific tab and the HART and Advanced tabs.
2. From the Equipment property page, click the Channel-specific tab.
3. In the Channel Variable field, click

The Variable Selector displays.
4. In the Variable Selector, select the required variable and then click OK.

The analog output channel is wired.

5. In the Raw fields, set the Low and High counts for the values received from the field elements.
6. In the Engineering fields, set the Low and High scaling factor in the range of $0 \%$ to $100 \%$.

## See also

T9K AO FULL on page 288
Faulted State Value for Analog Outputs on page 289
Advanced Settings for Analog Output Channels on page 289

## T9K_AO_COMPACT

 Use the T9K_AO_COMPACT structure type to wire channels for analog outputs.| Description | Identifier | Type | Remarks |
| :--- | :--- | :--- | :--- |
| Command Value | <tagname>.CV | REAL | The demanded current. A scaled, floating-point value showing the analog loop current. <br> The default scaling factor is 0 to $100 \%$, representing 4 to 20 mA. |
| Line Fault | <tagname>.LF | BOOL | TRUE - One of the following: <br> - The field supply is missing <br> - The load is disconnected <br> - Unable to meet the commanded output current <br> - The wiring polarity is reversed <br> - Channel fault (states $1,3,5,6,7)$. |
| Discrepancy | <tagname>.DIS | BOOL | TRUE - The measured current and commanded current differ by more than the fail-safe guard band |

## See also

Channels for Analog Outputs on page 286
T9K_AO_FULL
Use the T9K_AO_FULL structure type to wire channels for analog outputs.

| Description | Identifier | Type | Remarks |
| :---: | :---: | :---: | :---: |
| Command Value | <tagname>.CV | REAL | The demanded current. A scaled, floating-point value showing the analog loop current. The default scaling factor is 0 to $100 \%$, representing 4 to 20 mA . |
| Line Fault | <tagname>.LF | BOOL | TRUE - One of the following: <br> - The field supply is missing <br> - The load is disconnected <br> - Unable to meet the commanded output current <br> - The wiring polarity is reversed <br> - Channel fault (states $1,3,5,6,7$ ) |
| Discrepancy | <tagname>.DIS | B00L | TRUE - The measured current and commanded current differ by more than the fail-safe guard band. |
| Channel Fault | <tagname>.CF | BOOL | TRUE - Module diagnostics detect a fault in the channel electronics or firmware (state = 7) |
| Voltage | <tagname>.V | INT | Reports the channel voltage at the output terminals, in units of millivolts and within an accuracy of 500 mV . |
| Raw Count | <tagname>.CNT | INT | Reports the current for the channel in raw units scaled 256 per mA, from 0 to 24 mA (0 to 6,144). |
| State | <tagname>.STA | USINT | Reports a state value for the channel: <br> - 1 = No vfield, the field supply voltage is at or below 18 Vdc and the commanded current is less than 0.4 mA for the channel. <br> Important: When the state value is 1 , the field voltage (<tagname>.V) is 0 mV . <br> - 2 = De-energized (off), the raw count value is less than 102 ( 0.4 mA ). <br> - 3 = No load or open circuit, the controller is unable to identify a load connected to the channel field wiring, or is unable to detect the loop voltage. A no load occurs when the commanded current is more than 0.4 mA , the raw count value is less than $51(0.2 \mathrm{~mA})$, and the measured voltage is less than 1000 mV . <br> - 4 = Energized (on), the raw count value is 102 or greater ( $>=0.4 \mathrm{~mA}$ ). <br> - 5 = Compliance fault, Insufficient loop voltage is available to sustain the commanded output current to within the safety accuracy specification (for example, $1 \%$ full scale or 0.2 mA ) and the commanded current is more than 102 counts ( 0.4 mA ). <br> - $6=$ Reverse polarity, the measured voltage is less than $-1000 \mathrm{mV}(<-1 \mathrm{Vdc})$. <br> - 7 = Faulted |

## See also

## Faulted State Value for Analog Outputs

## Channels for Analog Outputs on page 286

An analog output channel is faulted when normal operation or diagnostic tests find a known fault condition. When a channel is faulted, the state reports the value 7 , and the other variables report the following 'safe' values:

- Line Fault = TRUE
- Discrepancy = TRUE
- Channel Fault = TRUE
- Voltage $=0 \mathrm{mV}$
- Raw Count $=0 \mathrm{~mA}$


## See also

Channels for Analog Outputs on page 286
Configure the advanced settings for individual analog output channels:

- Shutdown State
- Custom Shutdown Value
- Fail-Safe Guard Band

Configure the advanced shutdown state for an analog output channel. The shutdown state defines how the output behaves when its parent 9481 or 9482 analog output module goes into shutdown mode due to a loss of communications with the processor or a failure in the module.

## To configure the advanced settings for analog output channels

1. From the Equipment View, expand the analog output module and then double-click the required channel.

The Equipment property page displays with the Channel-specific, HART, and Advanced tabs.
2. In the Equipment property page, click the Advanced tab.
3. Select the Use Advanced Settings property.

4. Set the Shutdown State to one of the following options:

- Off - Disables (de-energizes) the output when the module is in the shutdown state
- Hold Last State - Forces the output to stay in the most recent current during a module shutdown.
- Custom - During a module shutdown, forces the output to go to the value set in the Custom Shutdown Value property.

| Important: $\quad$ | To maintain the safety rating of the system, when using Hold Last |
| :--- | :--- |
|  | State for a safety-related output, restore the communications in |
|  | less than the MTTR or use other compensating measures during a |
|  | module failure. |

5. When setting the Shutdown State to Custom, set the Custom Shutdown Value to a value of current using the same scaling as the command value (<tagname>.CV).

- The default scaling factor is 0 to $100 \%$, representing 4 to 20 mA
- The default value is $\mathbf{- 2 5 . 0 0}$

6. Set the Fail-Safe Guard Band. This property defines the threshold for the discrepancy alarm between the command value (<tagname>.CV) and Count (<tagname>.CNT). The discrepancy alarm is reported by the structure member <tagname>.DIS. The Fail-Safe Guard Band can be set to the following values:

- 0 to $100 \%$ of full scale $20 \mathrm{~mA}(1 \%=0.2 \mathrm{~mA})$
- The default value is $1 \%$


# HART for Analog Outputs 

## See also

Channels for Analog Outputs on page 286
Highway Addressable Remote Transducer (HART ${ }^{\circledR}$ ) is an open protocol for process control instrumentation. HART combines digital signals with analog signals to supply control and status data for field devices. The AADvance controller supports the use of HART on each analog input and output channel. The AADvance controller also supports HART Pass-Through. The application can use HART data to monitor and respond to device conditions and to supply diagnostic information such as data comparison and error reporting. The HART protocol increases the level of safety system diagnostics.
If using HART in a safety system, follow these precautionary guidelines:

- Do not use HART variables as the primary initiator for a Safety Instrumented
Function (SIF). The HART protocol does not satisfy the required safety integrity
levels for Safety Instrumented Functions.
- Make sure to disable HART for field devices having an unlocked configuration.
This prevents the use of HART from changing a device configuration.
- Ensure the custom data for the device (the data given in response to HART
command 3) is used in accordance with the recommendations of the
manufacturer.

When enabling HART on an AADvance controller, ensure the HART field devices support the HART commands 0 (read unique ID) and 3 (read current and four dynamic variables). The commands are used for communication between the AADvance controller and the HART field devices. The analog input and output modules use the HART command 3 to collect data from the field devices as specified by Revision 5 of the HART specification.

To enable monitoring HART field device information, wire a HART variable to the required channels of analog input and output modules. HART variables must be declared using the following structure types:

- T9K_Al_HART
- T9K_Al_HART_FULL

The T9K_Al_HART_FULL structure type is used for HART Pass-Through communication.

The HART structure types supply the following information:

- Loop current in milliamps (mA)
- Process measurement in engineering units
- Errors on HART communication seen by the field device
- Status of the field device
- Time since the most recent update, in milliseconds (ms)

Use the loop current variable for diagnostic checks in the application, comparing the value of the variable with the value on the 4 to 20 mA loop, and reacting if there is a discrepancy. Also monitor the status of the field device and use this to report diagnostic errors and manual configuration changes.

Important: The update rate for HART data from field devices is slower than the update rate for the 4 to 20 mA analog signal itself. HART data can take a maximum of 4 seconds to update, depending on the device type and configuration.

## To configure HART for monitoring an analog output field device

1. In the Dictionary, declare a HART variable having the structure type T9K_AI_HART.
2. From the Equipment View, expand the analog output module and then double-click the required channel.

The Equipment property page displays with the Channel-specific tab and the HART and Advanced tabs.
3. In the Equipment property page, click the HART tab.
4. Select the Enable HART on this Channel property.

5. In the HART Variable field, click -...

The Variable Selector displays.
6. In the Variable Selector, select the required variable and then click OK.

## See also

Analog Output Modules on page 286
T9K AI HART on page 274
T9K AI HART FULL on page 274
HART Pass-Through for Analog Outputs on page 293
TgK_Al_HART
The T9K_AI_HART structure type is used when declaring HART variables.

| Identifier | Type | Description | Remarks |
| :--- | :--- | :--- | :--- |
| <tagname>.l | REAL | Current | Loop current in mA |
| <tagname>.V1 | REAL | First Variable | First loop current variable |
| <tagname>.U1 | BYTE | First Variable Units Code | First loop current variable units code |
| <tagname>.V2 | REAL | Second Variable | Second loop current variable |
| <tagname>.U2 | BYTE | Second Variable Units Code | Second loop current variable units code |
| <tagname>.V3 | REAL | Third Variable | Third loop current variable |
| <tagname>.U3 | BYTE | Third Variable Units Code | Third loop current variable units code |
| <tagname>.V4 | REAL | Fourth Variable | Fourth loop current variable |
| <tagname>.U4 | BYTE | Fourth Variable Units Code | Fourth loop current variable units code |
| <tagname>.COMMS | BOOL | Communication Status | HART communication status <br> $\bullet$ |

# Chapter 2 

Working with AADvance Applications

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.DEVICE | BYTE | Device Status | Field device status: <br> - Bit 7 - Field device malfunction <br> - Bit 6 - Configuration changed <br> - Bit 5 - Cold start <br> - Bit 4 - More status available <br> - Bit 3 - Analog output current fixed <br> - Bit 2 - Analog output saturated <br> - Bit 1 - Second, third, or fourth variable out of limits <br> - Bit 0 - First variable out of limits |

## See also

HART for Analog Inputs on page 272
T9K_Al_HART_FULL
The T9K_Al_HART_FULL structure type is used for HART Pass-Through communication:

| Identifier | Type | Description | Remarks |
| :---: | :---: | :---: | :---: |
| <tagname>.I | REAL | Current | Loop current in mA |
| <tagname>.V1 | REAL | First Variable | First loop current variable |
| <tagname>.U1 | BYTE | First Variable Units Code | First loop current variable units code |
| <tagname>.V2 | REAL | Second Variable | Second loop current variable |
| <tagname>.U2 | BYTE | Second Variable Units Code | Second loop current variable units code |
| <tagname>.V3 | REAL | Third Variable | Third loop current variable |
| <tagname>.U3 | BYTE | Third Variable Units Code | Third loop current variable units code |
| <tagname>.V4 | REAL | Fourth Variable | Fourth loop current variable |
| <tagname>.U4 | BYTE | Fourth Variable Units Code | Fourth loop current variable units code |
| <tagname>.COMMS | B00L | Communication Status | HART communication status <br> - TRUE - Communication OK <br> - FALSE - Communication stopped |
| <tagname>.DEVICE | BYTE | Device Status | Field device status: <br> - Bit 7 - Field device malfunction <br> - Bit 6 - Configuration changed <br> - Bit 5 - Cold start <br> - Bit 4 - More status available <br> - Bit 3 - Analog output current fixed <br> - Bit 2 - Analog output saturated <br> - Bit 1 - Second, third, or fourth variable out of limits <br> - Bit 0 - First variable out of limits |
| <tagname>.ELAPSED | DINT | Elapsed Time Since Update | Elapsed time (ms) since last non-Pass-Through communication. This parameter is reset to zero when data is received. |
| <tagname>.PASSTHROUGH | B00L | Pass-Through Communication in Progress | HART Pass-Through communication status: <br> - TRUE: Communication OK <br> - FALSE: Communication stopped |

## See also

HART for Analog Inputs on page 272
HART Pass-Through for
Analog Outputs
The HART ${ }^{\circledR}$ Pass-Through feature enables using an external asset management system to manage HART compatible field devices connected to an AADvance controller.

HART Pass-Through uses the Device Type Manager (DTM) standard to enable using any asset management system having the generic 'Frame' standard. Examples of compatible tools are the Fieldcare application by Endress+Hauser and the FactoryTalk AssetCentre by Rockwell Automation.

To use HART Pass-Through, install the AADvance HART DTM software on the computer running the asset management system. Then enable or disable the HART Pass-Through capability of the controller. Declare HART variables to monitor HART Pass-Through data for analog input and output channels.
When using HART Pass-Through in a safety system, follow these precautionary
guidelines:

- Ensure HART Pass-Through is only enabled under control of the application.
- Ensure HART Pass-Through is enabled only when necessary.
- Configure the application to start an alarm if HART Pass-Through is enabled on
any safety-critical channel of any module.

| Task | Procedure |
| :---: | :---: |
| Configure HART for Pass-Through communication monitoring | Wire a variable having the T9K_Al_HART_FULL structure type to the required channels. Enable HART on each channel to use HART Pass-Through. <br> 1. In the Dictionary, declare a HART variable having the structure type TgK_Al_HART_FULL. <br> 2. From the Equipment View, expand the analog output module and then double-click the required channel. <br> The Equipment property page displays with the Channel-specific tab and the HART and Advanced tabs. <br> 3. In the Equipment property page, click the HART tab. <br> 4. Select the Enable HART on this Channel property. <br> 5. In the HART Variable field, click <br> The Variable Selector displays. <br> 6. In the Variable Selector, select the required variable having the T9K_Al_HART_FULL type, and then click OK. |
| Install the AADvance HART DTM software | Install the software on the Windows computer running the asset management system. The AADvance HART DTM is accessible from the Rockwell Automation ${ }^{\ominus}$ Download Center. <br> 1. Uninstall the existing version of the DTM installed on the Workstation. <br> 2. Restart the system. <br> 3. Locate and install the file named AADvance DTM 1.xxx Setup.exe. <br> 4. Once the installation is complete, restart the system. |
| Enable HART Pass-Through in the controller | To use HART Pass-Through, define at least one CIP Produce variable. To activate the CIP stack, have CIP produce and consume variables. <br> 1. In the Dictionary, create a variable having these properties: <br> - Name: HART_CONTROL <br> - Data type: BOOL <br> - Direction: VarOutput <br> The HART_CONTROL variable is used to enable and disable HART Pass-Through. <br> 2. In the Equipment View, double-click the controller requiring HART Pass-Through. <br> The Equipment property page displays with the Processor, Status, Control, and RTC tabs. <br> 3. In the Equipment property page, click the Control tab. <br> 4. In the HART Pass-Through row, click <br> The Variable Selector displays. <br> 5. In the Variable Selector, select the HART_CONTROL variable, and then click OK. |

## HART Pass-Through has these features:

- Pass-Through support for HART standards 5, 6, and 7
- Dedicated Ethernet port for HART Pass-Through communication
- Compatible with the AADvance DTM supplied by Rockwell Automation ${ }^{\circledR}$

Tip: - Do not use HART Pass-Through on a safety application loop but is considered non-interfering on the SIL 3 loop.

- HART Pass-Through data is unmonitored.
- The HART feature in AADvance provides a controller capability to pass through requests and responses.

To use HART Pass-Through, perform the following:

- Declare a variable having the T9K_AI_HART_FULL structure type
- Declare a CIP Produce variable
- Declare a control processor variable for HART Pass-Through
- Enable HART on an analog output channel
- On an enabled channel, wire a HART variable
- Install the AADvance HART DTM software
- Enable HART Pass-Through at the controller level


HART Pass-Through has this limitation for the maximum size of a HART message:

- For an analog output module, the limits are 0x31 bytes for the command (plus between 5 and $\times 14$ bytes preamble) and $0 \times 63$ bytes for a response (including up to 0x14 bytes for the preamble).

A typical arrangement of a system using HART Pass-Through follows:


Set up the application to get status data for individual analog channels. These members of the T9K_Al_HART_FULL structure type supply data:

- <tagname>.ELAPSED (DINT), displays the time in milliseconds since the most recent valid non-Pass-Through communication. The value of this element resets to 0 (zero) when the application passes new HART data on the channel.
- <tagname>.PASSTHROUGH (BOOL), displays when the channel is carrying HART Pass-Through data.

The application can use these members to choose when to permit HART PassThrough communications.

## See also

HART for Analog Outputs on page 290

## AADvance Discover

Use AADvance Discover to set the IP address for the physical controller. Configure the resource number and IP address when assembling a new AADvance controller or installing a new 9100 processor base unit. Also configure Ethernet forwarding if required. The AADvance Discover tool uses a discovery and configuration protocol (DCP) to scan the broadcast domain for AADvance controllers. The tool then displays a list of AADvance controllers and their statuses. Refresh the list of controllers.

## Tip：Before using AADvance Discover，disable connections to networks other than the AADvance ${ }^{\oplus}$ Control System network．

| －A AADvance Discover－Controller Discovery and Configuration |  |  |  | $\square$－$\square^{\text {a }}$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Controller ID | Configuration Name | Type | Res Num | Status | － |
| 罝 $00-\mathrm{AO}-\mathrm{EC}-44-25-\mathrm{BC}$ | OPTIMADDINFBLDGENMO\DEVICE1 | Series 9000 | 15 | Configurable |  |
| 且00－A0－EC－44－25－C2 | BINDINGSTEST\C3 | Series 9000 | 8 | Locked |  |
| 罝 $00-\mathrm{AO}-\mathrm{EC}-44-25-\mathrm{C8}$ | DEMOKIT＿FROM＿Z＿V2\CONTROLLER3 | Series 9000 | 3 | Locked |  |
| 罝 $00-\mathrm{AO}-\mathrm{EC}-44-25-\mathrm{F} 2$ | SXLME $\backslash$ SIS＿H5 | Series 9000 | 6 | Locked |  |
| 䧃 $00-\mathrm{AO}-\mathrm{EC}-44-25-\mathrm{Fs}$ | PROJECT3\CONTROLLER＿1 | Series 9000 | 9 | Locked |  |
| 䧃 $00-\mathrm{AO}-\mathrm{EC}-44-26-0 \mathrm{~A}$ |  | Series 9000 | 2 | Configurable |  |
| 罝00－A0－EC－44－26－16 | TESTOLC $\backslash$ CONTROLLER1 | Series 9000 | 1 | Locked |  |
| 䧃 $00-\mathrm{AO}-\mathrm{EC}-44-26-1 \mathrm{C}$ | BINDINGSTEST\C1 | Series 9000 | 2 | Locked |  |
| 罝 00－A0－EC－44－26－34 | CIP＿PROD\CONTROLLER1 | Series 9000 | 1 | Locked |  |
| 圌 $00-\mathrm{AO}-\mathrm{EC}-44-26-4 \mathrm{C}$ | SXLME $\backslash$ SIS＿D1 | Series 9000 | 5 | Locked |  |
| 䧃 $00-\mathrm{AO}-\mathrm{EC}-44-26-64$ | PROJECT7\CONTROLLER1 | Series 9000 | 3 | Locked | － |
| － | II＇ | － | － | $\square$ |  |
| Ready |  |  |  | Refres |  |

From AADvance Discover，access the Controller Configuration dialog box， enabling the configuration of the Resource Number，IP Address，Subnet Mask，Gateway，and Gateway Port values to be stored in a physical controller．For Locked Controllers，this information is view only．Also use the Controller Configuration dialog box to save and load controller configurations．


## See also

## AADvance Discover <br> Features

Tasks to perform with AADvance Discover are:

| Task | Procedure |
| :---: | :---: |
| Configure the IP Address, Subnet Mask, Gateway, and Gateway Port in a controller | 1. Note the MAC address (Controller ID) labeled on the processor base unit and then install at least one 9110 processor module into the base. <br> 2. Ensure the program enable key is inserted in the KEY connector on the processor base unit. <br> 3. From the Start menu, click All Programs, then click Rockwell Automation, then AADvance, and then click <br> AADvDiscover. The AADvance Discover tool displays and lists the controllers available on the network. <br> 4. In the list, locate the controller with the MAC address previously noted and ensure the status is Configurable. <br> 5. Double-click the required Controller ID (MAC Address). <br> 6. The Controller Configuration dialog box displays. <br> 7. In the IP Address, Subnet Mask, and Gateway Port fields, type the required values for each Ethernet port. <br> 8. In the Gateway fields, type the required values for each processor module, then click Apply. <br> The AADvance Discover tool refreshes the list of controllers. The controller settings update and the controller is configurable. |
| Configure the Resource Number in a controller | 1. Note the MAC address (Controller ID) labeled on the processor base unit and then install at least one 9110 processor module into the base. <br> 2. Ensure the program enable key is inserted in the KEY connector on the processor base unit. <br> 3. From the Start menu, click All Programs, then click Rockwell Automation, then AADvance, and then click AADvance Discover. <br> The AADvance Discover tool displays and lists the controllers available on the network. <br> 4. In the list, locate the controller with the MAC address previously noted and ensure the status is Configurable. <br> 5. Double-click the required Controller ID (MAC Address). <br> 6. The Controller Configuration dialog box displays. <br> 7. In the Resource Number field, type the required value, then click Apply. <br> 8. The AADvance Discover tool refreshes the list of controllers. The controller status changed from Configurable to Pending Restart. <br> 9. To finish applying the modification, turn off power to the physical controller, and then restart the controller. <br> 10. In AADvance Discover, click Refresh to display the controller information. The resource number of the controller updates and the status is now Configurable. |


|  | Important: The Resource Number must also be configured in <br> AADvance-Trusted SIS Workstation software. |
| :---: | :---: |
| Configure Ethernet forwarding in a controller | 1. Ensure the program enable key is inserted in the KEY connector on the processor base unit. <br> 2. Stop the physical controller. <br> 3. From AADvance Discover, locate the required controller and ensure the status is Configurable. <br> 4. Double-click the required Controller ID (MAC Address). <br> 5. From the Controller Configuration dialog box, select or clear Enable Ethernet Forwarding, then click Apply. <br> 6. Remove the program enable key and start the controller. |
| View the MAC Addresses of Ethernet ports | 1. From AADvance Discover, double-click the required Controller ID (MAC Address). <br> 2. From the Controller Configuration dialog box, click View MAC addresses.... |
| Save the configuration of a controller | 1. In the AADvance Discover tool, double-click the required Controller ID (MAC Address). <br> 2. In the Controller Configuration dialog box, type the required configuration information, and then click Save. <br> 3. In the Save configuration file as... dialog box, specify a name and save location, then click Save. |
| Load the configuration into a controller | 1. In the AADvance Discover tool, double-click the required Controller ID (MAC Address). <br> 2. In the Controller Configuration dialog box, click Load. <br> 3. In the Select configuration file to load dialog box, browse to select the file to load into the controller, and then click Open. <br> 4. In the Controller Configuration dialog box, click Apply. |
| Configure the IP Address of the controller in AADvance-Trusted SIS Workstation software | 1. From the AADvance ${ }^{\ominus}$-Trusted ${ }^{\oplus}$ SIS Workstation software, open or create the required project. <br> 2. From the View menu, click Communication View. <br> 3. In the Communication View, right-click Ethernet, and then click Open. <br> The Communication property page displays with the six Ethernet Ports. <br> 4. In the Communication property page, type the IP address for each required port. |


| Task | Procedure |
| :--- | :--- |
| Configure the resource number of the | 1. From the AADvance-Trusted SIS Workstation software, open or create the required project. |
| AADvance controller | 2. From the View menu, click Application View. |
|  | 3. In the Application View, right-click the required controller, and click Properties. <br> The Properties window displays. |
|  | 4. From the Properties window, in the Number property, type the same resource number defined in AADvance |
| Discover. |  |

Using the AADvance-Trusted SIS Workstation software, configure the controller to use the same IP Address set in the physical controller.

The AADvance controller provides up to six Ethernet communication ports, two for each 9110 processor module present. Each pair of Ethernet ports is identified as $\mathrm{E} n-1$ and $\mathrm{E} n-2$ where n indicates the processor module. The En-1 Ethernet ports are all on one network and the En-2 ports are on a second network.

Tip: To avoid unexpected ARP requests, it is recommended to configure IP addresses for En-2 ports (even if they are unused).

The AADvance controller stores the IP address data in non-volatile memory in the 9100 processor base unit. The controller retains the address information when removing a processor module since the data is independent of the 9110 modules.

Allocate a local network IP address or use an address from the following ranges of private networks:

- 10.0.0.0 to 10.255.255.255
- 172.16.0.0 to 172.31.255.255
- 192.168.0.0 to 192.168.255.255

Each controller on a local area network must have a unique IP address.

| Important: | The two Ethernet ports on each 9110 processor module must be on <br> different subnets. |
| :--- | :--- |

Only define one Gateway address per processor module.

| Important: | Communication blocking or miss-routing by Windows ${ }^{®}$ (or other <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br> displaying certain controllers. DCP communications are unable to go <br> through network bridges and routers. When using virtual machines, <br>  <br>  <br>  <br> use a direct bridged connection. It may be necessary to disable <br> firewalls blocking access to AADvDiscover.exe. |
| :--- | :--- |

AADvance Discover is accessible from the Windows Start menu or from the AADvance-Trusted SIS Workstation software (in the View menu).

The DCP is proprietary to Rockwell Automation ${ }^{\circledR}$. The protocol uses the MAC address of the 9100 processor base unit to identify individual controllers. In AADvance Discover, the MAC address of the base unit displays as the Controller ID. Also view the MAC addresses of the six Ethernet ports.

A controller has one of the following statuses:

| Status | Description |
| :---: | :---: |
| Configurable | The controller is available for configuration. The following criteria are satisfied: <br> - AADvance Discover has established communications with the controller <br> - The program enable key is present <br> - No application is loaded or an application is loaded but not running <br> Set the Resource Number, IP Address, Subnet Mask, Gateway, and Gateway Port information for this controller. |
| Locked | AADvance Discover has established communications with the controller, but one or more of the following criteria is not satisfied: <br> - The program enable key is present <br> - No application is loaded or an application is loaded but not running Resource Number, IP Address, Subnet Mask, Gateway, and Gateway Port information for this controller are view only. |
| No Response | The controller is turned off or the communications between the computer running AADvance Discover and the controller have failed. |
| Pending Restart | Restart the controller after making modifications to a controller available for configuration. <br> Before restarting the controller, modify the Resource Number, IP Address, Subnet Mask, Gateway, and Gateway Port information. |

Click Refresh in the AADvance Discover tool to update the list of controllers. When refreshing, a status bar displays at the bottom of the window, below the list. The status bar displays the following messages:

- Initializing, the AADvance Discover tool starts
- Searching, the tool locates all the controllers connected to the network
- Querying, the tool displays the resource number and status of the controllers
- Ready, the information that displays in AADvance Discover is ready for use


## See also

AADvance Discover on page 295

## Ethernet Forwarding

Ethernet forwarding enables the Ethernet ports of an AADvance controller to re-send Ethernet packets intended for a third-party device along with unicast, broadcast, and multicast messages. A device connected through the processor module gets an IP configuration through BOOTP, DHCP, or statistically. Enable

Ethernet forwarding from the Controller Configuration dialog box. The following diagram displays Ethernet forwarding:


In Ethernet forwarding, the processor module located in the first slot of a 9100 base unit forwards the messages from port E1-1 to E1-2 and vice versa. Other processor modules fitted in the base unit behave the same way (E2-1 to E2-2 and vice versa, E3-1 to E3-2 and vice versa). In each instance, the second port represents an uplink to the network or to another network (if applicable). A device connected to the second port sees all the relevant messages passing through.

| Important: | Do not bridge safety and non-safety networks. Ethernet forwarding |
| :--- | :--- |
|  | is not designed to link different processor modules. |

Regardless of the setting for Ethernet forwarding, the processor module consumes the intended unicast, broadcast, and multicast messages. The controller stores the setting for Ethernet forwarding when replacing any 9110 processor modules. Corrective maintenance is unaffected by Ethernet forwarding.

## See also

AADvance Discover on page 295

## Compiler Verification

The Compiler Verification Tool (CVT) verifies the following:

- Compiler output is correct
- Compiler output matches source code input
- Compiler output contains only the essential elements of the source code input.

The compilation verification process compares individual program organization unit (POU) STF files with their resulting TIC code to validate the accuracy of the compilation process. The verification process starts when the compiler completes the generation of the TIC code from the STF file. The process consists of converting the STF file to an XML file and disassembling the TIC code towards another XML file, then comparing both resulting XML files for differences. Separate utilities perform the conversion of the STF file and the disassembly of the TIC code.


Apply the guidelines detailed in Applying Best Practices to make sure the CVT provides an efficient and accurate analysis of the source TIC code.

## To enable the compiler verification

1. From the Properties window of a controller, expand Compiler Options.
2. Select Enable Compiler Verification, and then select True.
3. From the Properties window of an FBD or LD program, select Generate Monitoring Symbols, and then select True.

## See also

## Applying best practices on page 27

## FBD Language

The Functional Block Diagram (FBD) is a graphic language enabling programmers to build complex procedures by taking existing functions from the standard library, function section, or function block section.

In FBD containers, also include Ladder Diagram (LD) elements like coils, contacts, jumps, labels, and returns. However, in contrast to LD elements usage in LD containers where these elements follow strict graphical positioning regulations, LD elements within FBD container are independent of these regulations.

## See also

FBD Diagram Main Format on page 302
Execution Order of FBD Programs on page 303
Debugging FBD Programs on page 304
FBD Elements on page 305
Function Block Diagram Settings on page 84

## FBD Diagram Main Format

Function Block Diagram (FBD) diagrams describe a process between input variables and output variables. A process is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. Outputs of blocks can also be connected to inputs of other blocks.


An entire process represented by an FBD program is built using the available variables, operators, functions, and function blocks. Each block has either a fixed or defined number of input and 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 inside its rectangle shape. Each input or output of a block is labeled and 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 literal, any internal or input variable, an output variable, or a block output.

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 can be used to connect two logical points of a 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

The connection is oriented, meaning that the line carries associated data from left to right. The left and right ends of the connection line must be of the same data type.

Multiple right connections, also called divergences can be used to broadcast information from their left end to each of the right ends. All ends of the connections must be of the same data type.

## See also

FBD Elements on page 305

## Execution Order of FBD

Programs

The execution order of a Function Block Diagram (FBD) diagram is the order in which the compiler executes the elements making up a program. Display the order of execution in the form of numerical tags for the following elements: coils, instances of function blocks, and variables where a value is assigned in the program. Perform this task from the menu bar, the toolbar, or using the keyboard shortcut (Ctrl+W).

For the execution order of a program, a block is any object in the diagram, a network is a group of blocks linked together, and the position of a block is based on its top-left corner. For function blocks, the instance name is included in the bounding box and considered the top-left corner when defining the execution order Rules that apply to the execution order of the program are:

- Elements execute from top to bottom, then left to right.
- All inputs must be resolved or known, for an element in order to assign its outputs. A constant or unassigned variable is considered resolved. When the inputs of two or more elements are resolved at the same time, the decision for the execution is based on the position of the element (top to bottom, then left to right).
- The order of execution is assigned from 1 to n following the order in which the assignable elements are found.

When analyzing the execution order for the following network, the compiler proceeds through the elements as indicated:

1. The compiler immediately locates the topmost element FB1_1 in the diagram and notices that only the first two inputs of three inputs are resolved.
2. To resolve the third input of element FB1_1, the compiler considers TON_1 where all inputs are resolved and assigns the first tag (1) for the execution order, since this block is the first to be resolved.
3. Since resolving TON_1 enables resolving all inputs for FB1_1, the compiler assigns the next tag (2) to FB1_1.
4. Because FB1_1 is resolved, the compiler starts assigning tags to the FB1_1 outputs which is not yet done for TON_1.
5. Since varBoolOut2 is the top-leftmost of the FB1_1 outputs, the compiler assigns the third tag (3) and assigns the fourth tag (4) to the FB2_1 block.
6. While FB1_1 is completely resolved, the compiler proceeds to resolving the FB2_1 outputs.
7. The compiler assigns the fifth tag (5) to varDintOut1 since this output is the topmost of those to resolve, then the sixth tag (6) to varBoolOut1.
8. The compiler assigns the next and final tag (7) to varTimeOut1.


## To display the execution order in an FBD diagram

Important: After importing an AADvance Workbench version 1.1, 1.2, 1.3, or 1.4 project and opening a POU, check out the POU before displaying the execution order.

- From the Format menu, click Execution Order.


## See also

Debugging FBD Programs on page 304
FBD Keyboard Shortcuts on page 321

## Debugging FBD Programs

When debugging Functional Block Diagram (FBD) programs, monitor the output values of variables. These values display using color, numeric, or textual values according to their data type:

- Output values of Boolean type display using color. The output value color continues to the next input. When the output value is unavailable, Boolean elements remain black. The colors are red when True and blue when False.

[^8]- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are display as a numeric or textual value in the variable. When the output is a structure type, the display value is the selected member.

For structure-type variables, display one simple-type field.
When the output value for a numeric or textual value is unavailable, WAIT is displays in the output label. Values also display in the corresponding dictionary instance.

## See also

## Execution Order of FBD Programs on page 303

FBD Keyboard Shortcuts on page 321
When programming in Function Block Diagram (FBD), place elements in the workspace by dragging them from the Toolbox into the language container. For FBD program organization units (POUs), available elements are:

- Blocks
- Variables
- Vertical Bars
- Labels
- Jumps
- Returns
- Rungs
- Left Power Rails
- Right Power Rails
- Coils
- Contacts
- Regions
- Comments

For links between FBD elements, set the property Is Negation to TRUE to place a negation at the end of a link. Also, indicate whether to display an arrow at the end of a link by enabling the property Is Arrow. The Line Style property specifies the style of the link where the possible values are solid, dash, dot, dash-dot, and dash-dot-dot. The Line Type property specifies the type of line used for links. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over.

For variables connected to variable, coil, and contact elements, add these to existing variables groups from the language container by right-clicking the variable, pointing to Add to Variable Group, then clicking the required group.

## See also

## FBD Diagram Main Format on page 302

Reset Container and Shape Settings in FBD POUs on page 34
Block elements can be operators, functions, or function blocks. Connect block inputs and outputs to variables, contacts or coils, or other block inputs and outputs. Insert block elements in language containers.

Functions and function blocks are represented by a box displaying the name of the function, function block, or operator, and the parameter short names. For function blocks, the instance name displays in italics.

For functions, the return parameter is the only output. For function blocks, multiple return parameters can provide multiple outputs. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.


Define the parameters of program organization units (POUs) for multiple controllers by navigating the tabs for individual controllers that display in the parameter view.

For loops in blocks, use local variables since these are initialized with a value. The following Functional Block Diagram (FBD) diagram shows valid loops where local variables initialize both inputs of the greater than or equal operator and one input of the first AND operator while the output of the second AND operator initializes the second input of the first AND operator.

Using loops with functions or operators may yield unpredictable results since these types of blocks use temporary variables generated by the FBD editor. For predictable results, it is recommended to assign variables in loops. The following example displays an FBD diagram with a loop that results in an unpredictable result. When a variable is added in the loop, the result becomes predictable.

Tip: Since instances of function blocks are defined in the library, the FBD editor does not generate temporary variables.

## Unpredictable Result



Predictable Result


Blocks elements may be resized.

## To insert a block element

1. From the Toolbox, drag the block element into the language container.

The block selector displays.
2. In the Block Selector, choose the required function block, then click OK. Sort the block list according to the columns by setting these in ascending or descending order.
The selected block displays in the language container.
Tip: If deleting a library function block instance from a POU by mistake, do not use the Undo command. Insert the library function block again.

## See also

FBD Variables on page 307
Coils on page 312
Contacts on page 316
Execution Order of FBD Programs on page 303
Block Library on page 36

## FBD Variables

To connect a new symbol to an existing one (another variable, a block input, or a block output) in the workspace, 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.


Replace existing variables in program organization units (POUs) by doubleclicking them to access the Variable Selector or single-clicking them to select from a drop-down combo box containing the global and local variables. Also, single-click a variable, then type a literal value in the text box provided. When inserting literal values that being with a letter or an underscore, enclose these in single quotes as follows: 'abc'.


When selecting items like local variables, controller variables, system variables, and defined words from the drop-down combo box, typing characters in the text box focuses on the possible items. Focus on listed items
by typing letters, digits, and specific special characters: !, \#, \$, \%, \&, <br>, ${ }^{*},+,-$, ,/ $<,:,=,>, ?, @, \backslash, \wedge, \_, \quad \mid,$ and ~.

For input and output variables, choose to display comments entered in the dictionary. From the View menu, access the Properties window and define the Comment Position property.


Resize variables that display in the workspace.

## To insert a variable

1. From the Toolbox, drag the variable element into the language container. The Variable Selector displays.
2. In the Variable Selector, select the required variable, then click OK.

## See also

Coils on page 312
Contacts on page 316
FBD Elements on page 305
Vertical bars are graphic components of Functional Block Diagram (FBD) programs enables closing multiple parallel links. More than one horizontal links on the left side of a vertical bar are connected to one link on the right side. The Boolean state of the right end is the logical OR between all the left extremities.


## To insert a vertical bar

- From the Toolbox, drag the vertical bar element into the language container.

The vertical bar appears in the language container.

## See also

FBD Elements on page 305
FBD Diagram Main Format on page 302
Place labels anywhere in a Functional Block Diagram (FBD) diagram. Use labels as a target for Jump instructions, to change the execution order of the diagram. Labels are not connected to other elements.

Place Labels on the left of the diagram in order to increase diagram readability.

Labels are used to control the execution of the diagram. No other object may be connected on the right of a label symbol.
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



## To insert a label

1. From the Toolbox, drag the label element into the language container.
2. In the language container, click the label, then type a label name in the space provided. The label displays in the language container.

## See also

FBD Elements on page 305
FBD Diagram Main Format on page 302
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.

Jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump symbol.

Inserting backward jumps in the diagram may block the programmable logic controller (PLC) loop.

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



Tip: Before inserting jumps, define one or more labels within the program.

## To insert a jump to a label

1. From the Toolbox, drag the jump element into the language container.
2. In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed in the language container with the required label name.

## Returns

If the connection line (to the left of the Return element) has the Boolean state TRUE, the Program ends - no further part of the diagram is executed.

No connection can be placed to the right of a Return element.

The Return element must be connected to a Boolean output connection point of a block. This element represents a Conditional End of the program: if the output of the box connected to the Return element 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;

## To insert a return

- From the Toolbox, drag the return element into the language container.


## See also

FBD Elements on page 305
FBD Diagram Main Format on page 302

## Rungs

Rungs are graphic components of Functional Block Diagram (FBD) programs and represent a group of circuit elements leading to the activation of a coil. Dragging the rung element into the workspace inserts a left power rail linked to a right power rail. Also, the rung contains a direct contact and a direct coil. Error symbols ( ${ }^{1}$ ) indicate that the direct contact and direct coil are undefined.


## To insert a rung

- From the Toolbox, drag the rung element into the language container.


## See also

FBD Elements on page 305
FBD Diagram Main Format on page 302

Left Power Rails

Right Power Rails

Coils

Left Power Rails are graphic components of Functional Block Diagram (FBD) programs that represent the left boundary of a rung. Any horizontal link connected to a left power rail has the Boolean state TRUE.

You can link left power rails to right power rails and many FBD and Ladder Diagram (LD) elements, including variables, blocks, jumps, returns, vertical bars, coils, and contacts.

## To insert a left power rail

- From the Toolbox, drag the left power rail element into the language container.


## See also

Right Power Rails on page 312
FBD Elements on page 305
Right Power Rails are graphic components of Functional Block Diagram (FBD) programs that represent the right boundary of a rung.

You can link right power rails to left power rails and many FBD and Ladder Diagram (LD) elements, including variables, blocks, vertical bars, coils, and contacts.

## To insert a right power rail

- From the Toolbox, drag the right power rail element into the language container.


## See also

Left Power Rails on page 312
FBD Elements on page 305
Coils are graphic components of Ladder Diagram (LD) programs that can be used in Function Block Diagram (FBD) programs representing the assignment of Boolean outputs. 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.

These types of coils are available from the FBD stencil:

- Direct Coil
- Reverse Coil
- Set Coil
- Reset Coil

Change the type of a coil at any time after insertion.
When inserting coils in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables are displayed above the coil elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values that begin with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.


| Task | Procedure |
| :--- | :--- |
| Insert a coil | 1. From the Toolbox, drag the desired coil type into the language container and place it on the rung. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> The coil element and its associated variable name are displayed in the language container. <br> 3. (optional) To change the type of a coil, select the coil, and then select the required type in the Modifier property of the Properties <br> window. |
| Insert a parallel coil | 1. From the Toolbox, drag a contact element into the language container while placing it parallel to the existing contact. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> 3. Drag the left and right connections to the respective connection points on the rung. <br> The contact and its associated variable name are displayed on the branch. |
| Change the type of a coil | •In the language container, select the coil, then select the required type in the Modify property of the Properties window. |

## See also

Direct Coil on page 313
Reverse Coil on page 314
Set Coil on page 315
Reset Coil on page 315
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 can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := input1;
output2 := input1;
```


## See also

## Reverse Coil on page 314

Set Coil on page 315
Reset Coil on page 315
Coils on page 312

## Reverse Coil

Reverse 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. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```


## See also

Direct Coil on page 313
Set Coil on page 315
Reset Coil on page 315
Coils on page 312

## 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)

```
IF input1 THEN
    output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END_IF;
```


## See also

Direct Coil on page 313
Reverse Coil on page 314
Reset Coil on page 315
Coils on page 312


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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

Example

(* ST Equivalence: *)
output1 $:=$ input1 AND input2;

## See also

Direct Coil on page 313
Reverse Coil on page 314
Set Coil on page 315
Coils on page 312
Contacts are graphic components of Ladder Diagram (LD) diagrams that can be used in Functional Block Diagram (FBD) programs. Depending on the type of contact, it represents the value or function of an input or internal variable.

These contact types are available from the FBD stencil:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

Change the type of a contact at any time after insertion.
When inserting contacts in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables are displayed above the contact elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo boxes
containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.


| Task | Procedure |
| :--- | :--- |
| Insert a contact | 1. From the Toolbox, drag the desired contact type into the language container and place it on the rung. <br> The Variable Selector appears. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> The contact and its associated variable name appear in the language container. |
| Insert a parallel contact | 1. From the Toolbox, drag a contact element into the language container while placing it parallel to the existing contact. <br> The Variable Selector appears. |
| 2. In the Variable Selector, select the required variable, then click OK. <br> 3. Drag the left and right connections to the respective connection points on the rung. <br> 4. The contact and its associated variable name appear on the branch. |  |
| Change the type of a contact | In the language container, select the contact, then select the required type in the Modifier property of the Properties window. |

## See also

Direct Contact on page 317

## Reverse Contact on page 318

Pulse Rising Edge Contact on page 319
Pulse Falling Edge Contact on page 319
Direct Contact
Direct contacts enable a Boolean operation between a connection line state and a Boolean variable.

|  |  |
| :--- | :--- |
| Left | Right <br> Connection <br> Connection |

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;
```


## See also

## Reverse Contact on page 318

Pulse Rising Edge Contact on page 319
Pulse Falling Edge Contact on page 319
Contacts on page 316
Reverse contacts enable 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);
```


## See also

Direct Contact on page 317
Pulse Rising Edge Contact on page 319
Pulse Falling Edge Contact on page 319
Contacts on page 316

Pulse Rising Edge Contact

## Example


(* ST Equivalence: *)
output1 $:=$ input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)

## See also

## Direct Contact on page 317

Reverse Contact on page 318
Pulse Falling Edge Contact on page 319
Contacts on page 316

## Pulse Falling Edge Contact

Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling 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 falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

## Example



## See also

Direct Contact on page 317
Reverse Contact on page 318
Pulse Rising Edge Contact on page 319
Contacts on page 316
Regions
Regions delineate and group together areas of a Function Block Diagram (FBD) program organization unit (POU). A region consists of a header and a delineated zone grouping together elements. The header section enables entering free-format text. After entering text in the header, click elsewhere in the region to exit editing mode. When moving the location of a region in the language container, also move all the content grouped within. Resize regions.


| Task | Procedure |
| :--- | :--- |
| Insert a region | From the Toolbox, drag the region element into the language container. |
| Move a region | 1. In the language container, left-click the top right corner of the region element and hold the mouse button. |

## See also

FBD Elements on page 649

## Comments

Comments are free format text inserted anywhere in the Functional Block Diagram (FBD) program organization unit (POU), for documentation purposes only.

After entering text, click elsewhere in the workspace to 'validate' the comment.

Expand and collapse comment elements that display in the workspace by clicking the maximize and minimize buttons. Also resize comments.


## To insert a comment

1. From the Toolbox, drag the comment element into the language container.
2. In the language container, double-click the comment, then type the required text within the space provided.

## See also

FBD Elements on page 305

## FBD Keyboard Shortcuts

These keyboard shortcuts are available for use with the Function Block Diagram (FBD) language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while <br> debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not <br> available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while <br> debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input <br> automatically opens the Block Selector and Variable Selector (not <br> available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+। | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | When a function block is selected, opens the Block Selector (not <br> available while debugging). <br> When a variable is selected, opens the Variable Selector (not available <br> while debugging). <br> When a comment is selected, starts editing it (not available while <br> debugging). |
| Ctrl+Enter | When a variable is selected, opens the drop-down list of available <br> variables (not available while debugging). <br> When editing a comment, confirms the text (not available while <br> debugging). |
| Ctrl+= | Decreases the magnification |
| Increases the magnification |  |


| Keyboard Shortcut | Description |
| :---: | :---: |
| CtrI+0 | 100\% magnification |
| Ctri+ 1 | Inserts a variable (not available while debugging) |
| Crr\|+2 | Inserts a function block (not available while debugging) |
| Ctrl+3 | Inserts a comment (not available while debugging) |
| Shift+Up Arrow | Reduces the height of the selected element (not available while debugging) |
| Shift+Down Arrow | Increases the height of the selected element (not available while debugging) |
| Shift+Left Arrow | Reduces the width of the selected element (not available while debugging) |
| Shift+Right Arrow | Increases the width of the selected element (not available while debugging) |
| Ctrl+Up Arrow | Moves the selection to the next element located higher in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Down Arrow | Moves the selection to the next element located lower in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Left Arrow | Moves the selection to the next element located to the left in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Right Arrow | Moves the selection to the next element located to the right in the diagram without keeping the previous element selected (not available while debugging) |
| Alt+Shift+Up Arrow | When a function block is selected, navigates up the different inputs and outputs (not available while debugging) |
| Alt+Shift+Down Arrow | When a function block is selected, navigates down the different inputs and outputs (not available while debugging) |
| Alt+Shift+Left Arrow | When a function block is selected, navigates left across the different inputs and outputs (not available while debugging) |
| Alt+Shift+Right Arrow | When a function block is selected, navigates right across the different inputs and outputs (not available while debugging) |
| Ctrl + Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Up Arrow | Moves selected elements up the language container. While debugging, scrolls up. |
| Down Arrow | Moves selected elements down the language container. While debugging, scrolls down. |
| Left Arrow | Moves selected elements left across the language container. While debugging, scrolls left. |
| Right Arrow | Moves selected elements right across the language container. While debugging, scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |
| Ctrl+D | Only available in debug mode for the date data type. When the Write Logical Value dialog box is open, enters the current date. |
| Ctrl+W | Displays the execution order of elements within the diagram (not available while debugging) |

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

## LD Language

by placing graphic symbols into the program chart. LD graphic symbols are organized within the chart as an electric contact diagram. Thus, the term "ladder" coming from the concept of rungs connected to vertical power rails at both ends where each rung represents an individual circuit.


Adjust editor and view settings for individual or all Ladder Diagrams. When working in a Ladder Diagram, set the properties for the diagram from the Container properties in the Properties window. Set the properties for all Ladder Diagrams using the options available from the Tools menu. Some of the available properties include:

- background and gradient colors for operators, functions, and function blocks
- displaying the grid and the height and width of grid cells, in pixels
- height and width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. For example, note the contact using the default settings of one grid cell high by four grid cells wide. The following block uses a basic element width for the inputs, another for the block, and another for the outputs. The block uses a basic element height for the EN/ENO level, another for the first input and the output, and another for the second input.

- font type, size, style, and color applied to the text displayed in elements
- various options such as displaying comments and labels, aligning coils, and setting the colors for variables, labels, comments, power rails, and rung headers


## See also

Debugging LD programs on page 324
LD Elements on page 326
Ladder Diagram Settings on page 87
Debugging LD programs
Debug Ladder Diagram (LD) programs to monitor the output values of elements. These values display using color, numeric, or textual values according to their data type:

- Output values of Boolean type display using color. The output value color continues to the next input. Boolean elements remain black when the output value is unavailable. The default colors are red when True and blue when False. Customize colors for Boolean items.
- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING
type display as a numeric or textual value in the element. The display value is the selected member when the output is a structure type.


For structure-type variables, display one simple-type field.
The WAIT text displays in the output label when the output value for a numeric or textual value is unavailable. Transitional elements like Pulse rising edge (positive) contacts with an unstable state remain black. Values also display in the corresponding dictionary instance.

While simulating an application, debug LD programs with breakpoints, cycle-to-cycle, and step-by-step execution. Breakpoints, cycle-to-cycle, and step-bystep execution are not available while connected to controllers. Enable step-by-step execution by generating debug information for individual program organization units (POUs). The application switches to step-by-step execution upon encountering a breakpoint when debug information is generated for LD programs. Set breakpoints to rungs to instantiate step-by-step execution. The application stops upon encountering a breakpoint when debugging. At this time, the application is in the DEBUGGING state and these operations are available:

- Step into the next rung, executes the next rung then steps into the subsequent rung. When the next rung includes a call to a function, stepping continues in the called function then returns to the next rung in the POU.
- Step over the next rung, executes the current rung then steps to the next rung

Breakpoints appear as red circles to the left of the rung in the language editor. A yellow arrow displays on the breakpoint and the next rung highlights in yellow when the application encounters a breakpoint.


The arrow points downward when stepping passes beyond the last rung of a POU.

While debugging, toggle the Boolean logical value of a selected contact, coil, or function block input.

| Task | Procedure |
| :--- | :--- |
| Generate debug information for an LD POU | Only available for simulation. <br> 1. In the Application View, select the LD POU for which to generate debug information. <br> 2. In the Properties window for the POU, set Generate Debug Info to True. |
| Set breakpoints in an LD POU | Only available for simulation. <br> Tip: Setting a breakpoint in a POU during simulation checks out the POU once <br> the simulation stops. |
| Remove breakpoints | 1. In the Properties window for the POU, set Generate Debug Info to True. <br> 2. Select the rung or rungs requiring breakpoints, right-click in the rung area, and then click Add Breakpoint. <br> Breakpoints appear as red circles to the left of rungs. |
| Toggle the logical value of a contact coil or | Only available for simulation. Remove breakpoints in the RUNNING state or cycle-to-cycle execution while simulating. <br> Select the rung or rungs requiring the removal of breakpoints, right-click the rung area, and then click Remove <br> Breakpoint. |
| Available for simulation and running online connected to controllers. |  |
| Select the required element, then right-click Toggle Boolean Value (or press Ctrl+T). |  |

Tip: $\quad$ Set and remove breakpoints in the RUNNING state or cycle-to-cycle execution while simulating.

## See also

## Simulate an application on page 137

## LD Elements

When editing a Ladder Diagram (LD) program organization unit (POU), place elements in a language container by dragging them from the LD toolbox or
from a contextual menu accessed by right-clicking. An element is inserted at the current position in the diagram. When inserting subsequent elements, these are placed to the right of the selected element on the rung, then onto the next rung. For LD POUs, elements available are:

- Rungs
- Blocks
- Coils
- Contacts
- Jumps
- Returns
- Branches

While dragging rungs and elements from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For rungs, the possible drop points are displayed between the rung handles on the left. For elements, the possible drop points are displayed on the rung.

For variables connected to block, coil, and contact elements, add these to existing variables groups from the language container by right-clicking the variable, pointing to Add to Variable Group, then clicking the required group.

Drop points between rung handles while dragging a rung in a diagram


Drop points on a rung while dragging elements over the rung


## See also

## LD Rungs on page 328

## Returns on page 344

## Reset Visual Settings in LD POUs on page 35

Rungs are graphic components of Ladder Diagram (LD) programs and represent a group of circuit elements leading to the activation of a coil. Rungs have labels for identification within a diagram. Labels along with jumps enable controlling the execution of a diagram. The label and jump must have the same name. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label element. Comments are free format text inserted above the rung, for documentation purposes only. To the left of a rung, a handle indicates the position within a diagram and enables repositioning by dragging and dropping.

While dragging rungs from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For rungs, the possible drop points are displayed between the rung handles on the left.


| Task | Procedure |
| :--- | :--- |
| Insert a rung | - From the Toolbox, drag the rung element into the language container where a drop point appears. |
| Drag a rung | When dragging rungs, the possible drop points are displayed between the handles on the left. <br> 1. To reposition a rung within a diagram, select the rung handle while holding down the left mouse button. The rung handle <br> indicates the rung position on the left. |
|  | 2. Drag the rung to the required position in the diagram where a drop point is displayed. |
| Define the label for a rung | 1. Right-click a rung, then click Add Label. <br> 2. In the upper left-hand corner, click in the text area beside the grey square and type the required label text. |


| Task | Procedure |
| :--- | :--- |
| Define the comment for a rung | Place comments in the space above the rung. After entering text, click elsewhere in the workspace to 'validate' the <br> comment. Text formatting options including bold, italic, underline, strikethrough, and justify, are available from the Format <br> menu. Using the Format menu, also define the foreground color. <br> - In the language container, click the rectangular space above the rung, then type the required text. |

## See also

## LD Elements on page 326

## Blocks

In a language container, connect blocks to Boolean lines. Blocks can be operators, functions, or function blocks. Boolean inputs and outputs are not always contained within blocks. Boolean inputs connecting blocks to rungs are always executed each cycle. Boolean outputs connecting blocks to rungs control the remaining rung power flow. When inserting blocks in a diagram, the EN and ENO parameters are added to some block interfaces. Also force the inclusion of the EN and ENO parameters for blocks with either one Boolean input, one Boolean output, or no Boolean input and output. Activate the Enable EN/ENO and Display Instance Names options from the Ladder Diagram options.

For functions and function blocks, set the value of return parameters using coils. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.

When working with different controllers, define parameters of program organization units (POUs) for multiple controllers by navigating the tabs for individual controllers that display in the Parameters view.

Insert blocks from the LD stencil in the Toolbox. Set the type of a block using the Block Selector at any time following insertion. When the type of block is set, variables automatically display and are connected to the inputs and outputs of the block.

Replace input and output variables by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combo box containing the global and local variables. Also, single-click a variable, then type a literal value in the text box provided. When inserting literal values with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down Type a literal value in the text box: combo-box:


When selecting items like local variables, controller variables, system variables, and defined words from the drop-down combo box, type characters in the text box to focus on the possible items. Focus on listed items by typing letters, digits, and specific special characters: !, \#, \$, \%, \&, <br>, *, +, -, /, <, :, =, >, ?, @, ^, _ `, l, and ~.

## EN Input

For operators, functions, and function blocks where the first input is not a Boolean data type, another input called EN is automatically inserted at the first position since the first input is always connected to the rung. The block is executed only when the EN input is TRUE. The following example shows a comparison operator and its equivalent code expressed in ST.


## ENO Output

For operators, functions, and function blocks where the first output is not a Boolean data type, another output called ENO is automatically inserted at the first position since the first output is always connected to the rung. The ENO output always has the same state as the first input of the block. The following example shows the AVERAGE function block and its equivalent code expressed in ST.


## EN and ENO Parameters

In some cases, both EN and ENO parameters are required. The following example shows an arithmetic operator and its equivalent code expressed in ST.


While dragging blocks from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung.

Drop points on a rung while dragging elements over the rung.


Tip: When dragging blocks, the possible drop points on a rung display while moving over the rung.

| Drag a block | To reposition a block within the diagram, select the block while holding down the left mouse button, then drag the block to the <br> required location on a rung where a drop point displays. |
| :--- | :--- |
| Access the parameters view | 1. In the Application View, right-click the required function or function block instance in the controller library, and then click <br> Parameters. <br> 2. To define the parameters of a function or function block, select the block, then enter the required information in the fields <br> provided. |
| Insert a block | While inserting or dragging a block, drop points indicate the possible locations where the block can be placed on a rung. <br> 1. From the Toolbox, drag the block element into the language container and place it on the rung where a drop point displays. <br> 2. In the Block Selector, locate the required block. Sort the block list according to the columns by setting these in ascending or <br> descending order. <br> To force the inclusion of the EN/ENO parameters, select Enable EN/ENO. |
| 3. Click OK. |  |

4. 

## See also

Debugging LD programs on page 324
Coils on page 332
Contacts on page 338

## Coils

Coils are graphic components of Ladder Diagram (LD) programs and represent the assignment of Boolean outputs. In an LD program, 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.

These types of coils are available from the LD stencil:

- Direct Coil
- Reverse Coil
- Pulse Rising Edge Coil
- Pulse Falling Edge Coil
- Set Coil
- Reset Coil

Change the type of a coil at any time after insertion.
When inserting coils in POUs, assign variables using the Variable Selector. Names of assigned variables display above the coil elements within program organization units (POUs). Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:


Type a literal value in the text box:


When selecting items such as local variables, controller variables, system variables, and defined words from the drop-down combo box, typing characters in the text box focuses on the possible items. Focus on listed items by typing letters, digits, and specific special characters: !, \#, \$, \%, \&, <br>, *, +, -, ,/ $<,:,=,>, ?, @, \backslash, \wedge, \ldots, \quad \mid$, and ~.

Align the coils of all rungs making up diagrams to improve readability.
While dragging coils from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For elements, the possible drop points are displayed on the rung. For instance, since a rung can only have one coil on the right, the editor does not display possibilities when dragging another coil onto the rung. However, if a parallel branch is placed around the coil, the editor displays a possible location for another coil on the branch.

Drop points on a rung while dragging elements over the rung:


Tip: When dragging coils, the possible drop points on a rung are displayed while moving over the rung.

| Task | Procedure |
| :--- | :--- |
| Drag a coil | To reposition a coil within the diagram, select the coil while holding down the left mouse button, then drag the coil to the <br> required location on a rung where a drop point is displayed. |
| Insert a coil | 1. From the Toolbox, drag the desired coil type into the language container and place it on the rung where a drop point <br> appears. <br> 2. In the Variable Selector, select the required variable, then click 0K. |
| Insert a parallel coil | While inserting or dragging an element, drop points indicate the possible locations where the element can be placed on a <br> rung. <br> 1. From the Toolbox, drag the branch element onto the existing coil in the language container. <br> 2. From the Toolbox, drag the desired coil type onto the branch. <br> 3. In the Variable Selector, select the required variable, then click OK. |
| Change the type of a coil in LD | In the language container, select the coil, and then press the space bar. |
| Align all coils in a diagram | 1. Right-click in the language container, and then click Properties. <br> 2. In the Properties window, set the Coil Alignment property to True. |

## See also

## Contacts on page 338

## LD Elements on page 326

## 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 can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := input1;
output2 := input1;
```


## See also

## Reverse Coil

Reverse coils enable a Boolean output according to the Boolean negation of a connection line state.


Tip: Instead of using a reverse coil, use a NOT operator before a direct coil.

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. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```


## See also

Direct Coil on page 333
Set Coil on page 336
Reset Coil on page 337
Coils on page 332
Pulse Rising Edge Coil
Pulse rising edge coils 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)
IF (inputl and NOT (inputlprev)) THEN
output1 := TRUE;
ELSE
output1 := FALSE;
END_IF;
(* inputlprev is the value of input1 at the previous cycle *)

## See also

Pulse Falling Edge Coil on page 335
Set Coil on page 336
Reset Coil on page 337
Coils on page 332
Pulse falling edge coils 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

Example

(* ST Equivalence: *)
IF (NOT(input1) and inputlprev) THEN output1 := TRUE;
ELSE output1 $:=$ FALSE;
END_IF;
(* inputlprev is the value of inputl at the previous cycle *)

## See also

Pulse Rising Edge Coil on page 335
Set Coil on page 336
Reset Coil on page 337
Coils on page 332
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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example



## See also

## Reset Coil on page 337

Direct Coil on page 333
Pulse Rising Edge Coil on page 335
Coils on page 332
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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)

```
IF input1 THEN
        output1 := TRUE;
END_IF;
IF input2 THEN
        output1 := FALSE;
END_IF;
```


## See also

Set Coil on page 336
Pulse Rising Edge Coil on page 335
Direct Coil on page 333
Coils on page 332

## Contacts

Contacts are graphic components of Ladder Diagram (LD) diagrams.
Depending on the type of contact, it represents the value or function of an input or internal variable.

These contact types are available from the LD stencil:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

Change the type of a contact at any time after insertion.
When inserting contacts in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables display above the contact elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

| var4 |  |
| :--- | :--- |
| value2 |  |
| Var1 |  |
| var2 |  |
| var3 |  |
| var4 |  |
| var5 |  |
| var6 |  |
| var7 |  |
| varBool1 |  |


| 'abci I |  |
| :---: | :---: |
| SYSVA_CCEXEC | A |
| SYSVA_KVBCERR |  |
| SYSVA_WNGCMD |  |
| -10_Qw1_1_0 |  |
| 10_Qw1_1_1 |  |
| -10_QW1_1_10 |  |
| $10-\mathrm{QW} 1$-1_11 |  |
| 10_QW1_1_12 |  |
| -_IO_QW1_1_13 | * |

When selecting items such as local variables, controller variables, system variables, and defined words from the drop-down combo box, typing characters in the text box focuses on the possible items. Focus on listed items by typing letters, digits, and specific special characters: !, \#, \$, \%, \&, <br>, *, +, -, /,


While dragging contacts from either the Toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For elements, the possible drop points are displayed on the rung.


Tip: When dragging contacts, the possible drop points on a rung are displayed while moving over the rung.

| Task | Procedure |
| :--- | :--- |
| Drag a contact | To reposition a contact within the diagram, select the contact while holding <br> down the left mouse button, then drag the contact to the required location <br> on a rung where a drop point is displayed. |
| Insert a contact | 1. From the Toolbox, drag the desired contact type into the language <br> container and place it on the rung where a drop point appears. <br> 2. In the Variable Selector, select the required variable, then click OK. |
| Insert a parallel contact | While inserting or dragging an element, drop points indicate the possible <br> locations where the element can be placed on a rung. <br> 1. From the Toolbox, drag the branch element on the existing contact in the <br> language container. |
| 2. From the Toolbox, drag the desired contact type on the branch. |  |
| 3. In the Variable Selector, select the required variable, then click OK. |  |

## See also

Coils on page 332
LD Elements on page 326

## Direct Contact

Direct contacts enable 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;
```


## See also

Reverse Contact on page 340
Pulse Rising Edge Contact on page 341
Pulse Falling Edge Contact on page 341
Contacts on page 338

Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.


Left Connection Right Connection
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);
```


## See also

Direct Contact on page 340
Pulse Rising Edge Contact on page 341
Pulse Falling Edge Contact on page 341
Contacts on page 338

## Pulse Rising Edge Contact

Pulse rising edge (positive) contacts enable 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. The state 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 *)

## See also

Pulse Falling Edge Contact on page 341
Direct Contact on page 340
Reverse Contact on page 340
Contacts on page 338

## Pulse Falling Edge Contact

Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling 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 falls from TRUE to FALSE. The state 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 *)

## See also

Pulse Rising Edge Contact on page 341
Direct Contact on page 340
Reverse Contact on page 340
Contacts on page 338
Conditional and unconditional jump elements enable controlling the execution of diagrams. Connections cannot be placed to the right of a jump element. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label. The label and jump must have the same name.

Inserting backward jumps in the diagram may block the PLC loop.

## Example:



While dragging jumps from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For elements, the possible drop points are displayed on the rung. For instance, since a rung can only have one jump, the editor does not display possibilities when dragging another jump onto the rung. However, if a parallel branches are placed on a rung, the editor displays possible locations for jumps on the rung and on all branches.


| Task | Procedure |
| :--- | :--- |
| Insert a jump | Before inserting jumps, define one or more labels within the program. Insert jumps from the Toolbox, contextual menus, or <br> using keyboard shortcuts. While inserting or dragging a jump, drop points indicate the possible locations where the jump <br> can be placed on a rung. <br> 1. From the Toolbox, drag the jump element into the language container and place it on the rung where a drop point is <br> displayed. <br> 2. In the language container, click the jump element, then select the required label name from the drop-down combo-box. |


| Task | Procedure |
| :--- | :--- |
| Drag a jump | When dragging jumps, the possible drop points on a rung are displayed while moving over the rung. <br> - To reposition a jump within the diagram, select the jump while holding down the left mouse button, then drag the jump to <br> the required location on a rung where a drop point is displayed. |

## See also

Branches on page 345
Contacts on page 338
LD Elements on page 326

## Returns

Use Return elements as outputs representing a conditional end of a diagram. Connections cannot be placed to the right of a Return element.

When the left connection line has the TRUE Boolean state, the diagram ends without executing the equations located on the next lines of the diagram.

When the LD diagram is a function, its name is associated with an output coil to set the return value (returned to the calling diagram).

While dragging Return elements from either the Toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung. For instance, since a rung can only have one Return, the editor does not display possibilities when dragging another Return onto the rung. However, if a parallel branch is placed around the Return, the editor displays a possible location for another Return on the branch.


Insert Return elements from the Toolbox, contextual menus, or using keyboard shortcuts. While inserting or dragging a Return element, drop points indicate the possible locations where the contact can be placed on a rung.

| Task | Procedure |
| :--- | :--- |
| Insert a return | 1. From the Toolbox, drag the Return element into the language container, placing it on the rung where a drop point is displayed. <br> 2. (optional) To reposition a return within the diagram, select the Return element while holding down the left mouse button, then drag <br> the Return element to the required location on a rung where a drop point is displayed. |
| Drag a return | - To reposition a Return element within the diagram, select the Return element while holding down the left mouse button, then drag <br> the Return element to the required location on a rung where a drop point is displayed. |

## Example


(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

1. 

## See also

LD Rungs on page 328
LD Elements on page 326

## Branches

Branches create alternative routing for connections. Add parallel branches to elements on a rung. Parallel branch structures display with connections upon clicking a rung.

To perform operations like cutting, copying, pasting, and deleting branches, select a single branch or all branches from a parallel structure. When selecting a single branch, click the connection on the left or right extremity of the branch. The branch connection is highlighted and the branch is outlined. When selecting all branches, click any of the vertical lines joining the branch connections on the left or right extremity of the branches. The branch arm is highlighted and all branches in the structure are outlined.


Single Branch Selection


All Branches Selection

While dragging branches from either the Toolbox or within a diagram, the language editor displays the possible drop points. Drop points are display as plus signs in a black circle. For elements, possible drop points display on the rung.


Tip: When dragging branches, the possible drop points on a rung display while
moving over the rung.

## To insert a branch

1. From the Toolbox, drag the branch element into the language container and place in on the rung where a drop point appears.
2. (optional) To reposition a branch within the diagram, select the branch while holding down the left mouse button, then drag the branch to the required location on a rung where a drop point displays.

## See also

Coils on page 332

## Contacts on page 338

LD Elements on page 326

## Selecting LD Elements

In Ladder Diagram (LD) program organization units (POUs), select individual or multiple LD elements to perform management tasks like cutting, copying, pasting, and deleting. Select multiple elements by holding down Ctrl while clicking the left mouse button or by dragging to enclose.

Only select elements having the same level on a rung; elements from different rungs are unavailable. Upon clicking a rung, parallel branch structures display with connections. This diagram shows the selection of two contacts and a parallel branch structure on a rung. These three elements are at the same level while the individual parallel branches are sub-elements of the branch structure.


For branches, select a single branch or all branches from a structure. When selecting a single branch, click the connection on the left or right extremity of the branch. The connection for the selected branch highlights and the branch is outlined. When selecting all branches, click any of the vertical lines joining
the branch connections on the left or right extremity of the branches. The branch arm highlights and all branches in the structure are outlined.


## To select elements

1. To select a single element, click the element on the rung.
2. To select multiple elements, click each of the required elements while holding down the Ctrl key.
3. To select a single branch from a parallel branch structure, click the connection on the left or right extremity of the branch.
The branch connection is highlighted and the branch is outlined.
4. To select all branches from a parallel branch structure, click any of the vertical lines joining the branch connections on the left or right extremity of the branches.
The branch arm is highlighted and all branches are outlined.

## LD Keyboard Shortcuts

Use keyboard shortcuts in Ladder Diagram (LD) program organization units (POUs) to perform many editing operations and navigate the elements making up a diagram. When navigating an LD diagram, the arrow keys enable moving up and down and left and right from rung to rung and from element to element. Enable navigating in a diagram by clicking in the language container.

Keyboard shortcuts available for LD POUs follow. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+A | Selects all rungs (not available while debugging) |
| Ctrl + C | Copies selected elements to the clipboard (not available while <br> debugging) |
| Ctrl +V | Pastes elements saved on the clipboard to the insertion point (not <br> available while debugging) |
| Ctrl +X | Cuts selected elements to the clipboard (not available while <br> debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in rungs (only available when the Display <br> Grid property is set to True) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |


| Keyboard Shortcut | Description |
| :---: | :---: |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+1 | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | Calls the Variable Selector or Block Selector depending on the selected element (not available while debugging) |
| F9 | Toggles between setting or removing a breakpoint on a selected rung (available when Generate Debug Info is True). If more than one rung is selected, only sets a breakpoint on the first selected rung. |
| Space Bar | For coils or contacts, toggles between the available types (not available while debugging) |
| Ctrl+0 | Inserts a rung below a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+Alt+0 | Inserts a rung above a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+1 | Inserts a branch after a selected element (not available while debugging) |
| Ctrl+Alt+1 | Inserts a branch before a selected element (not available while debugging) |
| Ctrl+2 | Inserts a block after a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+Alt+2 | Inserts a block before a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+3 | Inserts a contact after a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl + Alt +3 | Inserts a contact before a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl +5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl+7 | When an element is selected, surrounds the element with a branch (not available while debugging). |
| Page Up | Jumps to the top of the language container |
| Page Down | Jumps to the bottom of the language container |
| Ctrl+Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Ctrl+Up Arrow | Slowly scrolls up. |
| Ctrl+Down Arrow | Slowly scrolls down. |
| Ctrl+Left Arrow | Slowly scrolls left. |
| Ctrl+Right Arrow | Slowly scrolls right. |
| Up Arrow | Moves up the elements. |


| Keyboard Shortcut | Description |
| :--- | :--- |
| Down Arrow | Moves down the elements. |
| Left Arrow | Moves to the left across the elements. |
| Right Arrow | Moves to the right across the elements. |
| Alt+Up Arrow | Selects the previous rung. When no element or rung is selected, <br> selects the last rung. |
| Alt+Down Arrow | Selects the next rung. When no element or rung is selected, selects <br> the first rung. |
| Alt+Left Arrow | Selects the rung of the selected element. When no element is <br> selected, selects the first rung. |
| Alt+Right Arrow | Selects the rung of the selected element. When no element is <br> selected, selects the first rung. |
| Shift+Up Arrow | Scrolls up |
| Shift+Down Arrow | Scrolls down |
| Shift+Left Arrow | Scrolls left |
| Shift + Right Arrow | Scrolls right |
| Delete | Removes a selected rung or element (not available while debugging) |
| Ctrl+D | When in design mode, edits the comment of the selected rung. <br> When in debug mode, for the date data type, enters the current date <br> when the Write Logical Value dialog box is displayed. |
| Ctrl+T | Only available in debug mode for the Boolean data type. Toggles the <br> logical value of a selected contact, coil, or block input. |

## See also

LD Elements on page 326

## ST Language

## ST Main Syntax

Structured Text (ST) 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 is also used for the description of the actions within the Steps and conditions attached to the Transitions of the Sequential Function Chart (SFC) Language.

## See also

ST Main Syntax on page 349
Expressions and Parentheses on page 351
Debugging ST programs on page 353
ST Basic Elements and Statements on page 354
Structured Text Settings on page 90
A Structured Text (ST) program is a list of ST statements. Each statement ends with a semi-colon (";") separator. Inactive or active separators separate names used in the source code (variable identifiers, constants, language keywords...). Inactive separators are space character, end of line or tab stops. Active separators have a well-defined significance, for example, the " $>$ " separator indicates a "greater than" comparison.
Comments enable the inclusion of non-executed information throughout code. Insert comments anywhere in an ST program. Comments can run
multiple lines and must begin with "(*" and end with "*)". Do not use interleave comments, for example, comments within comments.

When typing statements, a drop-down combo box lists the available items such as identifiers, operators, functions, and function blocks. Filter the listed items by typing letters, digits, and specific special characters: !, \#, \$, \%, \&, <br>, *, ,,$+- /<,:,=,>, ?, @, \backslash, \wedge, \quad$,,$~ \mid, ~ a n d ~$.

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

Tip: Use wired variables instead of directly represented variables. The compiler does not support a directly represented variable or a default wired variable after wiring the directly represented variable to a user-defined variable. For example, use _I0_OW1_1_10 instead of \%QW1.1.10.

Basic coding is black while other items display using customizable colors for ST syntax. The default colors for ST elements are the following:

- Comments are green
- The Editor background is white
- Identifiers are black
- Numbers are firebrick
- Operators are black
- Program organization units (POUs) are blueviolet
- Punctuation marks are black
- Reserved words are fuchsia
- Strings of text are gray

Inactive separators between active separators, literals, and identifiers increase ST program legibility. The ST inactive separators are space (blank), tabs, and end of line. Place end of lines anywhere in a program. These rules apply to inactive separators:

- Write one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase legibility of lines or paragraphs


## Examples

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;
```

| Task | Procedure |
| :--- | :--- |
| Customize the default display settings <br> for ST programs | 1. From the Tools menu, click Options. <br> 2. From the Options dialog box, expand IEC Languages, and then click Structured Text (ST). <br> 3. Expand the respective category, customize the required setting, then click OK. |
| Customize the display settings for <br> current ST program | 1. From the View menu, click Properties Window. The Properties window displays. <br> 2. Select the Structured Text (ST) Container. |
|  | 3. From the Properties Window: |
|  | Customize the font for the required item by clicking |
|  | displays allowing customization of the font, text size, bold, italic, strikeout, and <br> underline styles. |
|  | - Customize the text color for the required items. The possible colors are custom, web, and system colors. |

## See also

## ST Basic Elements and Statements on page 354

## ST Keyboard Shortcuts on page 368

Structured Text (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:

## Single expression

| (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 |

Parentheses are used to isolate sub parts of an expression and to explicitly order the priority of 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:

## Parentheses priority

| $2+3 * 6$ | equals $2+18=20$ | because multiplication operator has a higher priority |
| :--- | :--- | :--- |
| $(2+3)^{*} 6$ | equals $5 * 6=30$ | priority is given by parenthesis |

## See also

ST Main Syntax on page 349

## Calling Functions

The Structured Text (ST) programming language enables calling functions. Function calls can be used in any expression.

| Function calls |  |
| :--- | :--- |
| Name | name of the called function written in IEC 61131-3 language or in "C" |
| Meaning | calls ST, LD, or FBD functions or a "C" function and gets its return value |
| Syntax | <variable> <br> $) ;$ |
| Operands | The type of return value and calling parameters must follow the interface defined for <br> the function. |
| Return value | value returned by the function |

When setting the value of the return parameter in the body of a function, assign the return parameter using the same name as the function:

```
FunctionName := <expression>;
```


## Example

Example1: IEC 61131-3 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 61131-3 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);
```


## See also

## ST Basic Elements and Statements on page 354

## Calling Function Blocks

## Debugging ST programs

The Structured Text (ST) programming language enables calling function blocks. Function block calls can be used in any expression.

## Function Block calls

| Name | Name of the function block instance |
| :--- | :--- |
| Meaning | calls a function block from the standard library or from the user's library and accesses <br> its return parameters |
| Syntax | (* call of the function block *) <br> $<$ blockname> ( <pl>, PLC-2 . . ) ; <br> (* gets its return parameters *) <br> <result> $:=$ <blockname>. <ret_param1>; <br> $\ldots$ <br> <result> $:=<$ blockname>. <ret_paramN>; |
| Operands | parameters are expressions which match the type of the parameters specified for that <br> function block |
| Return value | See Syntax to get the return parameters. |

When setting the value of the return parameter in the body of a function block, assign the return parameter using its name concatenated with the function block name:

FunctionBlockName.OutputParaName := <expression>;

## 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;
```


## See also

ST Basic Elements and Statements on page 354
For Structured Text (ST) programs, generate debug information for individual program organization units (POUs) to enable step-by-step execution while simulating an application. For ST programs in a resource with generated debug information, the controller switches to step-by-step execution when the application encounters a breakpoint. Set breakpoints to lines of code to instantiate step-by-step execution.

Tip: Breakpoints, cycle-to-cycle, and step-by-step execution are only available while simulating an application.

In the language editor, breakpoints appear as red circles to the left of the line of code and the line is highlighted in red.

4 MyUINT1 := FctST_UINT(MyUINT2, MyUINT3, MyUINT4 );
While debugging, the application stops when it encounters a breakpoint. At this time, the application is in the DEBUGGING state. Perform one of these operations:

- Step into the line of code, execute the current line of code then steps into the subsequent line of code. When the current line of code includes a call to a function, stepping continues in the called function then returns to the line of code in the POU.
- Step over the line of code, execute the current line of code then steps to the next line of code
- Execute in real-time mode
- Execute one cycle

Tip: Set breakpoints for Target-independent code (TC); C source code POUs do not support breakpoints.

When the application stops on a breakpoint, a yellow arrow appears over the breakpoint and the line of code is highlighted pink.

## 4 MyUINT1 : $=$ FetST_UINT(MyUINT2, MyUINT3, MyUINT4 ) :

The arrow points downward when stepping passes beyond the last line of code of a POU.

| Task | Procedure |
| :--- | :--- |
| Generate debug information for an ST POU | 1. In the Application View, select the ST POU for which to generate debug information. <br> 2. In the Properties for the POU, set Generate Debug Info to True. |
| Set a breakpoint | - Right-click in the margin to the left of the line of code on which to add a breakpoint, then select Add/Remove <br> Breakpoint. |
| Remove a breakpoint | - Right-click in the area to the left of the line of code having a breakpoint to remove, then select Add/Remove <br> Breakpoint. |
| Step into the line of code | - From the Target Execution toolbar, select Go (or press F11). |
| Step over the line of code | - From the Target Execution toolbar, select © (or press F10). |
| Execute in real-time mode | - From the Simulation menu, select Continue. <br> The POU executes in real-time mode. |
| Execute in cycle-to-cycle mode | • From the Simulation menu, select Pause. <br> The POU executes in cycle-to-cycle mode. |
| Execute one cycle | • From the Target Execution toolbar, select |

Tip: Set or remove breakpoints in the RUNNING state or cycle-to-cycle execution while simulating.

## See also

Simulate an application on page 137
ST Main Syntax on page 349

## ST Basic Elements and Statements

The basic elements and statements of the Structured Text (ST) language are:

- Assignments
- CASE Statement
- EXIT Statement
- FOR Statement
- IF-THEN-ELSIF-ELSE-END_IF Statement
- REPEAT Statement
- RETURN Statement
- WHILE Statement

For variables in the syntax, add these to existing variables groups from the language container by right-clicking the variable, pointing to Add to Variable Group, then clicking the required group.

## See also

ST Extensions on page 362
Assignments on page 355

## Assignments

## 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) );
```


## To insert an assignment

- In the language container, type :=.

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.

| CASE Statement |  |
| :--- | :--- |
| Name | CASE ... OF ... ELSE ... END_CASE |
| Meaning | executes one of several lists of ST statements |


| CASE Statement |  |
| :---: | :---: |
| Syntax | ```CASE <integer_expression> OF <value> : <statements> ; <value> , <value> : <statements> ; ... ELSE <statements> ; END CASE;``` |

## 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;
```


## To insert a CASE

- From the Toolbox, drag the CASE element into the language container.


## See also

## ST Basic Elements and Statements on page 354

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

| EXIT Statement |  |
| :--- | :--- |
| Name | EXIT |
| Meaning | exit from a FOR, WHILE or REPEAT iteration statement |
| Syntax | EXIT; |

## 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;

## To insert an EXIT

- In the language container, type EXIT.


## See also

Assignments on page 355
FOR Statement on page 357
REPEAT Statement on page 359
WHILE Statement on page 361
The FOR statement executes a limited number of iterations, using an integer index variable.

| FOR Statement |  |
| :--- | :--- |
| Name | FOR ... TO ... BY ... DO ... END_FOR |
| Syntax | FOR <index> := <mini> TO <maxi> BY <step> DO <br> $\quad$ <statement> ; <br> $\quad$ <statement> ; <br> END_FOR; |
| Operands | index: internal integer variable increased at each loop <br> mini: initial value for index (before first loop) <br> maxi: maximum allowed value for index <br> step: index increment at each loop |

The [ BY step ] statement is optional. If not specified, the increment step is 1 .

| Important: | Because the virtual machine is a synchronous system, input <br> 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;
```


## To insert a FOR statement

- From the Toolbox, drag the FOR element into the language container.


## See also

EXIT Statement on page 356
REPEAT Statement on page 359

## WHILE Statement on page 361

## Assignments on page 355

IF-THEN-ELSIF-ELSE-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 can be used more than once. The ELSE statement, if used, must appear only once at the end of the 'IF, ELSIF...' sequence.

IF-THEN-ELSIF-ELSE-END_IF Statement

| Name | IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF |
| :---: | :---: |
| Meaning | executes one of several lists of Structured Text (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;``` |

AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software evaluates complete Boolean expressions. For instance, evaluating the following line of code, where $i$ represents the array index having a definition of $2 . .10$, causes a run-time error upon reaching the second part where it applies the value 1 as the array index.

IF i >= 2 and i <= 10 and Arrayl[i] > 5 THEN
To avoid this type of error, use the following code:

```
IF i >= 2 and i <= 10 THEN
IF Array1[i] THEN
```


## Example

```
(* ST program using IF statement *)
IF manual AND not (alarm) THEN
    level := manual_level;
    bx126 := bil2 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;
```


## To insert an IF-THEN-ELSIF-ELSE-END_IF statement

- From the Toolbox, drag the IF THEN ELSE element into the language container.


## REPEAT Statement

## See also

## Expressions and Parentheses on page 351

The REPEAT statement is an iteration structure for a group of Structured Text (ST) statements. The "continue" condition is evaluated AFTER any iteration

| REPEAT Statement |  |
| :---: | :---: |
| Name | REPEAT ... UNTIL ... END_REPEAT |
| Syntax | ```REPEAT <statement> ; <statement> ; UNTIL <Boolean_condition> END REPEAT ;``` |


| Important: | Because the virtual machine is a synchronous system, input <br> variables are not refreshed during REPEAT iterations. The change of <br> state of an input variable cannot be used to describe the ending <br> condition of a REPEAT statement. |
| :--- | :--- |

## Example

```
(* ST program using REPEAT statement *)
(* this program uses specific "C" functions to read
characters *)
(* on a serial port *)
```

```
str := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
REPEAT
str := str + ComGetChar ( );
nbchar := nbchar + 1;
UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
END REPEAT;
END_IF;
```


## To insert a REPEAT statement

- From the Toolbox, drag the REPEAT element into the language container.


## See also

## Assignments on page 355

EXIT Statement on page 356
WHILE Statement on page 361
Ends the execution of the current program.

| RETURN Statement |  |
| :--- | :--- |
| Name | RETURN |
| Syntax | RETURN ; |
| Operands | (none) |

In a Sequential Function Chart (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; (* ends the program *)
end_if;
```

```
if RESET then
CV := 0;
else
if (CV < PV) then
CV := CV + 1;
end_if;
end_if;
Q := (CV >= PV);
```


## To insert a RETURN statement

- In the language container, type RETURN.


## See also

REPEAT Statement on page 359
WHILE Statement on page 361
The WHILE statement is an iteration structure for a group of Structured Text (ST) statements. The "continue" condition is evaluated BEFORE any iteration

| WHILE Statement |  |
| :---: | :---: |
| Name | WHILE ... DO ... END_WHILE |
| Syntax | ```WHILE <Boolean_expression> DO <statement>; <statement>; .. END_WHILE;``` |
| Important: | 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 *)
    str := ''; (* empty string *)
    nbchar := 0;
    WHILE ((nbchar < 16) & ComIsReady ( )) DO
    str := str + ComGetChar ( ) ;
    nbchar := nbchar + 1;
    END_WHILE;
```


## To insert a WHILE statement

- From the Toolbox, drag the WHILE element into the language container.


## See also

Assignments on page 355
EXIT Statement on page 356
REPEAT Statement on page 359

## ST Extensions

These statements and functions are available to control the execution of Sequential Function Chart (SFC) child programs. Use these within action blocks written in Structured Text (ST) for SFC steps.

| ST Statements and Functions |  |  |
| :--- | :--- | :---: |
| GSTART | Starts an SFC program or function block |  |
| GFREEZE | Freezes an SFC program |  |
| GKILL | Terminates an SFC program |  |
| GSTATUS | Gets the current status of an SFC program |  |
| GRST | Restarts a frozen SFC program or function block |  |
| Important: | These functions are not part of the IEC 61131-3 standard. |  |

Simple equivalents for the GSTART and GKILL statements are available using this syntax in an SFC step:

- child_name with the S qualifier (* equivalent to GSTART (child_name) ; *)
- child_name with the R qualifier (* equivalent to GKILL(child_name); *)

These fields enable accessing the status of an SFC step or child (from its parent):

| ST Fields |  |
| :--- | :--- |
| StepName.x | Boolean value that represents the activity of the Step |
| StepName.t | time elapsed since the last activation of the step: activity duration <br> ("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 <br> ("ChildName" represents the name of the SFC child) |

## See also

GSTART Statement in SFC Action on page 363
GFREEZE Statement in SFC Action on page 365
GKILL Statement in SFC Action on page 366
GSTATUS Statement in SFC Action on page 365
GRST Statement in SFC Action on page 366

GSTART Statement in SFC Action

Starts a Sequential Function Chart (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.

Children of the child program are not automatically started by the GSTART statement.

Since GSTART is not part of the IEC 61131-3 standard, it is preferable to use the $S$ qualifier attached to the child name.

## GSTART Statement

| Name | GSTART |
| :--- | :--- |
| Syntax | GSTART ( <child_name> ); <br> or <br> GSTART ( <child_name,_step_name,inputl,input2,...inputn> ) <br> where <br> child_name represents the name of the SFC child POU <br> step_name represents the name of the active step. step_name must be preceded by two <br> underscore characters (for example, _-S1) <br> inputl,input2,...inputn 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) |

## Example



## To insert a GSTART statement

- In the language container, type GSTART.


## See also

GFREEZE Statement in SFC Action on page 365
GKILL Statement in SFC Action on page 366
GSTATUS Statement in SFC Action on page 365
GRST Statement in SFC Action on page 366

GFREEZE Statement in SFC Action

Freezes a child Sequential Function Chart (SFC) (program or function block); suspends its execution. The suspended SFC program organization unit (POU) can then be restarted using the GRST statement.

Children of the child program are automatically frozen along with the specified program.

GFREEZE is not part of the IEC 61131-3 standard.

## GFREEZE Statement

| Name: | GFREEZE |
| :--- | :--- |
| Syntax | GFREEZE ( <child_name> ); <br> where <br> 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 <br> written |
| Return value: | (none) |

## To insert a GFREEZE statement

- In the language container, type GFREEZE.


## See also

GSTART Statement in SFC Action on page 363
GKILL Statement in SFC Action on page 366
GSTATUS Statement in SFC Action on page 365
GRST Statement in SFC Action on page 366

GSTATUS Statement in SFC Action

Returns the current status of a Sequential Function Chart (SFC) program.
GSTATUS is not part of the IEC 61131-3 standard.

| GSTATUS Statement |  |
| :--- | :--- |
| Name: | GSTATUS |
| Syntax: | <var> := GSTATUS ( <child_name> ); <br> where <br> 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 <br> written |
| Return value: | $0=$ Program is inactive (killed) <br> $1=$ Program is active (started) <br> 2 = Program is frozen |

## To insert a GSTATUS statement

- In the language container, type GSTATUS.


## See also

GSTART Statement in SFC Action on page 363
GFREEZE Statement in SFC Action on page 365
GKILL Statement in SFC Action on page 366
GRST Statement in SFC Action on page 366
Terminates a child Sequential Function Chart (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.

Children of the child program are automatically terminated with the specified program.

Since GKILL is not part of the IEC 61131-3 standard, it is preferable to use the $R$ qualifier attached to the child name.

GKILL Statement

| Name: | GKILL |
| :--- | :--- |
| Syntax: | GKILL ( <child_name> ) ; <br> where <br> 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 <br> written |
| Return value: | (none) |

## To insert a GKILL statement

- In the language container, type GKILL.


## See also

GSTART Statement in SFC Action on page 363
GFREEZE Statement in SFC Action on page 365
GSTATUS Statement in SFC Action on page 365
GRST Statement in SFC Action on page 366
The GRST statement automatically restarts children of the child program.
GRST is not part of the IEC 61131-3 standard.

## GRST Statement

| Name | GRST |
| :--- | :--- |
| Meaning | Restarts a child Sequential Function Chart (SFC) program frozen by the GFREEZE <br> statement: all the Tokens removed by GFREEZE are restored. The extended syntax only <br> applies to SFC child function blocks. |

## GRST Statement

\(\left.\begin{array}{l|l}\hline Syntax \& GRST ( <child_name> ); <br>
or <br>
GRST ( <child_name,input1,input2,...inputn> ) ; <br>
where <br>
child_name represents the name of the SFC child POU <br>

input1,input2,...inputn indicate the value of the input parameter of the SFC child POU\end{array}\right]\)| Operands | the specified SFC program must be a child of the one in which the statement is written |
| :--- | :--- |
| Return value | (none) |

## Example



## To insert a GRST statement

- In the language container, type GRST.


## See also

GSTART Statement in SFC Action on page 363
GFREEZE Statement in SFC Action on page 365

## GSTATUS Statement in SFC Action on page 365

GKILL Statement in SFC Action on page 366

## ST Keyboard Shortcuts

These keyboard shortcuts are available for use with the Structured Text (ST) language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| CtrI+A | Selects the entire document (not available while debugging) |
| Ctrl+C | Copies the selected text to the clipboard (not available while <br> debugging) |
| CtrI + Insert | Copies the selected text to the clipboard (not available while <br> debugging) |
| Ctrl+ + V | Pastes text saved on the clipboard to the insertion point (not <br> available while debugging) |
| Shift+Insert | Pastes text saved on the clipboard to the insertion point (not <br> available while debugging) |
| CtrI $+X$ | Cuts the selected text to the clipboard (not available while debugging) |

Chapter 2 Working with AADvance Applications

| Keyboard Shortcut | Description |
| :---: | :---: |
| Right Arrow | Moves right across lines and characters |
| Ctrl+Left Arrow | Moves to the previous statement or word |
| CtrI+Right Arrow | Moves to the next statement or word |
| Home | Jumps to the start of the line |
| End | Jumps to the end of the line |
| Ctrl+Home | Jumps to the start of the document |
| Ctrl+End | Jumps to the end of the document |
| Page Up | Jumps to the top of the visible code |
| Page Down | Jumps to the bottom of the visible code |
| Ctrl+Page Up | Jumps to the top of the visible code |
| Ctrl+Page Down | Jumps to the bottom of the visible code |
| Ctrl+Up Arrow | Scrolls up |
| Ctrl+Down Arrow | Scrolls down |
| Shift+Up Arrow | Selects up |
| Shift+Down Arrow | Selects down |
| Shift+Left Arrow | Selects left |
| Shift+Right Arrow | Selects right |
| Ctrl+Shift+Left Arrow | Selects to the previous statement or word |
| Ctrl+Shift+Right Arrow | Selects to the next statement or word |
| Shift+Home | Selects from the insertion point until the start of the line |
| Shift+End | Selects from the insertion point until the end of the line |
| Ctrl+Shift+Home | Selects from the insertion point until the start of the document |
| Ctrl+Shift+End | Selects from the insertion point until the end of the document |
| Shift+Page Up | Selects from the insertion point until the top of the visible code |
| Shift+Page Down | Selects from the insertion point until the end of the visible code |
| Ctrl+Shift+Page Up | Selects from the insertion point until the top of the visible code |
| Ctrl+Shift+Page Down | Selects from the insertion point until the end of the visible code |
| Ctrl+Shift+W | Selects the next word |
| Shift+Alt+Up Arrow | Selects the current and previous lines |
| Shift+Alt+Down Arrow | Selects the current and next lines |
| Shift+Alt+Left Arrow | Selects left on the current line |
| Shift+Alt+Right Arrow | Selects right on the current line |
| Ctrl+Shift+Alt+Left Arrow | Selects available columns in lines of code from the left to right |
| Ctrl+Shift+Alt+Right Arrow | Selects available columns in lines of code from the right to left |
| Escape | Deselects the selected text |
| Ctrl+ | Opens the Variable Selector. (Design mode only) |
| Ctrl+Shift+1 | Opens the Variable Selector. (Design mode only) |
| Ctrl+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Ctrl+Alt + R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Ctrl+Shift+Alt+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Delete | Removes the character on the right (not available while debugging) |
| Ctrl+Shift+L | Removes the current line (not available while debugging) |
| Ctrl+Delete | Removes the next word in the current line (not available while debugging) |
| Ctrl+Backspace | Removes the previous word in the current line (not available while debugging) |
| Backspace | Removes the character on the left (not available while debugging) |
| Shift+Backspace | Removes the character on the left (not available while debugging) |

## See also

## SFC Language

## ST Basic Elements and Statements on page 354

The Sequential Function Chart (SFC) language is a graphic language used to describe operations of a sequential process. This language 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. The SFC Editor is launched automatically when an SFC program is opened or when an SFC program is edited.

SFC is the core of the IEC 61131-3 standard. The other languages 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.

## See also

## SFC Main Format on page 370

SFC Program Hierarchy on page 372

## SFC Main Format

A Sequential Function Chart (SFC) program is a graphic set of steps and transitions, linked together using oriented links. Divergences and convergences represent multiple connection links from 1 to n and n to 1 respectively. The basic graphic rules of SFC are:

- SFC programs must have at least one initial step
- A step must follow a transition
- A transition must follow a step

SFC programs describe sequential operations, where the time variable explicitly synchronizes basic operations. These are called sequential programs. Programs before and after SFC programs describe cyclic operations and are not time-dependent. These are called cyclic programs. Main sequential programs (at the top of the hierarchy) are executed according to the SFC dynamic behavior. Cyclic programs are systematically executed at the beginning of each run time cycle. In Programs sections, sequential programs are grouped together.

Main sequential programs are described with the SFC language; cyclic programs cannot be described with the SFC language. Any SFC program can have one or more SFC child programs.

Functions and function blocks can be called from actions or conditions of SFC programs.

SFC programs and SFC child programs have dynamic behavior limits which are set at the controller level. These dynamic behavior limits determine the amount of memory, allocated by a target at initialization time, designated to manage SFC dynamic behavior (for example, token moving). The amount of
allocated memory is calculated as a linear relation with the number of SFC POUs:

Alloc Mem (bytes) = N * NbElmt * sizeof(typVa)
N.bElmt $=$ GainFactor * NbOfSFC + OffsetFactor

Where:
$\mathrm{N}=5$ (constant linked to SFC engine design)
NbElmt = The maximum number of valid transitions for each executed cycle, for example, transitions with at least one of their previous steps being active.
typVa $=16$ bits in the medium memory model ( 32 bits in the large memory model)

GainFactor and OffsetFactor = the linear parameters of the linear relation
NbOfSFC = the number of SFC POUs in the project
Factor of dynamic behavior limits determining the amount of memory, allocated by a target at initialization time, designated to manage token moving. The amount of allocated memory is calculated as a linear relation with the number of SFC POUs:

```
Alloc Mem (bytes) = N * NbElmt * sizeof(typVa)
NbElmt = GainFactor * NbOfSFC + OffsetFactor
```

These points offer a simplified and more approximate definition of the allocated memory:

- The maximum number of steps that can be active
- The maximum number of actions ( $\mathrm{N}, \mathrm{P} 1$ or P0 action linked to the step) that can be executed

When the available memory is insufficient at a specific moment for a target where check mode (ITGTDEF_SFCEVOCHECK defined in dsysOdef.h) is generated, the target kernel generates a warning. This warning signals an SFC token moving error or an action execution error and the controller is set to ERROR mode where cycles are no longer executed or kernel behavior may become unpredictable.

For SFC function blocks and SFC child function blocks, each has a maximum number of tokens which is set in the properties of the block.

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 controller level.

## See also

SFC Program Hierarchy on page 372
SFC Execution Behavior on page 371

## SFC Execution Behavior

The SFC execution behavior consists of three stages: initial situation (start), code execution, and end. Each virtual machine cycle consists of determining
all clearable transitions and executing their active steps. Execution ends upon reaching unclearable transitions or the end of the control chart.

Within the execution cycle, 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 has three properties: enabled/disabled, active/inactive, and clearable/non-clearable. A transition is enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise, the transition is disabled. A transition is active if its condition is True.

A transition is clearable if it is enabled and active at the same time. When a transition is clearable, the steps immediately preceding it become inactive and those immediately following it become active. When transitions follow a divergence, multiple transitions may become clearable. However, the only transition that is cleared is the one having the highest priority

## 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. The code within a step is only executed if the step is active.

## Simultaneous clearing of transitions

All transitions (of all SFC programs) that can be cleared (enabled and active), are simultaneously cleared.

However, for transitions following divergences, the only transition that is cleared is the one having the highest priority among those that are enabled and active.

## End

The End is characterized by reaching the end of clearable transitions or the end of the control chart.

## SFC Program Hierarchy

The system enables the description of the vertical structure of SFC programs. SFC programs are organized in a hierarchical-tree structure. Each SFC program can control (start, terminate,...) 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 "parent - child" relationship:

## Parent Program



## Child Program

The basic rules implied by the hierarchy structure are:

- SFC programs having no parent are called "main" SFC programs

Within a controller, the relative positions of programs within the hierarchy have certain restrictions:

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

Main SFC programs are activated by the system when the application starts

- A program can have several child programs
- A child of a program can only have one parent
- A child program can only be controlled by its parent
- A program cannot control the children of one of its own children

The basic actions that a parent SFC program can take to control its child program are:

| Actions | Description |
| :--- | :--- |
| Start (GSTART) | Starts the child program: activates each of its initial steps. Children of this <br> child program are not automatically started. |
| Terminate (GKILL) | Stops the child program by deactivating each of its active steps. All the <br> children of the child program are also stopped. |
| Freeze (GFREEZE) | Deactivates each of the active steps of the program, and memorizes them <br> so the program can be restarted. All the children of the child program are <br> also frozen. |
| Restart (GRST) | Restarts a frozen SFC program by reactivating all the suspended steps. <br> Children of the program are not automatically restarted. |
| Get status (GSTATUS) | Gets the current status (active, inactive or frozen) of a child program. |

## See also

SFC Main Format on page 370

## Child SFC POUs

Any Sequential Function Chart (SFC) program organization unit (POU) may control other SFC POUs. These level units are called child SFC POUs. A child SFC POU is a parallel unit that can be started, terminated, frozen, or restarted by its parent. The parent POU and child POU must both be described with the SFC language. A child SFC POU can have local variables.

Tip: When using version control, do not use child SFC POUs. Use only SFC
programs and function blocks in the application.
When a parent POU starts a child SFC, the parent places 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 terminates 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 parent POU starts a child, the parent continues its execution.

When a parent POU freezes a child SFC, the parent 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 for their SFC function block parents, have a maximum number of tokens, unlike SFC programs whose dynamic behavior limits are set at the controller level. Specify the tokens limit for an SFC function block in its settings properties.

When using an SFC function block with an SFC child, access, for read-only purposes, the local values of the child from its parent by entering the child's name and the parameter in an action or transition's code. Example: to access the Local1 parameter of an SFC child named FB_Child, in an action or transition defined for the SFC function block parent, write this syntax:

FB Child.Local1

| Task | Procedure |
| :--- | :--- |
| Add a child SFC POU | - In the Application View, right-click an existing SFC POU, and then click Add SFC Child. |
| Move a child SFC POU | - In the Application View, drag the SFC POU to the required position and hierarchy level. <br> Tip: $\quad$When moving an SFC POU with children, the children follow their parent. A <br> child SFC POU can be moved to a different main SFC POU or to a different <br> hierarchy level. A child SFC POU can only be moved to hierarchical levels <br> that already contain an SFC child. |

## See also

## SFC Program Hierarchy on page 372

## Debugging SFC programs

When debugging Sequential Function Chart (SFC) programs, visually follow the execution of the individual steps. Steps are colored red while active. Also place SFC breakpoints on SFC steps or transitions. The controller switches to the BREAK state upon encountering a breakpoint. This mode is equivalent to the cycle-to-cycle mode. Execute one cycle or switch to real-time mode to pass the breakpoint. When a controller is in the BREAK state and step-by-step execution is activated for other POUs within the controller, also step to the first line of the first POU of the controller for which debug information is generated.

Tip: Breakpoints and execute one cycle execution are only available while simulating an application.

For steps, these types of breakpoints are available:

- Breakpoint on step activation
- Breakpoint on step deactivation

Breakpoints appear as red circles with a white checkmark on the left part of the step or transition. For steps, breakpoints on activation appear at the top corner while breakpoints on deactivation appear at the bottom corner.

| Task | Procedure |
| :--- | :--- |
| Set a breakpoint on a step or <br> transition | $\bullet$ Right-click on the step or transition, and then click the required breakpoint command. |
| Remove a breakpoint from a step or <br> transition | $\bullet$ Right-click the step or transition, and then click Remove Breakpoint. |

## See also

Breakpoint on Step Activation on page 375
Breakpoint on Step Deactivation on page 376
Breakpoint on Transition on page 377
Transition Clearing Forcing on page 377
SFC Execution Behavior on page 371

## 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 PO 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.

This illustrates cycle execution when a breakpoint on step activation is encountered.


The behavior of setting a breakpoint on step activation is the same as setting a breakpoint on step deactivation of the previous step. Whether placing a breakpoint on step activation or on deactivation of the previous step, the target executes the break at the same moment.

## To set a breakpoint on step activation

- Right-click the step, and then click Set Breakpoint on Activation.


## See also

## Breakpoint on Step Deactivation on page 376

Breakpoint on Transition on page 377
Transition Clearing Forcing on page 377

## 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 PO 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.

This illustrates cycle execution when a breakpoint on step deactivation is encountered.


The behavior of setting a breakpoint on step activation is the same as setting a breakpoint on step deactivation of the previous step. Whether placing a breakpoint on step activation or on deactivation of the previous step, the target executes the break at the same moment. Both breakpoint on step activation and breakpoint on step deactivation are available to avoid setting multiple breakpoints as shown below.


Tip: On a given step, set either a breakpoint on step activation or a breakpoint on step deactivation.

## To set a breakpoint on step deactivation

- Right-click the step, and then click Set Breakpoint on Deactivation.


## See also

## Breakpoint on Step Activation on page 375

Breakpoint on Transition on page 377
Transition Clearing Forcing on page 377

## Breakpoint on Transition

Transition Clearing Forcing

When a transition becomes clearable (transition is valid, for example 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.

This illustrates cycles execution when a breakpoint on transition is encountered.


## To set a breakpoint on a transition

- Right-click the transition, and then click Set Breakpoint.


## See also

## Breakpoint on Step Activation on page 375

Breakpoint on Step Deactivation on page 376
Transition Clearing Forcing on page 377
Force the clearing of a transition while in simulation whether all previous steps are active or not. The tokens are moved and the actions are executed the same as with usual transition clearings.

Tokens of all predecessor steps are removed and tokens of all successor steps are created. All P0 actions linked to predecessor steps and P1-S - R - N actions linked to successor steps are executed.

This illustrates cycle execution when forcing transition clearing:


## To clear a transition

- Right-click the transition while in simulation mode, and then click Clear Transition.


## See also

Breakpoint on Step Activation on page 375
Breakpoint on Step Deactivation on page 376
Breakpoint on Transition on page 377
When working in Sequential Function Chart (SFC) programs, insert these elements. A program always has an initial step.

- Steps
- Transitions
- Sequence Controls
- Jumps to Steps

When inserting steps and transitions, these are assigned a default naming convention including numbering. For steps, the default naming is Sn where S indicates a step and n indicates the numbering for the step. For transitions, the default naming is Tn where T indicates a transition and n indicates the numbering for the transition. Rename steps and transitions. However, when renaming steps and transitions using the default naming convention and changing only the numbering, renumber these elements to a numbering scheme starting from top to bottom, then from left to right.


## To renumber steps and transitions

Renumbering ignores steps and transitions using a naming convention other than the default Sn for steps and Tn for transitions.

1. Open the SFC program for which to renumber the steps and transitions.
2. From the Tools menu, point to Multi-language Editor, and then click Renumber Steps and Transitions.

## See also

## Sequence Controls on page 382

Coding Action Blocks for Steps on page 386
Calling Functions from Transitions on page 393
Calling Function Blocks from Transitions on page 394
Sequential Function Chart (SFC) programs contain initial steps and steps. Initial steps express the initial situation of an SFC program. Whereas, steps are placed throughout an SFC program. An SFC program must contain at least one initial step. Initial steps are double bordered. Initial steps and steps are referenced by a name, written in their square symbol. This information is the level 1 of the step.

An initial step has a double bordered graphic symbol.
Initial Step


A step is represented by a single square.
Step


At run time, a token indicates that the step is active. For initial steps, a token is automatically placed in each when the program is started.


Steps have attributes. These can 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)
- Activity of a step is an attribute of a step which is activated by an SFC token.

For SFC function blocks, when reading a child active step or duration from a parent:

- 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)

| Task | Procedure |
| :--- | :--- |
| Insert an initial step | • From the Toolbox, drag the Initial Step element into the language container. |
| Insert a step | • From the Toolbox, drag the step element into the language container. |
| Renumber steps and transitions | Renumbering ignores steps and transitions using a naming convention other than the default Sn for steps and Tn for <br> transitions. <br> 1. Open the SFC program for which to renumber the steps and transitions. <br> 2. From the Tools menu, point to Multi-language Editor, and then click Renumber Steps and Transitions. |

## See also

Transitions on page 381
Sequence Controls on page 382
Jumps to Steps on page 385

Coding Action Blocks for Steps on page 386

## Transitions

Transitions are represented by small horizontal bars that cross the connection link. Each transition is referenced by a name, written next to the transition symbol. This information is called the level 1 of the transition.

When using single divergences, the first transitions following the divergence are set in a group to define their priority of execution. The priority of transitions is automatically assigned and follows the order of creation of the divergence branch. Specify a different priority for a transition. The possible priority values range from 1 to 255 . The higher value indicates the higher priority.


Tip: For transitions following divergences, set the priorities in the Properties window for transitions. Access by right-clicking the transition or the title bar of the transition's level 2.

## To insert a transition

1. From the Toolbox, drag the transition element into the language container.
2. To link the transition to an existing step, click the mouse with the cursor on the grid cell above or below the step.
3. Specify a different priority for a transition following a divergence:
a. Right-click the transition, and then click Properties.
b. In the Properties window, set the priority to the required value, and then click OK.
4. (optional) To renumber the steps and transitions, from the Tools menu, point to Multi-language Editor, and then click Renumber Steps and Transitions.

## See also

Steps on page 379
Sequence Controls on page 382
Jumps to Steps on page 385
Coding Conditions for Transitions on page 391

Sequence Controls

Selection Divergences

Sequence controls are divergences or convergences. These elements adjust automatically to the context of the Sequential Function Chart (SFC) diagram. The editor automatically inserts the type of sequence control required according to the elements at the insertion point. When adding a parallel element below a sequence control, the sequence control automatically branches out to the added element. When a sequence control is placed erroneously within a diagram, the editor displays it as red.

- Selection Divergences, a multiple link from a step to multiple transitions
- Selection Convergences, a multiple link from multiple transitions to a single step
- Simultaneous Divergences, a multiple link from a transition to multiple steps
- Simultaneous Convergences, a multiple link from multiple steps to a single transition

Divergences are multiple links from one SFC element (step or transition) to multiple SFC symbols. Convergences are multiple connections from more than one SFC symbol to one other symbol.

When inserting a sequence control, the type is determined logically according to the number of SFC elements of a same type (whether multiple) located initially above then below the control.

## To insert a sequence control

- From the Toolbox, drag the sequence control to the required location in the language container.


## See also

Selection Divergences on page 382
Selection Convergences on page 384
Simultaneous Convergences on page 384
Simultaneous Divergences on page 384
A selection divergence (OR) is a multiple link from one step to multiple transitions. The selection divergence enables an active token to pass into one of a number of branches.

Conditions attached to the different transitions at the beginning of a selection divergence are not implicitly exclusive. Exclusivity of transitions is defined by the priorities set to those transitions following the divergence.

Selection divergences are represented by single horizontal lines.


The first transitions following a single divergence are set in a group to define their priority of execution. The priority of transitions is automatically assigned and displayed on the left, in the order of creation of the divergence branch. A different priority for a transition in the properties may be selected. The possible priority values range from 1 to 255 where the higher value corresponds to the higher priority.

## Example

(* SFC Program with selection divergence and convergence *)


## See also

Selection Convergences on page 384
Simultaneous Convergences on page 384
Simultaneous Divergences on page 384

## Selection Convergences

A selection convergence (OR) is a multiple link from multiple transitions to a single step. Selection convergences are generally used to group branches which were started using selection divergences. Selection convergences are represented by single horizontal lines.


## See also

Selection Divergences on page 382
Simultaneous Convergences on page 384
Simultaneous Divergences on page 384

## Simultaneous

 ConvergencesA simultaneous convergence (AND) is a multiple link from multiple steps to a single transition. Simultaneous convergences are generally used to group branches which were started using simultaneous divergences. Simultaneous convergences are represented by double horizontal lines.


## See also

Simultaneous Divergences on page 384
Selection Divergences on page 382
Selection Convergences on page 384

## Simultaneous Divergences

A simultaneous divergence (AND) is a multiple link from one transition to multiple steps. A simultaneous divergence corresponds to parallel operations of a process.

Simultaneous divergences are represented by double horizontal lines.


## Example

(* SFC program with simultaneous divergence and convergence *)


## See also

## Simultaneous Convergences on page 384

Selection Divergences on page 382

## Selection Convergences on page 384

## Jumps to Steps

Jump symbols indicate a connection link from a transition to a step, without having to draw a connection line. Reference the jump symbol with the name of the destination step:

|  | Jump to Step S1 |
| :--- | :--- |
| S1 |  |

## Example

These charts are equivalent. The chart on the left uses links to return from the bottom to the top of the chart while the chart on the right uses jumps to return to the top of the chart.


## To insert a jump to a step

1. From the Toolbox, drag the jump element into the language container and place it directly below the existing transition.
2. In the Select Destination Step dialog box, select the required step and then click OK.

The jump is displayed in the language container.

## Coding Action Blocks for Steps

Action blocks are operations executed when a step is active. Steps can contain multiple action blocks of the same or different type. Add action blocks to the level 1 of a step. Depending on the action block type, program the level 2 for the block. The level 2 of an Sequential Function Chart (SFC) step is the detailed description of the actions executed during the step activity. Program level 2 code for an action block in a level 2 window, displayed to the right of the program organization unit (POU). Available action block types are:

- Boo where the action block name is automatically associated to a Boolean variable selected from the variable selector. Possible qualifiers are Action (N), Reset (R), and Set (S).
- LD to program an Ladder Diagram (LD) diagram in the level 2 window. Possible qualifiers are Action (N), Pulse on Deactivation Action (PO), and Pulse On Activation Action (P1).
- SFC where the action block name is automatically associated to the SFC child. Possible qualifiers are Action ( N ), Reset ( R ), and Set ( S ).
- ST to define Structured Text (ST) code in the level 2 window. Possible qualifiers are Action (N), Pulse on Deactivation Action (PO), and Pulse On Activation Action (P1).
Individual SFC steps execute in this order:

1. Step activation - beginning when the previous transition is cleared. During this period, defined action blocks execute in the order of appearance.
2. Step cycle - beginning when the step becomes active and ending when the step completes deactivation. During this period, defined action blocks execute in the order of appearance.
3. Step deactivation - ending when the following transition becomes active. During this period, defined action blocks other than Boolean (Boo) action blocks with the $N$ qualifier execute in the order of appearance. Boolean (Boo) action blocks execute after all other action blocks.

## To add action blocks to steps

1. Select the step for which to define operations.
2. Right-click the step, point to Add, then click one of these:

- Boo Action
- LD Action
- SFC Action
- ST Action

3. Specify the required properties for the action block from the Properties window by clicking the action block definition on the step.
a. To rename the ST or LD action block, type the required text in the Name field.

Tip: $\quad$ The names for Boo and SFC action blocks are automatically associated to their respective assignation (Boolean variable or SFC child).
b. To specify the qualifier for the action block, select the required type in the Qualifier field.
c. To include a comment, type the required text in the Comment field.
4. For a Boo action block, double-click the action block name, then from the Variable Selector, select the variable for use in the block.
5. For an ST or LD action block, access the level 2 for the block by doubleclicking the action block name on the step, then program the required level 2 operations in the level 2 window displayed to the right of the POU.
6. (optional) To rearrange the order of action blocks for a step, right-click the action block, and then click Move Up or Move Down.
7. (optional) To delete an action block on a step, right-click the action block, and then click Delete.

## Boolean Actions

Boolean (Boo) actions assign a Boolean variable to the activity of the Step. The Boolean variable can be an output or a memory variable. The variable is
assigned each time the step activity starts or stops. The operation for Boolean actions differs for the different qualifiers:

| 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 <br> TRUE. |
| R on a Boolean Variable | Resets the variable to FALSE when the step activity signal <br> becomes TRUE. |

The Boolean variable must be an OUTPUT or a MEMORY variable. The following SFC programming leads to the indicated behavior:


## See also

Coding Action Blocks for Steps on page 386

## 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 deactivation of the step: PO qualifier. Instructions are written using the Structured Text (ST) or Ladder Diagram (LD) syntax. This shows the results of a pulse action with the P1 qualifier:


## Example

In this Sequential Function Chart (SFC) program, step S1 is assigned an ST action named Edgelnit having the P1 qualifier and S2 is assigned an ST action
named EdgeCount having the P1 qualifier. The code for these actions is programmed in their respective level 2 window.


## See also

## Coding Action Blocks for Steps on page 386

Non-Stored Actions

A non-stored (normal) action is a list of Structured Text (ST) or Ladder Diagram (LD) instructions which are executed at each cycle during the whole active period of the step. Instructions are written according to the language syntax in use. Non-stored actions have the " N " qualifier.

This is the results of a non-stored action:
Step Activity

Execution


## Example



## See also

## Coding Action Blocks for Steps on page 386

## SFC Actions

A Sequential Function Chart (SFC) action is a child SFC sequence, started or terminated according to the change of the step activity signal. An SFC action can have the N (Non stored), $\mathrm{S}($ Set ), and R (Reset) qualifiers. This is the meaning of the action on an SFC child:

| Action | Description |
| :--- | :--- |
| $N$ on a child | Starts the child sequence when the step becomes active and terminates the child <br> sequence when the step becomes inactive. |
| S on a child | Starts the child sequence when the step becomes active. |
| $R$ on a child | Stops the child sequence when the step becomes active. |

The SFC sequence specified as an action must be a child SFC program of the program currently being edited.

## Example

(* SFC Program using SFC Action *)

The main SFC program is named Parent having one SFC child, called SeqMIx. The SFC programming of the parent SFC program is:


## Coding Conditions for Transitions

## See also

## Coding Action Blocks for Steps on page 386

Code conditions for clearing of transitions by programming these in the level 2 window. Only one condition can be attached to a transition. When changing the programming language of a transition, the code zone must be empty. When defining the properties of conditions, indicate a name, priority, comment (optional), and the programming language (type). The available programming languages for transitions are Ladder Diagram (LD) and Structured Text (ST).

When no expression is attached to the Transition, the default condition is TRUE.

## To code conditions for transitions

1. Select the transition for which to code a condition.
2. Right-click the transition, then choose Properties.
3. Specify the required properties for the transition from the Properties window.
a. To rename the transition, type the required text in the Name field.
b. To specify the type (programming language) for the transition condition, choose the required type in the Type field.
c. To include a comment, type the required text in the Comment field.
4. In the Level 2 window, program the required condition.

## See also

Conditions Programmed in ST on page 392
Conditions Programmed in LD on page 393
Coding Action Blocks for Steps on page 386

## Conditions Programmed in ST

Describe the condition attached to a transition with the Structured Text (ST) language. The complete expression must have Boolean type and may be terminated by a semi colon, according to this 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 *)


## See also

Conditions Programmed in LD on page 393

Conditions Programmed in LD

Describe the condition attached to a transition with the Ladder Diagram (LD) language. A diagram can have only one rung and one coil. The coil, to which a variable cannot be assigned, represents the activation status of the transition. Add blocks and contacts to define the logic.

## Example

(* SFC Program with LD programming for transitions *)


## See also

Conditions Programmed in ST on page 392
LD Elements on page 326
Any function written in Structured Text (ST), Ladder Diagram (LD), Function Block Diagram (FBD), or a "C" function can be called to evaluate the condition attached to a transition, according to this syntax in ST:

```
< function > ( ) ;
```

The value returned by the function must be Boolean and yield the resulting condition:

| Value | Condition |
| :--- | :--- |
| return value $=$ FALSE | condition is FALSE |
| return value $=$ TRUE | condition is TRUE |

## Example

(*SFC program with function call for transitions *)


## See also

## Calling Function Blocks from Transitions

Calling Function Blocks from Transitions on page 394
It is not recommended to call a function block in a Sequential Function Chart (SFC) condition for these reasons:

- A function block should be called at each cycle, typically in a cyclic program.
- An SFC condition is evaluated only when all of its preceding steps are active (not at each cycle)


## See also

Calling Functions from Transitions on page 393
The following keyboard shortcuts are available for use with the Sequential Function Chart (SFC) language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while <br> debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not <br> available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while <br> debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+0 | Inserts an initial step (not available while debugging) |
| Ctrl+1 | Inserts a step (not available while debugging) |
| Ctrl+2 | Inserts a transition (not available while debugging) |
| Ctrl+3 | Inserts a sequence control (not available while debugging) |
| Ctrl+4 | Inserts a jump (not available while debugging) |
| Ctrl+Shift+R | Renumbers the steps and transitions using the default naming <br> convention (Sn and Tn) |
| Ctrl+Page Up | Jumps to the top edge of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Ctrl+Up Arrow | Slowly scrolls up |
| Ctrl+Down Arrow | Slowly scrolls down |
| Ctrl+Left Arrow | Slowly scrolls left |
| Ctrl+Right Arrow | Slowly scrolls right |
| Up Arrow | Moves up the grid or from one selected element to the next |
| Down Arrow | Moves down the grid or from one selected element to the next |
| Left Arrow | Moves to the left across the grid or from one selected element to the <br> next |
| Right Arrow | Moves to the right across the grid or from one selected element to <br> the next |
| Delete | Removes the selected elements (not available while debugging) |

## See also

SFC Language on page 370

## Language Reference

## Programs

The language reference includes information about the usage and limitations of various project elements and other aspects:

- Programs
- Functions
- Function Blocks
- Execution Rules
- Reserved Keywords
- Variables
- Directly Represented Variables
- Defined Words
- Data Types


## See also

Execution Rules on page 398
Reserved Keywords on page 399
Directly Represented Variables on page 400
Data Types on page 402
Programs, also known as program organization units (POUs), are logical programming units describing operations between variables of a process. Programs describe either sequential or cyclic operations. Cyclic programs are executed at each target system cycle. Sequential programs, representing sequential operations, are grouped together. The execution of sequential programs has a dynamic behavior.
Programs before and after sequential programs describe cyclic operations.
These are cyclic programs which are not time-dependent. 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 their respective dynamic behavior.

- Begin: Cyclic operations (FDB, LD, ST)
- Sequential: Sequential operations (SFC, SFC child)
- End: Cyclic operations (FDB, LD, ST)

Programs located at the beginning of a cycle (before sequential programs) typically describe preliminary operations on input controllers to build high level filtered variables. Sequential programs frequently use these variables. Programs located at the end of the cycle (after sequential programs) typically describe security operations on the variables operated on by sequential programs, before sending values to output controllers.
Programs are described using the available graphic or literal languages. Specify the programming language when creating a program; the programming language cannot be changed for an existing program.

POUs defined as programs are executed on the target system respecting the order shown in the Programs section.

Respect the hierarchy of programs. Programs are linked together in a hierarchical tree. Those placed at the top of the hierarchy are activated by the system. Child-programs (lower level of the hierarchy) are activated by their parent.

Projects can contain up to 65536 programs.
For programs within a project and the global library, names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a program name must be a letter or digit; program names cannot end with an underscore character. Reserved words, defined words, or data types such as, elementary, structures, or arrays, are not valid names. The same type of elements within a scope must have unique names.

## See also

## Functions on page 396

Function Blocks on page 397
Data Types on page 402

## Functions

Functions are program organization units (POUs) having one or more 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 meaning that local data is not stored and is usually lost from one call to the other.

The execution of a function is driven by its parent program. Therefore, the execution of the parent program is suspended until the function ends:


Any POU of any section can call one or more functions. A function can have local variables.

Recursivity is not supported during function calls. When a function of the Functions section is called by itself or one of its called functions, a run-time error occurs. Furthermore, functions do not store the local values of their local variables. Since functions are not instantiated, these 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 (input) parameters or return (output) parameter. A function has one return parameter. The value of the return parameter for a function differs for the various programming languages.

Functions within a project and the global library must have unique names within a scope. Function names and function parameter names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a function name must be a letter or digit; function names cannot end with an underscore character. Reserved words, defined words, or data types, such as elementary, structures, or arrays are not valid names.

## See also

Reserved Keywords on page 399

Function Blocks

Function blocks are program organization units (POUs) with multiple input and output parameters. These are instantiated meaning local variables of a function block are copied for each instance. When calling a function block in a program, call the instance of the block where the same code is called but the data used is that which has been allocated to the instance. The values of the variables of an instance are stored from one cycle to the other.

Any POU in a project can call function blocks. Function blocks can call functions or other function blocks. The order of appearance of function blocks within their section is not important.

Define the interface of a function block with a type and a unique name for each of its calling (input) parameters or return (output) parameters. Function blocks can have more than one output parameter. The value of a return parameter for a function block differs for the various programming languages.

Move or copy all function blocks to the Programs section and all but the SFC function block to the Functions section. 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.

Function blocks within a project and the global library must have unique names within a scope. Function block names and function block parameter names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscores. The last character for a function block name must be a letter or digit; function block names cannot end with an underscore character. Reserved words, defined words, or data types, such as elementary, structures, or arrays are not valid names.

Each instance of IEC 61131-3 or 'C' written function corresponds to data stored in AADvance virtual machine real time database. Do not add new automatic instances of IEC 61131-3 function blocks or of standard 'C' function blocks with or without initial values. Work with function blocks declared in the Dictionary to enable online changes.

User 'C' Function Blocks cannot be added.

## See also

## Functions on page 396

Programs on page 395
Data Types on page 402

## Execution Rules

Execution of an application for a controller follows these main steps within a loop:

1. Read input
2. Consume bound variables
3. Execute target independent code (TIC)
4. Process ISaGRAF ${ }^{\otimes}$ eXchange Layer (IXL) messages
5. Produce bound variables
6. Write outputs
7. Save retain variables
8. Process IXL messages
9. Execute other controller tasks including diagnostics, synchronization, and cycle timing
[^9]- When bindings are defined, variables consumed by a controller update after inputs are scanned and the variables produced for other controllers are sent before updating outputs.
- When a cycle time is specified, a controller waits until this time has elapsed before starting the execution of a new cycle. The program organization units (POUs) execution time varies depending on the size of the application. When a cycle exceeds the specified time, the loop continues to execute the cycle but sets an overrun flag. In such a case, the application no longer runs in real time.
- When a cycle time is not specified, a controller performs all programs then restarts a new cycle without waiting.


## See also

Forcing the Values of Variables on page 139
Modifying Cycle Timing on page 136

Reserved Keywords
Reserved keywords are unavailable for use as names of program organization units (POUs) or variables. Also, all keywords beginning with the underscore ('_') character are internal keywords and must not be used in textual instructions.

| Keywords |  |
| :---: | :---: |
| - | _AND, _CALL, _CALL_IEC_SFC_-FB, _END,_GOTO, _IF, _NOT, _PUSH_PAR, _OR, _POP_CSTK, _PUSH_CSTK,_RET,_STEP,_XOR |
| 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, BINDING, 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_RESOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHLLE, ENo, EQ, EXIT, EXP, EXPT, |
| F | FALSE, FIND, FOR, FUNCTION, |
| G | GE, GFREEZE, GKILL, GLOBALVARIABLE, GRST, GSTART, GSTATUS, GT, |
| H | HEADER, |
| 1 | IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME, 10, |
| 」 | 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, |
| 0 | 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, RESOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR, |
| S | S, SEL, SET, SHL, SHR, SIN, SINT, SORT, 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_TOVERFLOW, 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, UNTIL, 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 |

The scope of variables can be local to a program organization unit (POU) or global to a controller. Local variables are available for use within one POU only. Global variables are available for use within any POU of the controller.

Variable names are limited to 128 characters beginning with a letter or single underscore character followed by letters, digits, and single underscore characters. The last character for a variable name must be a letter or digit; variable names cannot end with an underscore character. Names cannot be reserved words, defined words, or data types (elementary, structures, or arrays). Names must be unique for the same type of elements within a scope.

Tip: Avoid using the same name for a local and global variable. When a local and global variable share the same name, only the global variable is used in the built project.

All variables have an attribute and direction. Variables can have one of the following attributes:

| Read/Write | 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 |

Variables can have one of the following directions. However, some variables such as time cannot have the input or output direction. Refer to the corresponding data type for restrictions.

| Var | Internal variable updated by the programs |
| :--- | :--- |
| Varlnput | Variable connected to an input controller (refreshed by the system) |
| VarOutput | Variable connected to an output controller |
| VarGlobal | System variable used to access diagnostic information |
| VarDirectlyRepresented | $1 / 0$ channel of an I/0 device |

Initial values can be assigned to variables of elementary IEC 61131-3 data types. Default initial values are 0 or FALSE. An initial value is the value of a variable when a target starts its first cycle.

## See also

## Data Types on page 402

## Directly Represented Variables

The system enables the use of directly represented variables in the source of programs to represent an unwired channel. Unwired channels are those not linked to a declared I/O variable. The identifier of a directly represented variable always begins with the "\%" character.

The naming conventions of a directly represented variable for a channel of a single $\mathrm{I} / \mathrm{O}$ device. " s " is the slot number of the $\mathrm{I} / \mathrm{O}$ device. " c " is the number of the Channel:

Single device naming conventions

| Convention | Description |
| :--- | :--- |
| \%IXS.C | unwired Channel of a Boolean input I/O device |
| \%IBs.C | unwired Channel of a Short integer, Unsigned short integer, or <br> BYTE input I/O device |
| \%IWs.C | unwired Channel of an Integer, Unsigned integer, or WORD input <br> I/O device |
| \%IDs.C | unwired Channel of a Double integer, Unsigned double integer, <br> Double word, or DATE input I/O device |
| \%ILS.C | unwired Channel of a Long integer, Unsigned long integer, Long <br> word, or Long real input I/O device |
| \%IRs.C | unwired Channel of a Real input I/O device |
| \%ITs.C | unwired Channel of a Time input I/O device |
| \%ISs.C | unwired Channel of a String input I/O device |
| \%OXs.C | unwired Channel of a Boolean output I/O device |
| \%OBs.C | unwired Channel of a Short Integer, Unsigned short integer, or <br> BYTE output I/O device |
| \%OWs.C | unwired Channel of an Integer, Unsigned integer, or WORD <br> output I/O device |

## Single device naming conventions

| \%ODs.C | unwired Channel of a Double integer, Unsigned double integer, <br> Double word, or DATE output I/O device |
| :--- | :--- |
| \%OLs.C | unwired Channel of a Long integer, Unsigned long integer, Long <br> word, or Long real output I/O device |
| \%QRs.C | unwired Channel of a Real output I/O device |
| \%OTS.C | unwired Channel of a Time output I/O device |
| \%OSS.C | unwired 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:

| Complex device naming conventions |  |
| :---: | :---: |
| Convention | Description |
| \%\|Xs.b.c | unwired Channel of a Boolean input I/O device |
| \%\|Bs.b.c | unwired Channel of a Short Integer, Unsigned short integer, or BYTE input $1 / 0$ device |
| \%IWs.b.c | unwired Channel of an Integer, Unsigned integer, or WORD input 1/0 device |
| \%\|Ds.b.c | unwired Channel of a Double integer, Unsigned double integer, Double word, or DATE input I/O device |
| \%ILs.b.c | unwired Channel of a Long integer, Unsigned long integer, Long word, or Long real input I/0 device |
| \%IRs.b.c | unwired Channel of an Real input I/O device |
| \%ITs.b.c | unwired Channel of a Time input 1/0 device |
| \%ISs.b.c | unwired Channel of a String input I/O device |
| \%0Xs.b.c | unwired Channel of a Boolean output I/O device |
| \%0Bs.b.c | unwired Channel of a Short Integer, Unsigned short integer, or BYTE output I/O device |
| \%OWs.b.c | unwired Channel of an Integer, Unsigned integer, or WORD output I/O device |
| \%0Ds.b.c | unwired Channel of a Double integer, Unsigned double integer, Double word, or DATE output I/O device |
| \% | unwired Channel of a Long integer, Unsigned long integer, Long word, or Long real output I/O device |
| \%0Rs.b.c | unwired Channel of a Real output I/O device |
| \%OTs.b.c | unwired Channel of a Time output I/O device |
| \%OSs.b.c | unwired Channel of a String output 1/0 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)
```


## See also

Reserved Keywords on page 399

## Defined Words

AADvance projects support the use of identifier names, called defined words. Defined words are replaced by the variables and expressions these represent
when building. Defined words have a common scope making these available for use in any program organization unit (POU) of any controller of a project.

For POUs, a defined word replaces literal expressions, boolean expressions, reserved keywords or complex ST expressions.

These are examples of defined words:


When an equivalence is defined, its identifier is available anywhere in an ST program to replace the attached expression. This ST programming example uses defined words:

```
If OK Then
angle := PI / 2.0;
isdone := YES;
End_if;
```

Defined word names are limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for a defined word name must be a letter or digit; defined word names cannot end with an underscore character. Names cannot be reserved words, defined words, data types, structures, or arrays. The same type of elements within a scope must have unique names. The definition of a defined word cannot contain a defined word.

> Tip: Defined words must have unique names. Create defined words in a library for use in a project. Build errors occur when a project and library contain the same defined word.

## See also

Defined Words Grid on page 163
Any literal, expression, or variable used in a program organization unit (POU) (written in any language) must be characterized by a data type. Data type coherence must be followed in graphic operations and literal statements. Data types are one of these types:

- Elementary IEC 61131-3 Types
- Derived Types: Arrays
- Derived Types: Structures

Program objects using these elementary IEC 61131-3 types:

- BOOL: logic (true or false) value
- SINT: short integer value (8 bit)
- USINT: unsigned short integer value (8 bit)
- BYTE: byte value (8 bit)
- INT: single integer value (16 bit)
- UINT: unsigned single integer value (16 bit)
- WORD: word value (16 bit)
- DINT: double integer value (32 bit)
- UDINT: unsigned double integer value (32 bit)
- DWORD: double word value (32 bit)
- LINT: long integer value (64 bit)
- ULINT: unsigned long integer value (64 bit)
- LWORD: long word value (64 bit)
- REAL: real (floating) value (32 bit)
- LREAL: long real (floating) value (64 bit)
- TIME: time values less than 49d17h2m47s295ms; these value types cannot store dates (32 bit)
- DATE: date values (32 bit)
- STRING: character string having a defined size, representing the maximum number of characters the string can contain.

Define new user types based on these elementary IEC 61131-3 types. Define arrays or structures using elementary IEC 61131-3 types, arrays, or other user types.

Assign a dimension to define an array when creating a variable. Example: the MyVar variable of type BOOL with a defined dimension:

```
[1..10]
```

FOR i $=1$ TO 10 DO
MyVar[i] := FALSE;
END_FOR;

Boolean variables (BOOL) take one of the Boolean values: TRUE or FALSE. Boolean variables are typically used in Boolean expressions.

For Boolean literal expressions, targets evaluate all parts of such expressions. Whereas, the IEC 61131-3 standard states that Boolean expressions may be evaluated only to the extent necessary to determine the resultant value. In the following example according to the IEC 61131-3 standard, if $B$ is zero then the first expression $(B<>0)$ is false and the second expression $(A / B>0)$ is not performed.

```
if ((B <> 0) and (A/B > 0)) then
GREATER := true;
else
GREATER := false;
end_if;
```

Boolean literal expressions are:

- TRUE is equivalent to the integer value 1


## FALSE is equivalent to the integer value $\mathbf{0}$ See also

Elementary IEC 61131-3 Types on page 402

## Short Integer Data Type

## Unsigned Short Integer or BYTE Data Type

Short Integer (SINT) variables are 8-bit signed integers from -128 to +127 .
Access a bit of a short integer variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is a short Integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 7 .
Short integer literal expressions represent signed integer ( 8 bit ) values:
from -128 to +127
Express short integer constants with one of these Bases. Short integer constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | 19 |
| HEXADECIMAL | "16\#" | $16 \# A 1$ |
| OCTAL | "8\#" | $8 \# 27$ |
| BINARY | "2\#" | 2\#0101_0101 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Unsigned Short Integer or BYTE Data Type on page 404
Elementary IEC 61131-3 Types on page 402
Unsigned Short Integer or BYTE variables are 8-bit unsigned integers from 0 to 255.

Access a bit of an unsigned short integer or BYTE variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is an unsigned short integer or BYTE.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 7
Unsigned short integer and BYTE literal expressions represent unsigned integer (8 bit) values:
from 0 to 255
Express short integer and BYTE constants with one of these Bases. These constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | 19 |
| HEXADECIMAL | "16\#" | $16 \# A 1$ |
| OCTAL | "8\#" | $8 \# 27$ |
| BINARY | "2\#" | $2 \# 0101 \_0101$ |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

## Short Integer Data Type on page 404

Elementary IEC 61131-3 Types on page 402

## Integer Data Type

Integer (INT) variables are 16-bit signed integers from -32768 to 32767.
Access a bit of a bit of an integer variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is an Integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 15 . Integer literal expressions represent signed integer (16 bit) values:
from -32768 to 32767
Express integer constants with one of these Bases. Integer constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | -260 |
| HEXADECIMAL | "16\#" | $16 \# F E F C$ |
| OCTAL | "8\#" | $8 \# 177374$ |
| BINARY | "2\#" | 2\#0101_0101_0101_0101 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Unsigned Integer or Word Data Type on page 405
Elementary IEC 61131-3 Types on page 402

## Unsigned Integer or Word Data Type

Unsigned Integer or WORD variables are 16-bit unsigned integers from 0 to 65535.

Access a bit of an unsigned integer or WORD variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is an unsigned integer or WORD.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 15.
Unsigned integer and WORD literal expressions represent unsigned integer (16 bit) values:

Express unsigned integer and WORD constants with one of these Bases. These constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | +33000 |
| HEXADECIMAL | "16\#" | $16 \# 80 E 8$ |
| OCTAL | "8\#" | $8 \# 100350$ |
| BINARY | "2\#" | $2 \# 0101 \_0101 \_0101 \_0101$ |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Integer Data Type on page 405
Elementary IEC 61131-3 Types on page 402

## Double Integer Data Type

## Unsigned Double Integer or Double Word Data Type

If MyVar is an Integer.
MyVar. i is a Boolean. " i " must be a constant value from 0 to 31 .
Double integer literal expressions represent signed double integer (32 bit) values:
from - 2147483648 to +2147483647
Express double integer constants with one of these Bases. Double integer constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16\#" | $16 \# 1$ A2B3C4D |
| OCTAL | "8\#" | $8 \# 1756402$ |
| BINARY | "2\#" | 2\#1101_0001_0101_1101_0001_0010_1011_1001 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Unsigned Double Integer or Double Word Data Type on page 406
Elementary IEC 61131-3 Types on page 402
Double Integer variables are 32-bit signed integers from -2147483648 to +2147483647.

Access a bit of a double integer variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
fMyVar is an integer.

Unsigned Double Integer or Double Word variables are 32-bit unsigned integers from 0 to 4294967295.

Access a bit of an unsigned double integer or double word variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is an unsigned double integer or double word.
MyVar. i is a Boolean. "i" must be a constant value from 0 to 31 .
Unsigned double integer and Double Word literal expressions represent unsigned double integer ( 32 bit) values:
from 0 to 4294967295
Express double integer and double word constants with one of these Bases. Double integer and double word constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | +908 |
| HEXADECIMAL | "16\#" | $16 \# 1$ 1A2B3C4D |
| OCTAL | "8\#" | 8\#1756402 |
| BINARY | "2\#" | 2\#1101_0001_0101_1101_0001_0010_1011_1001 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Double Integer Data Type on page 406

## Elementary IEC 61131-3 Types on page 402

Long Integer variables are 64-bit signed integers from -
9223372036854775808 to 9223372036854775807.
Access a bit of a long integer variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is a long integer.
MyVar.i is a Boolean. " i " must be a constant value from 0 to 63.
Long integer literal expressions represent signed long integer ( 64 bit ) values:
from -9223372036854775808 to 9223372036854775807
Express long integer constants with one of these Bases. Long integer constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
| :--- | :--- | :--- |
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16\#" | $16 \# 1 A 2 B 3 C 4 D$ |
| OCTAL | "8\#" | $8 \# 1756402$ |
| BINARY | "2\#" | 2\#1101_0001_0101_1101_0001_0010_1011_1001_1101_0001_0101_11 <br> 01_0001_0010_101_1001 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Unsigned Long Integer or Long Word Data Type on page 408
Elementary IEC 61131-3 Types on page 402

## Unsigned Long Integer or Long Word Data Type

Unsigned Long Integer or Long Word variables are 64-bit unsigned integers from 0 to 18446744073709551615.

Access a bit of an unsigned long integer or long word variable, array, structure, or the output of a function block instance using this syntax:

MyVar.i
If MyVar is an unsigned long integer or long word.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 63.
Unsigned long integer and long word literal expressions represent unsigned long integer ( 64 bit ) values:
from 0 to 18446744073709551615
Express unsigned long integer and long word constants with one of these Bases. Long integer and long word 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_1101_0001_0101_11 <br> 01_0001_0010_1011_1001 |

Use the underscore character ('_') to separate groups of digits. This character has no significance other than to improve literal expression readability.

## See also

Long Integer Data Type on page 407
Elementary IEC 61131-3 Types on page 402
Real variables are standard IEEE 32-bit floating values (single precision).
1 sign bit + 23 mantissa bits +8 exponent bits
A real variable has six significant digits. For larger values, the maximum possible value is $\pm 3.402823466 \mathrm{E}+38$ while for smaller values, the minimum possible value is $\pm 1.175494351 \mathrm{E}-38$. Therefore, values greater than $\pm$ $3.402823466 \mathrm{E}+38$ and greater than 0.0 but less than $\pm 1.175494351 \mathrm{E}-38$ are
not supported. This example shows the value ranges including 0.0 that are supported for real variables:


Real literal expressions can be written with either Decimal or Scientific representation. The decimal point ('.') separates the Integer and Decimal parts. The decimal point differentiates a Real literal expression from an Integer expression. 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.

\section*{Example <br> | 3.14159 | $-1.0 \mathrm{E}+12$ |
| :--- | :--- |
| +1.0 | $1.0 \mathrm{E}-15$ |
| -789.56 | $+1.0 \mathrm{E}-37$ |}

The expression "123" does not represent a Real literal expression. Its correct real representation is "123.0".

## See also

Long Real Data Type on page 409
Elementary IEC 61131-3 Types on page 402

Long Real Data Type

Long Real (LREAL) variables are standard IEEE 64-bit floating values (double precision).

1 sign bit + 52 mantissa bits + 11 exponent bits
A long real variable has 15 significant digits. For larger values, the maximum possible value is $\pm 1.7976931348623158 \mathrm{e}+308$ while for smaller values, the minimum possible value is $\pm 2.22507385850721 \mathrm{E}-308$. Therefore, values greater than $\pm 1.7976931348623158 \mathrm{e}+308$ and greater than 0.0 but less than $\pm 2.22507385850721 \mathrm{E}-308$ are not supported. The following example shows the value ranges including 0.0 that are supported for long real variables:


Long real literal values can be written with either Decimal or Scientific representation. The decimal point ('.') separates the Integer and Decimal parts. The decimal point differentiates a Real literal value from an Integer expression. The scientific representation uses the letter 'E' to separate the mantissa part and the exponent. The range of a real scientific expression must be a signed integer value from $1.7 \mathrm{E}-308$ to $1.7 \mathrm{E}+308$. A long real variable has 15 significant digits.

Tip: When floating point values are considered REAL values instead of LREAL values, use the ANY_TO_LREAL operator to force REAL values into LREAL values.

Example

| 3.14159 | $-1.0 \mathrm{E}+12$ |
| :--- | :--- |
| +1.0 | $1.0 \mathrm{E}-15$ |
| -789.56 | $+1.0 \mathrm{E}-37$ |

The value "123" does not represent a long real literal value. Its correct real representation is "123.0".

## See also

## Real Data Type on page 408

## Elementary IEC 61131-3 Types on page 402

## Time Data Type

Use time variables in time expressions. A time value represents values from 0 to 1193 h 2 m 47 s 294 ms . Time variables are stored in 32 -bit words. The internal representation is a positive number of milliseconds. Use time variables with timer function blocks such as TOF and TON.

Time literal expressions represent time values from 0 to 1193 h 2 m 47 s 294 ms . The lowest allowed unit is a millisecond. Literal expressions use these standard time units:

| Time unit | Description |
| :--- | :--- |
| Hours | The "h" letter must follow the number of hours |
| Minutes | The "m" letter must follow the number of minutes |
| Seconds | The "s" letter must follow the number of seconds |
| Milliseconds | The "ms" letter must follow the number of milliseconds |

The time literal expression must begin with "T\#" or "TIME\#" prefix. Prefixes and unit letters are not case sensitive. Some units may not appear.

When the TIME value is equal to -1 (as a DINT value), the value is considered as overflow and invalid. For example:

```
IF ANY_TO_DINT(TIME1) = -1 then
(* Handle overflow *)
END_IF;
```


## Example

```
T#1H450MS, 1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes
```

This Structured Text (ST) code gets the current time for use in the clock portion of a date variable:

```
clock := any_to_time(CURRENT_ISA_DATE()); (* Gets the
current time **)
```


## See also

TOF on page 480
TON on page 480
Date Data Type on page 411
Elementary IEC 61131-3 Types on page 402

## Date Data Type

## String Data Type

Date variables have date values and are typically used in Date expressions. A Date value ranges from 1970-01-01 to 2038-01-18. Date variables are stored using the 32 bit ISO 'C' time_t data type. The internal representation is a positive number of seconds since 1970-01-01 at midnight GMT.

Date literal expressions represent date values in the year-month-day format, separated by hyphens. Possible date literal expressions range from DATE\#1970-01-01 to DATE\#2038-01-18 GMT.

The date literal expression must begin with "D\#" or "DATE\#" prefix. Prefixes and unit letters are not case sensitive.

## Example

D\#2005-02-20
date\#2005-02-20

## See also

Time Data Type on page 410
Elementary IEC 61131-3 Types on page 402
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 length is limited to 255 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. When declaring string variables, the maximum number of characters is defined in the String Size column of the Dictionary or Variable Selector.

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.

String literal values represent character strings. Precede and follow characters with single quote (') characters. For example:
'THIS IS A MESSAGE'

| Important: | Express a string literal expression on one line of the program source <br> code. When placing single quote (') characters within a string literal, <br> precede these characters with the dollar (\$) character. In the <br> following string literal, the dollar character precedes the single <br> quote character. |
| :--- | :--- |

'THIS IS \$' A MESSAGE'
The length of a string literal cannot exceed 255 characters, including spaces.
Empty string literal values are represented by two single quote (') characters, 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 literal values to represent a non-printable character:

| Sequence | Meaning | ASCII (hex) | Example |
| :---: | :---: | :---: | :---: |
| \$\$ | '\$' character | 16\#24 | 'I paid \$\$5 for this' |
| \$ | apostrophe | 16\#27 | 'Enter \$'YS' for YES' |
| \$L | line feed | 16\#0a | 'next \$L line' |
| \$R | carriage return | 16\#0d | 'Ilo \$R He' |
| \$N | new line | 16\#0dOa | 'This is a lineSN' |
| \$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\$4' |

$\left(^{*}\right)$ " hh " is the hexadecimal value of the ASCII code for the expressed character.

## See also

## Variables Grid on page 164

Elementary IEC 61131-3 Types on page 402

## Derived Types: Arrays

Define arrays of standard IEC 61131-3 types or derived types. An array has one or more dimensions. After defining an array, create a variable with this type and a structure can have a field with this type. Array dimensions are positive DINT literal expressions and array indexes are DINT literal expressions or variables.

Array names are limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for an array name must be a letter or digit; array names cannot end with an underscore character. Reserved words, defined words, or data types, such as elementary, structures, or arrays are not valid names. The same type of elements within a scope must have unique names.

Tip: User-defined data types (arrays and structures) must have unique names. Build errors may occur if a library and project use the same name with different definitions for a data type. Replace BYTE, WORD, DWORD, and LWORD data types with USINT, UINT, UDINT, and ULINT types.

## 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. Two dimensions are defined:
[1..10,1..3]
MyVar2 is of type MyArrayType2
MyVar2[1,2] := 100;
3. Array of an array:

MyVar3 is an array of MyArrayType; Dimensions are defined as:
[1..3]
FORI:= 1 TO 3 DO
FOR J:= 1 TO 10 DO
MyVar3[I][J] := FALSE;
END_FOR;
END_FOR;

## Derived Types: Structures

Define structures using elementary IEC 61131-3 types or derived types. A structure consists of sub-entries called fields. After defining a structure, create a variable with this type.

Structure names are limited to 128 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for a structure name must be a letter or digit; structure names cannot end with an underscore character. Names cannot be reserved words, defined words, or data types (elementary, structures, or arrays). Names must be unique for the same type of elements within a scope.

Tip: User-defined data types (arrays and structures) must have unique names. Build errors may occur if a library and project use the same name with different definitions for a data type. Replace BYTE, WORD, DWORD, and LWORD data types with USINT, UINT, UDINT, and ULINT types.

## 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 *)

## Operators

These are standard operators of the IEC 61131-3 languages:

| Operators | Description |
| :---: | :---: |
| 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 |
| Comparators |  |
| 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 |  |
| Equal | Converts to Boolean |
| Equal | Converts to Short integer |
| Equal | Converts to Unsigned short integer |
| Equal | Converts to BYTE |
| Equal | Converts to Integer |
| ANY_TO_UINT | Converts to Unsigned integer |
| ANY_TO_WORD | Converts to WORD |
| ANY_TO_DINT | Converts to Double integer |
| Equal | Converts to Unsigned double integer |
| Equal | Converts to Double WORD |
| Equal | Converts to Long integer |
| Equal | Converts to Unsigned long integer |
| Equal | Converts to Long WORD |
| Equal | Converts to Real |
| Equal | Converts to Long real |
| Equal | Converts to Time |
| Equal | Converts to Date |
| Equal | Converts to String |

Directly links the input to the output. When used with a Boolean negation, inverts the state of the line connected to the output.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| IN | BOOL - SINT - USINT - <br> BYTE - INT - UINT - <br>  <br> WORD - DINT - UDINT - <br>  <br> DWORD - LINT - ULINT - <br>  <br>  <br>  |  |  |

## Arguments

| 0 | BOOL - SINT - USINT - | il and ol must have the same format |
| :--- | :--- | :--- |
| BYTE - INT - UINT - |  |  |
|  | WORD - DINT - UDINT - |  |
|  | DWORD - LINT - ULINT - |  |
|  | LWORD - REAL - LREAL - |  |

## Example

(* FBD example with assignment Operators *)

(* ST equivalence: *)

```
ao23 := ai10;
bo100 := NOT (bi1 AND bi2);
```


## Addition

Adds 2 to 127 integer, real, TIME, or STRING variables.


Arguments

| (inputs) | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - <br> STRING | Can be of any integer, real, TIME, or STRING format. All inputs <br> must have the same format. From 2 to 127 inputs. |
| :--- | :--- | :--- |
| 01 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - - LWORD - REAL - <br> LREAL - TIME - <br> STRING |  |

## Example

(* FBD example with Addition Operators *)

(* ST equivalence: *)

```
ao10 := ai101 + ai102;
ao5 := (ai51 + ai52) + ai53;
```


## See also

## Subtraction on page 437

Boolean AND between 2 and 127 Boolean variables.


Arguments

| (inputs) | BOOL | From 2 to 127 inputs. |
| :--- | :--- | :--- |
| 01 | BOOL | Boolean AND of the input variables. |

## Example

(* FBD example with "AND" Operators *)

(* ST equivalence: *)

```
bo10 := bi101 AND NOT (bi102);
    bo5 := (bi51 AND bi52) AND bi53;
```


## See also

OR on page 436

ANY_TO_BOOL

ANY_TO_BYTE

## Converts variables to Boolean variables



| arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - <br> WORD - DINT - UDINT - DWORD - LINT - ULINT <br> - LWORD - REAL - LREAL - TIME - DATE - <br> STRING | Any value |  |
| 01 | BOOL | TRUE for non-zero numerical value <br> FALSE for zero numerical value <br> TRUE for 'TRUE' string <br> FALSE for 'FALSE' string |  |

## 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 <br> $('$ FALSE' $) ;$ | $(*$ mres is FALSE *) |

## Converts variables to 8-bit BYTE variables



| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - <br> WORD - DINT - UDINT - DWORD - LINT - ULINT - <br> LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | BYTE | 0 if il is FALSE / 1 if il is TRUE <br> number of milliseconds for a timer <br> integer part for real <br> decimal number represented by a string |  |

## Example

(* FBD example with "Convert to BYTE" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_BYTE (true); | $(*$ bres is 1 *) |
| :--- | :--- |
| tres $:=$ ANY_TO_BYTE <br> (t\#0s46ms); | $(*$ tres is 46 *) |
| mres $:=$ ANY_TO_BOOL <br> $\left(' 0198^{\prime}\right) ; ~$ | $(*$ mres is $198 *)$ |

ANY_TO_DATE
Converts variables to DATE variables. A 32-bit variable, providing the number of seconds since Jan 1, 1970, based on the time_t data type.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - <br> WORD - DINT - UDINT - DWORD - LINT - ULINT - <br> LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | DATE | date represented by il. A value of - 1 indicates an <br> invalid date. |  |

## Example

(* FBD example with "Convert to DATE" Operators *)

(* ST Equivalence: *)

| ares $:=$ ANY_TO_DATE | $(*$ ares $:=$ d\#2005-02- |
| :--- | :--- |
| $(1109110199) ;$ | 22 *) $\quad(*$ rres $:=$ d\#2005-02-22 |
| rres $:=$ ANY_TO_DATE | *) |
| $(1109110199.3) ;$ |  |


| ANY_TP_DINT |  |  |
| :---: | :---: | :---: |
| -11 | $\vdots$ | 01 |
| Any | DINT | Dimt |


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - <br> WORD - DINT - UDINT - DWORD - LINT - ULINT - <br> LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | DINT | 0 if i is FALSE $/ 1$ if i 1 is TRUE <br> number of milliseconds for a timer integer part for <br> real decimal number represented by a string |  |

## Example

(* FBD example with "Convert to Double Integer" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_DINT (true); |
| :--- |
| tres $:=$ (* bry_TO_DINT |
| (t\#1s46ms); |
| mres $:=$ ANY_TO_DINT ('0198'); |

## Converts variables to 32-bit double WORD variables



| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL <br> - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | DWORD | 0 if $i 1$ is FALSE / 1 if $i 1$ is TRUE <br> number of milliseconds for a timer integer part <br> for real decimal number represented by a string |  |

## Example

(* FBD example with "Convert to Double WORD" Operators *)

(* ST Equivalence: *)
bres $:=$ ANY_TO_DWORD (true); (* bres is 1 *) $^{\text {b }}$

## ANY_TO_LREAL

## Example

(* FBD example with "Convert to Long REAL" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_LREAL (true); |
| :--- |
| tres $:=$ ANY_TO_LREAL |
| (t\#1s46ms); |
| ares $:=$ ANY_TO_LREAL ('198'); |

## Converts variables to 16 -bit integer variables



## Arguments

| $i 1$ | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - | Any value |
| :--- | :--- | :--- |
|  | DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL |  |
|  | - LREAL - TIME - DATE - STRING |  |


| 01 | INT | 0 if il is FALSE / 1 if il is TRUE <br> number of milliseconds for a timer <br> integer part for real <br> decimal number represented by a string |
| :--- | :--- | :--- |

## Example

(* FBD example with "Convert to Integer" Operators *)

(*ST Equivalence: *)

| bres $:=$ ANY_TO_INT (true); | (* bres is 1 *) |
| :--- | :--- |
| tres $:=$ ANY_TO_INT | (* tres is 46 *) |
| (t\#0s 46 ms$) ;$ |  |

## Converts variables to 64-bit long integer variables.

Tip: The maximum value for a REAL or LREAL input must be less than $9.2233720 \mathrm{e}+18$. For input values greater than this maximum, the output value is determined by the target type. For Windows ${ }^{\oplus}$ and Linux targets, the output value is reset to zero when the input value is greater than $9.2233720 \mathrm{e}+18$. While for ONX targets, the output value will go into overflow.


| Arguments |  |  |
| :--- | :--- | :--- |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DIN - UDINT - DWORD - LINT - ULINT - LWORD - REAL | Any value |
| 01 | - LREAL - TIME - DATE - STRING | 0 if il is FALSE / 1 if ii is TRUE <br> number of milliseconds for a timer <br> integer part for real <br> decimal number represented by a string |

## Example

(* FBD example with "Convert to Long Integer" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_LINT (true); | $(*$ bres is $1 \quad *)$ |
| :--- | :--- | :--- |
| tres $:=$ ANY_TO_LINT | $(*$ tres is $46 \quad *)$ |
| $($ t\#0s 46 ms$) ;$ |  |

## ANY_TO_LWORD

## Converts variables to 64-bit long WORD variables

Tip: The maximum value for a REAL or LREAL input must be less than $1.8446744 \mathrm{e}+19$. For input values greater than this maximum, the output value is determined by the target type. For Windows ${ }^{\ominus}$ and Linux targets, the output value will be reset to zero when the input value is greater than $1.8446744 \mathrm{e}+19$. While for QNX targets, the output value will go into overflow.

|  | $\mathrm{TO}_{2}$ |
| :---: | :---: |
| An | ${ }_{\text {LWORD }}$ |


| Arguments |  |  |
| :--- | :--- | :--- |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL <br> - LREAL - TIME - DATE - STRING | Any value |
| 01 | LWORD | 0 if $i 1$ is FALSE / 1 if $i 1$ is TRUE <br> number of milliseconds for a timer integer part <br> for real decimal number represented by a string |

## Example

(* FBD example with "Convert to Long Word" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_LWORD (true) ; | $(*$ bres is $1 *)$ |
| :--- | :--- |
| tres $:=$ ANY_TO_LWORD | $(*$ tres is $46 *)$ |
| $($ t\#0s46ms); |  |
| mres $:=$ ANY_TO_LWORD <br> $\left(' 0198^{\prime}\right) ;$ | $(*$ mres is $198 *)$ |

## ANY_TO_REAL

ANY_TO_SINT

Converts variables to REAL variables


## Arguments

| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL <br> -LREAL - TIME - DATE - STRING | Any value |
| :--- | :--- | :--- |
| 01 | REAL | 0.0 if il is FALSE / 1.0 if il is TRUE <br> number of milliseconds for a timer <br> equivalent number for integer |

## 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 *) |

Converts variables 8 -bit short integer variables


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - <br> WORD - DINT - UDINT - DWORD - LINT - <br> ULINT - LWORD - REAL - LREAL - TIME - <br> DATE - STRING | Any value |  |
| 01 | SINT | 0 if il is FALSE / 1 if il is TRUE number of milliseconds for <br> a timer <br> integer part for real decimal number represented by a <br> string |  |

## Example

(* FBD example with "Convert to Short Integer" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_SINT (true); |
| :--- |
| tres $:=$ ANY_TO_SINT |
| (t\#0s46ms); |
| mres $:=$ ANY_TO_SINT ('0198'); |

## See also

Operators on page 413

## ANY_TO_STRING

Converts variables to STRING variables


| Arguments |  |  |
| :--- | :--- | :--- |
| i1 | BOOL - SINT - USINT - BYTE - INT - <br> UINT - WORD - DINT - UDINT - DWORD <br> - LINT - ULINT - LWORD - REAL - <br> LREAL - TIME - DATE - STRING |  |
| 01 | STRIN value |  |
|  |  | If il is a Boolean, 'FALSE' or 'TRUE' <br> If il is an integer or a real, decimal representation <br> If il is a TIME: <br> TIME time1 <br> STRING s1 <br> time1 :=13 ms; <br> s1 :=ANY_TO_STRING(time1); <br> (* s1 = 'Osi3' *) |

## 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 *) |

## ANY_TO_TIME

Converts variables to TIME variables, except for TIME and DATE variables. The SUB_DATE_DATE function enables the conversion of a DATE to TIME format.


| Arguments |  |  |
| :--- | :--- | :--- |
| i1 | BOOL - SINT - USINT - BYTE - INT - <br>  <br> UINT - WORD - DINT - UDINT - DWORD <br>  <br> - LINT - ULINT - LWORD - REAL - <br> LREAL - TIME - DATE - STRING | Any value <br> il (or integer part of il if it is real) is the number of milliseconds <br> STRING (number of milliseconds, for example, a value of 300032 <br> represents 5 minutes and 32 milliseconds) |
| 01 | TIME | time value represented by il. A value of 1193h2m47s295ms <br> indicates an invalid time. |

## 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 *) |  |

Converts variables to 32-bit unsigned double integer variables


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL <br> - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | UDINT | if i 1 is FALSE / 1 if i 1 is TRUE <br> number of milliseconds for a timer integer part <br> for real decimal number represented by a string |  |

## Example

(* FBD example with "Convert to Unsigned Double Integer" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_UDINT (true); | $(*$ bres is $1 *)$ |
| :--- | :--- |
| tres $:=$ ANY_TO_UDINT <br> (t\#1s46ms); | $(*$ tres is 1046 *) |
| mres $:=$ ANY_TO_UDINT <br> $\left(' 0198^{\prime}\right) ;$ | $(*$ mres is $198 *)$ |

## ANY_TO_UINT

Converts variables to 16 -bit unsigned integer variables


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL <br> - LREAL - TIME - DATE - STRING | Any value |  |
| 01 | UINT | 0 if 1 is FALSE / 1 if 1 is TRUE <br> number of milliseconds for a timer integer part <br> for real decimal number represented by a string |  |

## Example

(* FBD example with "Convert to Unsigned Integer" Operators *)

(*ST Equivalence: *)

| bres $:=$ ANY_TO_INT (true); | (* bres is 1 *) |
| :--- | :--- |
| tres $:=A N Y \_T O \_I N T ~$ <br> (t\#0s46ms) ; | $(*$ tres is 46 *) |
| mres $:=$ ANY_TO_INT ('0198'); | (* mres is $198 *)$ |

## ANY_TO_ULINT

Converts variables to 64-bit unsigned long integer variable
Tip: The maximum value for a REAL or LREAL input must be less than
$1.8446744 \mathrm{e}+19$. For input values greater than this maximum, the output value is determined by the target type. For Windows ${ }^{\circledR}$ and Linux targets, the output value is reset to zero when the input value is greater than $1.8446744 e+19$. While for $O N X$ targets, the output value goes into overflow.

| ANY_T ${ }_{\text {_ }}$ UULINT |  |  |
| :---: | :---: | :---: |
| $j_{\text {any }}^{11}$ | บINT |  |


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - | Any value |  |
|  | DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL |  |  |


| 01 | ULINT | 0 if i 1 is FALSE $/ 1$ if i 1 is TRUE <br> number of milliseconds for a timer integer part <br> for real decimal number represented by a string |
| :--- | :--- | :--- |

## Example

(* FBD example with "Convert to Unsigned Long Integer" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_ULINT (true); | (* bres is 1 *) |
| :--- | :--- |
| tres $:=$ ANY_TO_ULINT <br> (t\#0s46ms); | (* tres is $46 *)$ |
| mres $:=$ ANY_TO_ULINT <br> $\left(' 0198^{\prime}\right) ; ~$ | $(*$ mres is $198 *)$ |

## ANY_TO_USINT

## Converts variables to 8-bit unsigned short integer variables



| Arguments |  |  |
| :--- | :--- | :--- |
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - <br> REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | USINT | 0 if il is FALSE / 1 if il is TRUE number of <br> milliseconds for a timer integer part for real <br> decimal number represented by a string |

## Example

(* FBD example with "Convert to Short Integer" Operators *)

(* ST Equivalence: *)

| bres := ANY_TO_USINT (true); | $\left({ }^{*}\right.$ bres is $\left.1^{*}\right)$ |
| :--- | :--- |
| tres :=ANY_TO_USINT (t\#Os46ms); | $\left({ }^{*}\right.$ tres is $\left.466^{*}\right)$ |
| mres := ANY_TO_USINT ('O198'); | $\left({ }^{*}\right.$ mres is $\left.198{ }^{*}\right)$ |

Converts variables to 16 -bit WORD variables


Arguments

| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - <br> DINT - UDINT - DWORD - LINT - ULINT - LWORD - <br> REAL - LREAL - TIME - DATE - STRING | Any value <br> Oif i is FALSE / 1 if il is TRUE <br> number of milliseconds for a timer <br> integer part for real <br> decimal number represented by a string |
| :--- | :--- | :--- |
| 01 | WORD |  |

## Example

(* FBD example with "Convert to WORD" Operators *)

(* ST Equivalence: *)

| bres $:=$ ANY_TO_WORD (true); |
| :--- |
| tres $:=$ ANY_TO_WORD |
| $($ t\# 0 s 46 ms$) ;$ |

Division of two integer or real variables (the first divided by the second).


Tip: Fix all potential divide by zero instances in program organization units (POUs). A divide by zero error displays "Wait" instead of live data in the language editor when simulating an application.

| Arguments |  |  |  |  |
| :--- | :--- | :--- | :---: | :---: |
| i1 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br>  <br> - LWORD - REAL - <br> LREAL |  |  |  |


| Arguments |  |  |
| :--- | :--- | :--- |
| i2 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL |  |
| 01 | SINT - USINT - BYTE - - <br> INT - UINT - WORD - integer or real value (divisor) <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL | integer or real division of il by i2 |

## Example

(* FBD example with Division Operators *)

(* ST Equivalence: *)

```
ao10 := ai101 / ai102;
ao5 := (ai5 / 2) / ai53;
```


## See also

## Multiplication on page 434

Simulate an application on page 137
For integer, REAL, TIME, DATE, and STRING variables, compares the first input to the second to determine equality.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | BOOL - SINT - USINT - <br> BYTE - INT - UINT - <br> WORD - DINT - UDINT <br> - DWORD - LINT - <br> ULINT - LWORD - <br> REAL - LREAL - TIME - <br> DATE - STRING | Both inputs must have the same format. |  |
| i2 | BOOL - SINT - USINT - <br> BYTE - INT - UINT - <br> WORD - DINT - UDINT <br> - DWORD - LINT - <br> ULINT - LWORD - <br> REAL - LREAL - TIME - <br> DATE - STRING |  |  |
| 01 | BOOL | TRUE if i1 = i2 |  |

## Example

(* FBD example with "Is Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```


## See also

## Not Equal on page 436

## Greater Than

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is greater than the second.


| Arguments |  |  |  |  |
| :--- | :--- | :--- | :---: | :---: |
| i1 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWoth inputs must have the same type <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING |  |  |  |


| arguments |  |  |  |  |  |  |
| :--- | :--- | :--- | :---: | :---: | :---: | :---: |
| i2 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING |  |  |  |  |  |
| 01 | BOOL | TRUE if i1 > i2 |  |  |  |  |

## Example

(* FBD example with "Greater than" Operators *)

(* ST Equivalence: *)

```
aresult := (10 > 25); (* aresult is FALSE *)
mresult := ('ab' > 'a'); (* mresult is TRUE *)
```


## See also

Less Than on page 432
Greater Than or Equal on page 431
For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is greater than or equal to the second.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.


| Arguments |  |  |  |  |
| :--- | :--- | :--- | :---: | :---: |
| i1 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> Doth inputs must have the same type. <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING |  |  |  |


| Arguments |  |  |
| :---: | :---: | :---: |
| i2 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING |  |
| 01 | B00L | TRUE if i1 >= i2 |

## Example

(* FBD example with "Greater or Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 >= 25); (* aresult is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```


## See also

## Greater Than on page 430

Less Than or Equal on page 433

## Less Than

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is less than the second.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING | Both inputs must have the same type |  |
| i2 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME - DATE - <br> STRING |  |  |
| 01 | BOOL | TRUE if ili2 < i2 |  |

## Example

(* FBD example with "Less than" Operators *)

(* ST Equivalence: *)

```
aresult := (10<25); (* aresult is TRUE *)
mresult := ('z' < 'B'); (* mresult is FALSE *)
```


## See also

Greater Than on page 430
Less Than or Equal on page 433
For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is less than or equal to the second.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| i1 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT | Both inputs must have the same type. |  |
| - LWORD - REAL - |  |  |  |
| LREAL - TIME - DATE - |  |  |  |
| STRING |  |  |  |$\quad$.

## Example

(* FBD example with "Less or Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```


## See also

Greater Than or Equal on page 431
Less Than on page 432

## Multiplication

Multiplies 2 to 127 integer or real variables.


Arguments

| (inputs) | SINT - USINT - BYTE INT - UINT - WORD DINT - UDINT DWORD - LINT - ULINT - LWORD - REAL LREAL | All inputs must have the same format. From 2 to 127 inputs. |
| :---: | :---: | :---: |
| output | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL | Multiplication of the input variables. |

## Example

(* FBD example with Multiplication Operators *)

(* ST equivalence *)
ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;

## See also

Division on page 428

## Converts variables to negated variables



Arguments

| i1 | BOOL | Input and output must have the same format |
| :--- | :--- | :--- |
| 01 | BOOL |  |

## Example

(* FBD example with Negation Operators *)

(* ST equivalence: *)

```
ao23 := - (ail0);
ro100 := - (ri1 + ri2);
```

For Boolean expressions, converts variables to negated variables.


## Arguments

| i1 | both inputs must have the same type |
| :--- | :--- |
| 01 | TRUE when il is FALSE <br> FALSE when il is TRUE |

## Example

(* FBD example with "NOT" Operator *)

(* ST equivalence: *)
bo10 := NOT bil01;

Not Equal
For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is not equal to the second.


Arguments

| i1 | $\begin{aligned} & \text { BOOL - SINT - USINT - } \\ & \text { BYTE - INT - UINT - } \\ & \text { WORD - DINT - UDINT } \\ & \text { - DWORD - LINT - } \\ & \text { ULINT - LWORD - } \\ & \text { REAL - LREAL - TIME - } \\ & \text { DATE - STRING } \\ & \hline \end{aligned}$ | both inputs must have the same type |
| :---: | :---: | :---: |
| i2 | BOOL - SINT - USINT - <br> BYTE - INT - UINT - <br> WORD - DINT - UDINT <br> - DWORD - LINT - <br> ULINT - LWORD - <br> REAL - LREAL - TIME - <br> DATE - STRING |  |
| 01 | BOOL | TRUE if first <> second |

## Example

(* FBD example with "Is Not Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```


## See also

## Equal on page 429

Boolean OR of 2 to 127 Boolean variables.


## Arguments

| (inputs) | BOOL | From 2 to 127 inputs. |
| :--- | :--- | :--- |
| output | BOOL | Boolean OR of the input variables. |

## Example

(* FBD example with "OR" Operators *)

(* ST equivalence: *)

```
bo10 := bi101 OR NOT (bi102);
```

bo5 := (bi51 OR bi52) OR bi53;

## See also

## AND on page 416

XOR on page 438

## Subtraction

## Subtraction of two integer, real, or TIME variables.



| Arguments |  |  |
| :---: | :---: | :---: |
| i1 | SINT - USINT - BYTE <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME | can be of any integer, real or long real, or TIME format |
| i2 | SINT - USINT - BYTE - <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME | (i1 and i2 must have the same format) |
| 01 | SINT - USINT - BYTE <br> INT - UINT - WORD - <br> DINT - UDINT - <br> DWORD - LINT - ULINT <br> - LWORD - REAL - <br> LREAL - TIME | subtraction (first minus second) |

## Example

(* FBD example with Subtraction Operators *)

(* ST equivalence: *)

```
ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;
```


## See also

## Addition on page 415

Boolean exclusive OR of two variables.


Arguments

| i 1 | BOOL |  |
| :--- | :--- | :--- |
| i 2 | BOOL |  |
| o1 | B00L | Boolean exclusive OR of the two input items |

## Example

(* FBD example with "XOR" operators *)

(* ST equivalence: *)

```
bo10 := bil01 XOR NOT (bi102);
bo5 := (bi51 XOR bi52) XOR bi53;
```


## See also

OR on page 436

## Functions

The following are the functions supported by the system:

| Functions | Description |
| :---: | :---: |
| 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, COS | 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 |
| COS | 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 provides the difference in TIME format |

Yields the absolute (positive) value of a REAL value.


Arguments

| IN | IN | REAL | Any signed real value |
| :--- | :--- | :--- | :--- |
| ABS | 0 | REAL | Absolute value (always positive) |

## Example

(* FBD Program using "ABS" Function *)

(* ST Equivalence: *)

```
over := (ABS (delta) > range);
```

ACOS
Yields the Arc Cosine of a REAL value. Input and output values are in radians.


| Arguments |  |  |  |
| :--- | :--- | :--- | :--- |
| IN | IN | REAL | Must be in set $[-1.0 \ldots+1.0]$ |
| ACOS | 0 | REAL | Arc-cosine of the input value (in set $[0.0$ <br> . PI] <br> $=0.0$ |

## Example

```
(* FBD Program using "COS" and "ACOS" Functions *)
```


(* ST Equivalence: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```


## See also

ASIN on page 442
ATAN on page 442
COS on page 443
Integer AND bit-to-bit mask.


Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |

Arguments

| MSK | MSK | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| AND_MASK | 0 | DINT | Bit-to-bit logical AND between IN and <br> MSK |

## 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 *)
```


## See also

## NOT MASK on page 454

OR MASK on page 455
XOR MASK on page 465
Yields the ASCII code for characters in strings.


Arguments

| IN | IN | STRING | Any non-empty string |
| :--- | :--- | :--- | :--- |
| Pos | Pos | DINT | Position of the selected character in set <br> [1.. len] (len is the length of the IN string) |
| ASCII | Code | DINT | Code of the selected character (in set [0 <br> $. .255])$ <br> yields 0 is Pos is out of the 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 *)
```


## ASIN

## Example

(* FBD Program using "SIN" and "ASIN" Functions *)

(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```


## See also

ACOS on page 440
ATAN on page 442
SIN on page 462
Yields the Arc Tangent of a REAL value.


Arguments

| IN | IN | REAL | Any real value |
| :--- | :--- | :--- | :--- |
| ATAN | 0 | REAL | Arc-tangent of the input value (in set $[-\mathrm{PI} / 2 ~ . .+\mathrm{PI} / 2])$ <br> $=0.0$ for invalid input |

## Example

(* FBD Program using "TAN" and "ATAN" Function *)

(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```


## See also

ACOS on page 440
ASIN on page 442
TAN on page 464
Gives a one character message string from a given ASCII code.


## Arguments

| Code | Code | DINT | Code in set [0 .. 255] |
| :--- | :--- | :--- | :--- |
| CHAR | 0 | MESSAGE | One character string <br> the character has the ASCII code given in input Code <br> (ASCII code is used modulo 256) |

## Example


(* 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' *)
```

Yields the Cosine of a REAL value.


Arguments

| IN | IN | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| $\operatorname{COS}$ | 0 | REAL | Cosine of the input value (in set $[-1.0 . .+1.0]$ ) |

## Example

(* FBD Program using "COS" and "ACOS" Functions *)

(* ST Equivalence: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```


## See also

ACOS on page 440
Retrieves the current date.


| Arguments |
| :--- |
| CURRENT_ISA_DATE |

## Example

(* FBD Program using "CURRENT_ISA_DATE" Function *)

(* ST Equivalence: *)

```
datResult := CURRENT_ISA_DATE();
```


## See also

SUB DATE DATE on page 463
DELETE
Deletes part of a string.


Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first deleted character |
| DELETE | 0 | MESSAGE | modified string <br> empty string if Pos $<1$ <br> initial string if Pos $>$ IN string length <br> initial string if NbC $<=0$ |

## Example


(* ST Equivalence: *)

```
complete string := INSERT ('ABCD ', 'EFGH', 5); (*
```

complete_string is 'ABCDEFGH ' *)
sub_string := DELETE (complete_string, 4, 3); (*
sub_string is 'ABGH '*)

## See also

INSERT on page 446

## EXPT

Where 'base' is the first argument and 'exponent' is the second argument, yields the REAL result of this operation: (base exponent).


## Arguments

| IN | IN | REAL | Any signed real value |
| :--- | :--- | :--- | :--- |
| EXP | EXP | DINT | Integer exponent |
| EXPT | 0 | REAL | $($ IN EXP $)$ |

## Example


(* ST Equivalence: *)
t.b_size := ANY_TO_DINT (EXPT (2.0, range) );

## See also

## POW on page 456

Locates and provides the position of sub-strings within strings.


## Arguments

| IN | IN | MESSAGE | Any message string |
| :--- | :--- | :--- | :--- |
| Pat | Pat | MESSAGE | Any non-empty string (Pattern) |
| FIND | Pos | DINT | $=0$ if sub string Pat not found <br> = position of the first character of the first occurrence of the sub- <br> string Pat <br> (first position is 1) <br> this function is case sensitive |

## Example

(* FBD Program using "FIND" Function *)

(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is
'ABCDEFGH ' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```


## See also

REPLACE on page 457
Inserts sub-strings at user-defined positions within strings.


## Arguments

| IN | IN | STRING | Initial string |
| :--- | :--- | :--- | :--- |

Arguments

| Str | Str | STRING | String to be inserted |
| :--- | :--- | :--- | :--- |
| Pos | Pos | DINT | Position of the insertion <br> the insertion is done before the position <br> (first valid position is 1) |
| INSERT | 0 | STRING | Modified string <br> empty string if Pos <= 0 <br> concatenation of both strings if Pos is greater than the length of the <br> IN string |

## Example


(* ST Equivalence: *)

```
MyName := INSERT ('Mr JONES', 'Frank ', 4);
(* MyName is 'Mr Frank JONES' *)
```


## See also

## DELETE on page 444

From the left end of strings, yields the number of characters defined.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be <br> greater than the length of the IN string. |
| LEFT | 0 | MESSAGE | Left part of the IN string (its length $=$ NbC) <br> empty string if $\mathrm{NbC}<=0$ <br> complete $\operatorname{IN}$ string if $\mathrm{NbC}>=$ IN string length |

## Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)

(* ST Equivalence: *)

```
complete_string := INSERT (RIGHT ('12345678', 4), LEFT
('12345678', 4), 5);
(* complete_string is '56781234'
the value issued from RIGHT call is '5678'
the value issued from LEFT call is '1234'
*)
```


## See also

RIGHT on page 458
MID on page 450

## LIMIT

Restricts integer values to a given interval. Integer values between the minimum and maximum are unchanged. Integer values greater than the maximum are replaced with the maximum value. Integer values less than the minimum are replaced with the minimum value.


Arguments

| MIN | MIN | DINT | Minimum value allowed |
| :--- | :--- | :--- | :--- |
| IN | IN | DINT | Any signed integer value |
| MAX | MAX | DINT | Maximum value allowed |
| LIMIT | 0 | DINT | Input value restricted to the allowed range |

## 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
*)
```


## See also

## MIN on page 451

MAX on page 449

## Example

## (* FBD Program using "LOG" Function *)


(* ST Equivalence: *)

```
xpos := ABS (xval);
xlog := LOG (xpos);
```

Yields the maximum of two integer values.


| Arguments |
| :--- |
| IN1 |
| IN 1 N |
| IN |
| IN2 |
| DINT |
| DINT |
| MAX |

## 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 *)
```


## See also

## MIN on page 451

Using the position and number of characters provided, yields required parts of strings.


## Arguments

| IN | IN | STRING | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to extract (must be less than or equal to the <br> length of the IN string) |
| Pos | Pos | DINT | Position of the sub-string <br> the sub-string first character is the one pointed to by Pos <br> (the first valid position is 1) |
| MID | 0 | STRING | Middle part of the string lits length $=$ NbC). <br> When the number of characters to extract exceeds the length of the <br> IN string, NbC is automatically recalculated to get the remainder of <br> the string only. When NbC or Pos are zero or negative numbers, an <br> empty string is returned. |

## Example

(* FBD Program using "MID" Function *)

(* ST Equivalence: *)

```
sub_string := MID ('abcdefgh', 2, 4);
(* sub string is 'de' *)
```


## See also

LEFT on page 447
RIGHT on page 458
Yields the minimum of two integer values.


Arguments

| IN1 | IN1 | DINT | Any signed integer value |
| :--- | :--- | :--- | :--- |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MIN | 0 | DINT | Minimum of both input 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
*)
```


## See also

MAX on page 449
Yields the length of strings.


## Arguments

| IN | IN | STRING | Any string |
| :--- | :--- | :--- | :--- |
| MLEN | NbC | DINT | Number of characters in the IN string |

## Example

(* FBD Program using "MLEN" Function *)

(* ST Equivalence: *)

```
n.bchar := 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 three characters *)
```

Yields the modulo of an integer value.


## Arguments

| IN | IN | DINT | Any signed INTEGER value |
| :--- | :--- | :--- | :--- |
| Base | Base | DINT | Must be greater than zero |
| MOD | 0 | DINT | Modulo calculation (input MOD base) <br> yields -1 if Base $<=0$ |

## Example

(* FBD Program using "MOD" Function *)

(* ST Equivalence: *)

```
division_result := (value / divider); (* integer division
*)
rest_of_division := MOD (value, divider); (* rest of the
division *)
```

Yields a value between four integer values.


## Arguments

| SEL | SEL | DINT | Selector integer value (must be in set [0..3]) |
| :---: | :---: | :---: | :---: |
| IN1...IN4 | IN1...IN4 | DINT | Any integer values |
| MUX4 | 0 | DINT | $\begin{aligned} & =\text { = value1 if } S E L=0 \\ & =\text { value2 if } S E L=1 \\ & =\text { value3 if } S E L=2 \\ & =\text { value4 if } S E L=3 \\ & =0 \text { for all other values of the selector } \end{aligned}$ |

## Example


(* ST Equivalence: *)

```
range := MUX4 (choice, 1, 10, 100, 1000);
(* select from 4 predefined ranges, for example, if choice
is 1, range will be 10 *)
```


## See also

## MUX8 on page 453

Yields a value between eight integer values.


Arguments

| SEL | SEL | DINT | Selector integer value (must be in set [0..7]) |
| :--- | :--- | :--- | :--- |
| IN1...IN8 | IN1...IN8 | DINT | Any integer values |
| MUX8 | 0 | DINT | $=$ value1 if selector $=0$ <br> $=$ value2 if selector $=1$ <br> $\ldots$. <br> $=$ <br> $=$ <br> $=0$ value 8 if selector $=7$ |

## 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 *)
```


## See also

## MUX4 on page 453

Integer bit-to-bit negation mask.


## Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| NOT_MASK | 0 | DINT | Bit-to-bit negation on 32 bits of IN |

## Example

(* FBD example with NOT_MASK Operators *)

(*ST equivalence: ${ }^{*}$ )

```
result := NOT_MASK (16#1234);
(* result is 16#FFFF_EDCB *)
```


## See also

AND MASK on page 440
OR MASK on page 455
XOR MASK on page 465
Tests the parity of an integer: result is odd or even.


## Arguments

| IN | IN | DINT | Any signed integer value |
| :--- | :--- | :--- | :--- |
| Odd | 0 | DINT | TRUE if input value is odd <br> FALSE if input value is even |

## Example


(* ST Equivalence: *)

```
If Not (ODD (value)) Then Return; End_if;
```

value := value + 1;
(* makes value always even *)

Integer OR bit-to-bit mask.


## Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| MSK | MSK | DINT | Must have integer format |
| OR_MASK | 0 | DINT | Bit-to-bit logical OR between IN and MSK |

## 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 *)

## See also

AND MASK on page 440
NOT MASK on page 454
XOR MASK on page 465
When the first argument is 'base' and the second argument is 'exponent', yields the REAL result of this: (base exponent). 'Exponent' is a REAL value.


## Arguments

| IN | 1 N | REAL | Real number to be raised |
| :---: | :---: | :---: | :---: |
| EXP | EXP | REAL | Power (exponent) |
| Pow | 0 | REAL | (IN EXP) <br> 1.0 if IN is not 0.0 and EXP is 0.0 <br> 0.0 if IN is 0.0 and EXP is negative <br> 0.0 if both IN and EXP are 0.0 <br> 0.0 if IN is negative and EXP does not correspond to an integer |

## Example

(* FBD Program using "POW" Function *)

(* ST Equivalence: *)

```
result := POW (xval, power);
```


## See also

EXPT on page 445
RAND

## 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 [O..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,
*)
```

Replaces a part of a message string by a new set of characters.


## Arguments

| IN | IN | MESSAGE | Any string |
| :--- | :--- | :--- | :--- |
| Str | Str | MESSAGE | String to be inserted (to replace NbC chars) |

## Arguments

| NbC | NbC | DINT | Number of characters to be deleted |
| :--- | :--- | :--- | :--- |
| Pos | Pos | DINT | Position of the first modified character <br> (first valid position is 1) |
| REPLACE | 0 | MESSAGE | Modified string: <br> -NbC characters are deleted at position Pos <br> - then substring Str is inserted at this position <br> returns empty string if Pos $<=0$ <br> returns strings concatenation (IN + Str) if Pos is greater than the <br> length of the IN string <br> returns initial string IN if NbC < $<=0$ |

## Example

(* FBD program using "REPLACE" function *)

(* ST Equivalence: *)

```
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
(* MyName is 'Mr Frank JONES' *)
```


## See also

## FIND on page 446

From the right ends of strings, yields the number of characters defined.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be <br> greater than the length of the IN string. |
| RIGHT | 0 | MESSAGE | Right part of the string (length $=\mathrm{NbC})$ <br> empty string if $\mathrm{NbC}<=0$ <br> complete string if $\mathrm{NbC}>=$ string length |

## Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)

(* ST Equivalence: *)

```
complete_string := INSERT (RIGHT ('12345678', 4), LEFT
('12345678', 4),5);
(* complete_string is '56781234'
the value issued from RIGHT call is '5678'
the value issued from LEFT call is '1234'
*)
```


## See also

## LEFT on page 447

MID on page 450
Rotates the DINT type input by NbR bits to the left in a circular form and fills the bits on the right with the bits that are rotated.


Arguments

| IN | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| ROL | 0 | DINT | Left rotated value <br> no effect if $\operatorname{NbR}<=0$ |

## Example

(* FBD Program using "ROL" Function *)

(* ST Equivalence: *)

```
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result = 2#1001_1010_0110_1010*)
```


## See also

ROR (Rotation Right) on page 460

## ROR (Rotation Right)

## Example

> (* FBD Program using "ROR" Function *)

(* ST Equivalence: *)

```
result := ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1010_0110_1001_1010 *)
```


## See also

ROL (Rotation Left) on page 459
Specifies the input to use between two integer values.


Arguments

| SEL | SEL | BOOL | Indicates the chosen value |
| :--- | :--- | :--- | :--- |
| IN1, IN2 | IN1, IN2 | DINT | Any integer values |
| SEL | 0 | DINT | $=$ IN1 if SEL is FALSE <br> $=$ IN2 if SEL is TRUE |

## Example

(* FBD Program using "SEL" Function *)

(* ST Equivalence: *)
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
(* process command selection *)
SHL (Shift Left Arithmetic or
For 32-bit integers, moves integers to the left and places 0 in the least significant bit. Shift Left Signed)


Arguments

| $\mathbb{N}$ | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbS | NbS | DINT | Number of 1 bit shifts (in set $[1.31])$ |
| SHL | 0 | DINT | Left shifted value <br> no effect if NbS $<=0$ <br> 0 replaces the least significant bit |

## Example

(* FBD Program using "SHL" Function *)

(* ST Equivalence: *)

```
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```


## See also

SHR (Shift Right Arithmetic or Shift Right Signed) on page 462
SHR (Shift Right Arithmetic or Shift Right Signed)

## Example

(* FBD Program using "SHR" Function *)

(* ST Equivalence: *)

```
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result = 2#1110_0110_1001_1010 *)
```


## See also

SHL (Shift Left Arithmetic or Shift Left Signed) on page 461
Yields the Sine of a REAL value.


Arguments

| IN | IN | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| SIN | 0 | REAL | Sine of the input value (in set $[-1.0 \ldots+1.0])$ |

## Example

(* FBD Program using "SIN" and "ASIN" Functions *)

(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```


## See also

COS on page 443
TAN on page 464
Yields the square root of a REAL value.


## Arguments

| IN | IN | REAL | Must be greater than or equal to zero |
| :--- | :--- | :--- | :--- |
| SQRT | O | REAL | Square root of the input value |

## Example

(* FBD Program using "SQRT" Function *)

(* ST Equivalence: *)

```
xpos := ABS (xval);
xroot := SQRT (xpos);
```


## SUB_DATE_DATE

Compares two dates and yields the difference in TIME format.


| Arguments |  |  |  |
| :--- | :--- | :--- | :--- |
| DAT1 | DAT1 | DATE | First date in a comparison |
| DAT2 | DAT2 | DATE | Second date in a comparison |


| SUB_DATE_DATE | TIME | TIME | Difference in TIME format between DAT1 and DAT2. The <br> possible date difference values range from t\#Oh to <br> t\#1193h2m47s294ms inclusively. <br> A value of 1193h2m47s295ms indicates an error for either <br> of these conditions: <br> -DAT1 is less than DAT2 <br> - The difference between DAT1 and DAT2 is greater than <br> 1193h2m47s294ms |
| :--- | :--- | :--- | :--- |

## Example

(* FBD Program using "SUB_DATE_DATE" Function *)

(* ST Equivalence: *)

```
timResult := SUB_DATE_DATE (datVal1, datVal2);
```


## See also

CURRENT ISA DATE on page 444
Yields the Tangent of a REAL value.


## Arguments

| IN | IN | REAL | Must be greater than or equal to zero |
| :--- | :--- | :--- | :--- |
| TAN | 0 | REAL | Tangent of the input value <br> $=1 \mathrm{E}+38$ for invalid input |

## Example

(* FBD Program using "TAN" and "ATAN" Functions *)

(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```


## See also

## SIN on page 462

Truncates REAL values, leaving just the integer.


## Arguments

| IN | IN | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| TRUNC | 0 | REAL | If $I N>0$, biggest integer less or equal to the input <br> If $I N<0$, least integer greater or equal to the input |

## Example

(* FBD Program using "TRUNC" Function *)

(* ST Equivalence: *)

```
result := TRUNC (+2.67) + TRUNC (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)
```

Integer exclusive OR bit-to-bit mask


Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| MSK | MSK | DINT | Must have integer format |
| XOR_MASK | Q | DINT | Bit-to-bit logical Exclusive OR between IN and MSK |

## Example

(* FBD example with XOR_MASK Operators *)

(* ST Equivalence: *)

```
crc32 := XOR_MASK (prevcrc, nextc);
result := XOR_MASK (16#012, 16#011); (* equals 16#003
*)
```


## See also

AND MASK on page 440
NOT MASK on page 454
OR MASK on page 455
Function Blocks

AVERAGE
Use these function blocks to configure AADvance projects:

| Alarms Management |  |
| :---: | :---: |
| LIM_ALRM | High/low limit alarm with hysteresis |
| Boolean Operations |  |
| SR | Set dominant bistable |
| RS | Reset dominant bistable |
| R_TRIG | Rising edge detection |
| F-TRIG | Falling edge detection |
| Comparators |  |
| CTD | Full comparison function block |
| Counters |  |
| CTU | Up counter |
| CTD | Down counter |
| CTUD | Up-down counter |
| Data Manipulation |  |
| AVERAGE | Running average over N samples |
| Process Control |  |
| DERIVATE | Differentiation of a real value 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 |
| Time Operations |  |
| TON | On-delay timing |
| TOF | Off-delay timing |
| TP | Pulse timing |

Call new function blocks from any language.
Stores a value at each cycle and calculates the average value of all stored

values.

## 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 |

Only the latest N values are stored.

The maximum number of samples $N$ is 128 . When $N$ exceeds 128 , the number of samples is truncated to 128.

When the "RUN" command is FALSE (reset mode), the output value is equal to the input value.

Upon reaching the maximum $N$ of stored values, the first stored value is overwritten with the latest value.

Tip:
When setting or changing the value for N , set RUN to FALSE, then set it back to TRUE.

## Example

(* FBD program using the AVERAGE block: *)

(* ST Equivalence: AVERAGE1 instance of AVERAGE block *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;
```

Generates a blinking signal.


Arguments

| RUN | BOOL | Mode: TRUE=blinking / FALSE=reset the output to false |
| :--- | :--- | :--- |
| CYCLE | TIME | Blinking period. Possible values range from Oms to <br> 1193h2m47s294ms. |
| $\mathbf{O}$ | BOOL | Output blinking signal |

Timing diagram


## See also

SIG GEN on page 476

## CMP

Compares two values: tells if they are equal, or if the first is less or greater than the second one.


Arguments

| VAL1 | DINT | Any signed integer value |
| :--- | :--- | :--- |
| VAL2 | DINT | Any signed integer value |
| LT | BOOL | TRUE if vall is less than val2 |
| EQ | BOOL | TRUE if vall is equal to val2 |
| GT | BOOL | TRUE if vall is greater than val2 |

## Example

(* FBD program using the 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);
Counts (integer) from a given value down to 01 by 1


Arguments

| CD | BOOL | Counting input <br> (down-counting when CD is a rising edge) |
| :--- | :--- | :--- |
| LOAD | BOOL | Load command (dominant) <br> (CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed initial value |
| $O$ | BOOL | Underflow: TRUE when CV $<=0$ |
| CV | DINT | Counter result |

## Example

(* FBD program using the CTD block *)

(* ST Equivalence: CTD1 is an instance of block*)
CTD1(trigger,load_cmd,100);
underflow := CTD1.Q;
result := CTD1.CV;

## See also

CTU on page 469
CTUD on page 470
Counts (integer) from 0 up to a given value 1 by 1


## 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 |

## Example

(* FBD program using the CTU block *)

(* ST Equivalence: CTU1 is an instance of CTU block*)
CTU1 (command, NOT (auto_mode), 100);
overflow := CTU1.Q;
result := CTU1.CV;

## See also

## CTD on page 468

CTUD on page 470
Counts (integer) from 0 up to a given value 1 by 1 or from a given value down to 01 by 1


Arguments

| CU | BOOL | Up-counting (when CU is a rising edge) |
| :--- | :--- | :--- |
| CD | BOOL | Down-counting (when CD is a rising edge) |
| RESE | BOOL | Reset command (dominant) <br> (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 |
| OD | BOOL | Underflow: TRUE when CV $<=0$ |
| CV | DINT | Counter result |

## Example

(* FBD program using the CTUD block *)

(* ST Equivalence: We suppose CTUD1 is an instance of block*)

```
CTUD1(trigger1, trigger2, reset_cmd, load_cmd,100);
```

full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;

## See also

CTD on page 468
CTU on page 469

## DERIVATE

Differentiates a real value.


Arguments

| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| :--- | :--- | :--- |
| XIN | REAL | Input: any real value |
| CYCL | TIME | Sampling period. Possible values range from Oms to <br> 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

If the "CYCLE" parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period will use the real duration of the cycle time.

## Example

(* FBD program using the DERIVATE block: *)

(* ST Equivalence: DERIVATE1 instance of DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value := DERIVATE1.XOUT;
```


## See also

HYSTER on page 473
INTEGRAL on page 473
STACKINT on page 478
Detects a falling edge of a Boolean variable


## Arguments

| CLK | BOOL | Any Boolean variable |
| :--- | :--- | :--- |
| 0 | BOOL | TRUE when CLK changes from TRUE to FALSE <br> FALSE if all other cases |

## Example

(* FBD program using the 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;

## See also

R TRIG on page 475

## HYSTER

Hysteresis on a real value for a high limit.


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) |
| 0 | BOOL | TRUE if XIN1 has overpassed XIN2+EPS and is not yet below <br> XIN2-EPS |

Example of a timing diagram


## See also

DERIVATE on page 471
INTEGRAL on page 473
STACKINT on page 478
Integration of a real value.


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 Oms to <br> 23h59m59s999ms. |

## Arguments

| Q | BOOL | Not R1 |
| :--- | :--- | :--- |
| XOUT | REAL | Integrated output |

If the "CYCLE" parameter value is less than the cycle timing of the execution of the controller 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;
```


## See also

DERIVATE on page 471
HYSTER on page 473
STACKINT on page 478
Hysteresis on a real value for high and low limits. A hysteresis is applied on high and low limits. The hysteresis delta used for either the high or low limit is equal to the EPS parameter.


## Arguments

| $H$ | REAL | High limit value |
| :--- | :--- | :--- |
| $X$ | REAL | Input: any real value |
| L | REAL | Low limit value |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| OH | BOOL | "high" alarm: TRUE if $X$ above high limit $H$ |
| O | BOOL | Alarm output: TRUE if $X$ out of limits |
| QL | BOOL | "low" alarm: TRUE if $X$ below low limit $L$ |

Example of timing diagram


R_TRIG
Detects a rising edge of a Boolean variable


Arguments

| CLK | BOOL | Any Boolean Variable |
| :--- | :--- | :--- |
| 0 | BOOL | TRUE when CLK rises from FALSE to TRUE <br> FALSE in all other cases |

## Example

(* FBD program using the R_TRIG block *)

(* ST Equivalence: We suppose R_TRIG1 is an instance of the R_TRIG block *) R_TRIG1 (cmd);
nb_edge :=ANY_TO_DINT(R_TRIG1.Q) + nb_edge;

## See also

F TRIG on page 472
Reset dominant bistable.


Arguments

| SET | BOOL | If TRUE, sets 01 to TRUE |
| :--- | :--- | :--- |
| RESE | BOOL | If TRUE, resets 01 to FALSE (dominant) |
| O1 | BOOL | Boolean memory state |


| Set | Reset1 | $\mathbf{0 1}$ | Result 01 |
| :--- | :--- | :--- | :--- |
| 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 the RS block *)

(* ST Equivalence: We suppose RS1 is an instance of RS block *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```


## See also

SR on page 477

Generates various signal: blink on a boolean, an integer counter-up, and real sine wave.


## Arguments

| RUN | BOOL | Mode: TRUE=running / FALSE=reset to false |
| :--- | :--- | :--- |
| PERIOD | TIME | Duration of one sample. Possible values range from Oms to <br> $1193 h 2 m 47 s 294 m s$. |
| 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) |

When counting reaches maximum value, it restarts from 0 (zero). So END keeps the TRUE value only during 1 PERIOD.

## Timing diagram



## See also

BLINK on page 467
Set dominant bistable


Arguments

| SET1 | BOOL | If TRUE, sets Q1 to TRUE (dominant) |
| :--- | :--- | :--- |
| RESET | BOOL | If TRUE, resets Q1 to FALSE |
| Q1 | BOOL | Boolean memory state |


| Set | Reset | $\mathbf{0 1}$ | Result 01 |
| :--- | :--- | :--- | :--- |
| 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 the SR block *)

(* ST Equivalence: We suppose SR1 is an instance of SR block *)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```


## See also

RS on page 476
Manages a stack of integer values.


Arguments

| PUSH | BOOL | Push command (on rising edge only) <br> add the IN value on the top of the stack |
| :--- | :--- | :--- |

## Arguments

| POP | BOOL | Pop command (on rising edge only) <br> 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 <br> OUT equals O when OFLO is TRUE |

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.

This function manages invalid values as follows:

- if $\mathrm{N}<1$, STACKINT assumes a size of 1
- if $\mathrm{N}>128$, STACKINT assumes a size of 128

Tip: $\quad$ The OFLO value is valid only after a reset ( $R 1$ has been set to TRUE at least once and back to FALSE).

## Example

(* FBD program using the STACKINT block: error management *)

(* ST Equivalence: We suppose STACKINT1 is an instance of STACKINT block *)

```
STACKINT1(err_detect, acknowledge, manual_mode, err_code,
max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
```


## See also

DERIVATE on page 471
HYSTER on page 473
INTEGRAL on page 473

## TOF

Timing diagram


## See also

TON on page 480
TP on page 481
Increase an internal timer up to a given value.


## Arguments

| IN | BOOL | If rising edge, starts increasing internal timer <br> If falling edge, stops and resets internal timer |
| :--- | :--- | :--- |
| PT | TIME | Maximum programmed time |
| O | BOOL | If TRUE, programmed time is elapsed |
| ET | TIME | Current elapsed time. Possible values range from Oms to <br> $119332 m 47 s 294 m s$. |

## Timing diagram



## See also

TOF on page 480
TP on page 481
Increase an internal timer up to a given value.


| Arguments |  | BOOL |
| :--- | :--- | :--- |
| IN | If rising edge, starts increasing internal timer (if not already <br> increasing) <br> If FALSE and only if timer is elapsed, resets the internal timer <br> Any change on IN during counting has no effect. |  |
| PT | TIME | Maximum programmed time |
| $O$ | BOOL | If TRUE: timer is counting |
| ET | TIME | Current elapsed time. Possible values range from Oms to <br> 1193h2m47s294ms. |

Timing diagram


## See also

TOF on page 480
TON on page 480

## Series 9000_L Function Blocks

## KvbConsNetStatus

The Series 9000_L target supports these C function blocks:

- KvbConsNetStatus
- KvbProdNetStatus
- MathErr
- ModbusServiceStatus
- SoeServiceStatus

From a consumer controller, obtains the status of a network link to a given producer IP address. KvbConsNetStatus monitors incoming connections.


Tip: This function block should only be used for network diagnostics.

| Arguments |  |  |
| :--- | :--- | :--- |
| RES | DINT | Consumer controller number |
| IP | STRING | IP address of consumer |
| ERR | BOOL | TRUE $=$ unknown controller, IP address is invalid, or the controller has <br> no configured bindings <br> FALSE $=$ no error observed |
| STAT | BOOL | TRUE $=$ healthy network link to producer IP <br> FALSE $=$ link failure |

## Example

In an application which has two controllers with identities 1 and 2, controller 1 is a producer and has a bindings link to controller 2. Controller 1 has two network interfaces with IPs 10.44.200.1 and 192.168.3.1. Controller 2 has two network interfaces with IPs 10.44.200.2 and 192.168.3.2.

This example shows the KvbConsNetStatus functions used in a redundant manner verifying the error status of the network links for both controller 1 instances located at the different IP addresses.


## See also

## KvbProdNetStatus

Example
In an application which has two controllers with identities 1 and 2, controller 1 is a producer and has a bindings link to controller 2. Controller 1 has two network interfaces with IPs 10.44.200.1 and 192.168.3.1. Controller 2 has two network interfaces with IPs 10.44.200.2 and 192.168.3.2.

This example shows the KvbProdNetStatus functions used in a redundant manner verifying the error status of the network links for both controller 2 instances located at the different IP addresses:


## See also

KvbConsNetStatus on page 482
Obtains and enables clearing the mathematical error status lasted reported by the TIC interpreter.


| Arguments |  | BOOL |
| :--- | :--- | :--- |
| RST | TRUE = clears the math status after it is read <br> FALSE = preserves math status after it is read |  |
| ERR | Current math error status <br> $0=$ no error <br> $1=$ divide by zero occured |  |

In the following example, if the value of 'DintOp2' is zero, MathErr() returns 1 and clears the math error status.

## Structured Text Example

```
DintResult := DintOp1 / DintOp2;
MathError := MathErr(TRUE);
```

Function Block Diagram Example


## ModbusServiceStatus

The ModbusServiceStatus C function block provides the status of the Modbus ${ }^{\circledR}$ service.


Tip: To enable controller updates, perform a download operation after adding a ModbusServiceStatus function block.

This function block has these arguments:

| Arguments | BOOL | TRUE resets all status values to zero |
| :--- | :--- | :--- |
| RST | INT | The internal status |
| INTS | INT | The status of the Modbus service |
| SSTS | STRING(64) <br> ERR <br> The description of the possible error codes: <br> - Illegal function <br> - Illegal data address <br> - Illegal data value <br> - Device failure <br> - Acknowledge <br> - Busy, rejected message <br> - Nak -negative acknowledgment <br> • Memory parity error |  |
| RCV | UDINT | The total number of received messages |
| RCVE | UDINT | The number of messages received with an error |
| REP | UDINT | The total number of messages replied to |
| REPE | UDINT | The number of messages replied to with an error |
| FUNC | USINT | The Modbus function of the last message |


| ADDR | UDINT | The Base Modbus address of the last message |
| :--- | :--- | :--- |
| CNT | UINT | The Modbus data count of the last message |

## See also

## SoeServiceStatus on page 485

The SoeServiceStatus C function block provides the status of the Soe service.

| Important: | Only use the SoeServiceStatus function block in a single processor <br> configuration. Using this function block in a dual or triple processor <br> configuration stops the application. |
| :--- | :--- |



Tip: To enable controller updates, perform a download operation after adding an SoeServiceStatus function block.

This function block has these arguments.

| Arguments |  |  |
| :---: | :---: | :---: |
| RST | BOOL | TRUE resets all status values to zero |
| SSTS | INT | The status of the SOE server |
| ERR | STRING(64) | The description of the possible error codes: <br> - Not running <br> - Not running-service disabled <br> - Not running-configuration error <br> - Not running-only one service allowed <br> - Unable to initialize SOE log <br> - Unable to write SOE log |
| EVCT | UDINT | The number of events |
| EVMX | UDINT | The maximum number of events in the log |
| RESN | UINT | The number of the controller |
| COND | USINT | The event condition |
| VA | UDINT | SOE variable VA |
| RFVA | UDINT | Reference variable VA |
| RFVL | STRING(32) | Reference variable value |
| TMHI | UDINT | Event timestamp (High 32 bits) |
| TMLO | UDINT | Event timestamp (Low 32 bits) |

The timestamp is the number of milliseconds since January 1, 1900.

## See also

ModbusServiceStatus on page 484
The Glossary provides an alphabetical list of terms used in AADvance applications and their definitions.

The terms are grouped into these ranges for easy browsing:
A - C on page 486
D - H on page 492
I-N on page 498
O-R on page 502
S-Z on page 506
A-C

## access control

The use of password-protection to control access to projects, controllers, programs, libraries, and targets.

## accuracy

The degree to which the result of a measurement conforms to the correct value.

## action

A collection of operations to perform whose execution differs for each programming language.

## actuator

A device, like a valve or pump, causing an electrical, mechanical, or pneumatic action to occur in a plant component.

## add-in

A plug-in, utility, driver, or other software added to a primary application that extends its capabilities.

## address

A unique identifier used to locate a resource, such as the optional hexadecimal address that can be defined for each variable or controller.

Addresses can be used by external applications to access values of variables on a resource.

## alias

The property of a variable indicating a short name for the variable. For graphical programs, aliases indicate the parameters in functions and function blocks.

## arbitration

A mechanism for deciding which controller, device, module, or peer has control.

## architecture

Organizational structure of a computing system describing the functional relationship between the board level, device level, and system level components.

## 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].

## asynchronous

Data communications term describing a serial transmission protocol. A start signal is sent before each byte or character and a stop signal is sent after each byte or character.

## attribute

The property of a variable indicating whether a variable is read-only, writeonly, or read and write.

## battery alarm

Alarm signal indicating back-up battery voltage is low or not present.

## battery backup

Internal battery powering the internal Real Time Clock (RTC) and a portion of the volatile memory (RAM) when the power is off.

## binding

Bindings are links, for example access paths, between variables located in different controllers. From a controller, bindings are defined as having a producer variable sending values to a consumer variable. The system updates the values in the consumer to match the values in the producer.

## Boo action

A Boolean variable where the value corresponds to Step activity ( $0=$ inactive and 1=active). Possible qualifiers are Action (N), Reset (R), and Set (S).

## boolean (BOOL)

Basic type that can be used to define a variable, a parameter or an I/O module. A boolean can be TRUE (1) or FALSE (0).

## breakpoint

A mark placed by the user at particular locations within the code. While simulating, the application stops when it encounters a breakpoint. Breakpoint implementation varies for each programming language.

## build configuration

A way to store multiple versions of project properties. Build configurations are modifiable. Note that configuration is in the sense of settings.

## byte

Unsigned integer 8-bit format. Basic type that can be used to define a variable, a parameter, or an I/O module.

## C Function

A function written using the "C" language, that can be called from program organization units in a synchronous manner.

## C Language

Programming 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

A data structure of information about the routines in a program that can be used to track the steps between program organization units and called functions. Some debug information includes call stacks to assist in troubleshooting.

## cell

An individual area of a graphic matrix. Used with graphic languages and displays.

## channel

A hardware input/output point that can be connected to a controller. A channel is either an input or output. To enable use in program organization units, variables including directly represented variables are connected to channels.

## check in

Sending the contents of elements including projects, I/O modules, controllers, and program organization units for storage in a version control database. Checked-in elements can be recovered at a later time.

## child

A subordinate program which is activated by its parent. The child program has only one parent. Only the parent can start or stop child program. A parent can have more than one child.

## Common Industrial Protocol (CIP)

A communications protocol for use with automation system controllers which sends data to and from controllers using a consume/producer model.

## coil

A graphic component representing the assignment of an output or an internal variable. Standard defined in IEC 61131-3.

## common scope

Scope of a declaration applying to all program organization units within a project. (Only defined words and types can have common scope).

## condition

A boolean expression attached to an SFC Transition. In case of an SFC transition, the transition cannot be cleared when its condition is false.

## configuration

A grouping of the application and settings specified for a controller.

## connection

The link between different system components, such as networks and controllers.

## constant expression

Literal expression used to describe a constant value.

## consumer

The controller requesting the tag data from another controller.

## consumer group

A group holding external producer variables with bindings with consumer variables defined in the project.

## consumer status variable

Variables that enable the management of binding errors at the consumer level.

## 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

Depending on the type of contact, a graphic component representing the value or function of an input or an internal variable.

## continuous mode

Where the Safety Instrumented Function in the safety system continually maintains the process in a safe condition.

## controller

A programmable logic controller made up of one or more programs.

## controller system

One or more controllers, along with their power sources, communication networks, and workstations.

## controller variable

A variable whose scope is global to the project.

## 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

A function written in the " C " language that describes a conversion process. The conversion can be attached to any input or output, integer or real variable.

## coverage

The percentage of faults detected by automated diagnostics.

## Comma Separated Values (CSV) file format

A delimited data format with 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.

## cycle

The order of execution for the programs of a controller. All programs of the controller 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 timing

The amount of time given to each cycle. If a cycle is completed within the cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the program organization units of the controller, then updating physical outputs. The cycle time can differ for each cycle when no cycle timing is specified. When the cycle timing 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 trigger cycles property is false or the cycle time is 0 , the virtual machine does not wait to start a new cycle.

## cycle-to-cycle mode

Execution mode of a controller where cycles are executed one by one, according to the orders given by the user during debugging. Another execution mode for controllers is real-time mode.

## cyclic program

A time independent program that is executed during each cycle.

## data link

A directional link between controllers across which variable bindings data is conveyed.

## database

The collection of definitions making up a project. The version control feature stores checked-in information in a separate database.

## DATE

The format of a date is year-month-day, separated by hyphens. Basic data type that can be used to define a variable, a parameter or a controller.

## de-energize to action

An output circuit for a Safety Instrumented Function where the field device is usually energized. Removing the power deactivates the device.

## debug information

Log data which tracks results of stepping between program organization units and called functions Useful when troubleshooting a runtime program by simulating the step-by-step execution mode.

## debugging

Also known as simulating, the process of detecting defects in a project that includes setting and clear breakpoint, step-by-step debugging, and cycle-tocycle debugging.

## declared array

A user-defined array defined as a data type.

## declared instance

A function block having assigned instances that are declared in the dictionary. Declared instances of function blocks can be added to a program organization unit during online changes.

## defined word

Word that is an expression. This word can be used in program organization units. At compiling time the word is replaced by the expression. A defined word cannot include another defined word.

## dependency

A state in which one object requires another object to perform an operation, such as data variables in a program or functions within a library.

## design mode

An editing mode during which the application is not connected to the runtime module.

## 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.

## directly represented variable

A variable is generally declared before its use in one program organization unit. Inputs and outputs can be used without any declaration respecting a defined syntax. It corresponds to direct represented variables. Example: \%QX1.6, \%ID8. 2

## discrepancy

A condition existing when one or more elements disagree.

## DINT

An atomic data type consisting of a DWORD used to store a 32-bit signed integer value ( $-2,147,483,648$ to $+2,147,483,647$ ). Basic data type that can be used to define a variable, a parameter, or a controller.

## DWORD

Unsigned double word 32-bit format. Basic data type that can be used to define a variable, a parameter or a controller.

## duplex module

A processor or one I/O module belonging to a pair of modules arranged in a parallel configuration. The parallel arrangement enables two paths for command signals and I/O data and increases their fault tolerance.

## dynamic behavior

Continuous and sequential execution of the steps and operations of a program during an execution cycle.

## energize to action

An output circuit for a Safety Instrumented Function where the field device is usually de-energized. Applying power activates the field device.

## ETCP

A network driver that uses the TCP / IP stack.

## Eurocard I/O Module

A Eurocard I/O Module is a Eurocard sized Printed Circuit Board (PCB) without a casing. The I/O module is available with or without a front panel.

## execution control initial state (EC initial state)

The execution control state that is active upon initialization of an execution control chart.

## execution control state (EC state)

The situation in which the behavior of a basic function block with respect to its variables is determined by the algorithms associated with a specified set of execution control actions.

## execution control transition (EC transition)

The means by which control passes from a predecessor execution control state to a successor execution control state.

## execution mode

The mode in which a project is executed: real-time, cycle-to-cycle, and step-by-step.

## expression

A set of operators and identifiers grouped together to represent a numeric value, true or false value, or text string.

## 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.

## fail operational state

A condition where a fault has been masked.

## fail-safe

The ability to go to into a predetermined state when a specified malfunction occurs.

## falling edge

A falling edge of a Boolean variable corresponds to a change from TRUE (1) to FALSE (0).

## fault tolerance

Built-in capability of a system to provide continued correct execution of assigned functions in the presence of a limited number of hardware and software faults.

## fault tolerant

The ability of a system to accept the presence of one arbitrary fault while continuing to operate correctly.

## field device

Equipment, such as field wiring, sensors, final control elements, and operator interface devices, hard-wired to the field side of I/O terminals.

## field power status

A DINT variable giving status data about the field power supplies for the Digital Output Module.

## file mode

The mode where you save version control information to a repository located on a local or remote computer.

## function

A program organizational unit 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 using either structured text, instruction language, ladder diagram, function block diagrams, or "C" programming language.

## function block

A program organizational unit 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 cannot 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 using either structured text, instruction language, ladder diagram, function block diagrams, or "C" programming language.

## functional safety

The ability of a system to perform the actions required to maintain a safe state for procedures and the related equipment.

## global scope

Scope of a declaration applying to all program organizational units of one controller.

## hardware architecture

The view graphically displaying the controllers of a project and the network links between them.

## Highway Addressable Remote Transducer (HART®)

An open protocol for process control instrumentation combining digital signals with analog signals and giving control and status data for field controllers.

## healthy

A module status indicating the health of the module. The indicator is GREEN when the module is healthy and RED when there is an error.

## hidden parameter

Input parameters of a function block that are not displayed in programs.

## hierarchy

Architecture of a project, divided into several program organizational units. The hierarchy tree represents the links between parent programs and children programs.

## I-N

## identifier

A unique word used to represent a variable or a constant expression in a program.

## Instruction List (IL)

One of the five programming languages defined by the IEC 61131-3 standard for programmable logic controllers. This language was deprecated in the third edition (current) of the standard.

## initial situation

Set of the initial steps which represents the context of the program when it is started.

## initial step

The logical start of a sequential function chart (SFC). This is the step that is activated when the program starts.

## initial value

Value which has a variable when the virtual machine (application) starts the execution of the controller. 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 retained variable after the virtual machine has stopped.

## input

The 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

A copy of the internal data of a function block that 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

A mnemonic and data address defining an operation to be performed by the controller. A rung in a program consists of a set of input and output instructions. The input instructions are evaluated by the controller as being True or False. In turn, the controller sets the output instructions to True or False. Each instruction is entered on one line of text.

## integer (INT)

An atomic data type consisting of a word used to store a 16-bit signed integer value $(-32,768$ to $+32,767)$. Basic type that can be used to define a variable, a parameter, or a device.

## internal variable

A variable that is not linked to an input or output device.

## 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 complex device

Element grouping several simple I/O (input and output) devices. This provides the means for manufacturers to mix types and directions. The implementation of the I/O driver of an I/O complex device corresponds to the implementation of the drivers of all contained $\mathrm{I} / \mathrm{O}$ simple devices. OEM parameters enable providing parameters to I/O complex devices.

## I/O device

Element grouping several channels of the same type and direction. These can be either an I/O Simple Device or an I/O Complex Device.

## I/0 simple device

An I/O simple device corresponds to a piece of equipment having inputs or outputs, such as an I/O board. OEM parameters enable providing parameters to I/O simple devices. Integrators define I/O simple devices.

## I/0 variable

A variable connected to an input or output device. An I/O variable must be connected on a channel of an I/O device. An array can be connected to an I/O device if all elements are connected to contiguous channels, the type of the array must be the same type as the I/O device.

## 1/0 wiring

Definition of the links between the variables of the Project and the channels of the I/O devices existing on the Target system.

## IXL

ISaGRAF ${ }^{\circledR}$ eXchange Layer. The protocol for communications between ISaGRAF-based components.

## jump

A change in the normal sequence of program execution by executing an instruction that alters the program counter (sometimes called a branch). In ladder programs, a JUMP (JMP) instruction causes execution to jump to a labeled rung.

## keywords

Terms that have special meanings in a programming language. These terms are reserved and cannot be used as user-defined identifiers for variables, routines, or other elements.

## label

The identifier for an instruction within a program. Labels can also be used for jump operations.

## ladder diagram

An industry standard for representing relay control logic.

## link

A graphic component connecting elements in a diagram.

## literal

A value that is expressed as itself rather than as a variable's value or the result of an expression.

## local scope

The scope of a declaration applying to only one program.

## locked I/O

Input or output variable, disconnected logically from the corresponding I/O device, by a "lock" command.

## long integer (LINT)

Signed integer 64-bit format. Basic type that can be used to define a variable, a parameter, or a device.

## long real (LREAL)

Type of a variable, stored in a floating IEEE double precision 64-bit format. Basic type that can be used to define a variable, a parameter, or a device.

## long word (LWORD)

Unsigned long word 64-bit format. Basic type that can be used to define a variable, a parameter, or a device.

## modifier (IL)

Single character put at the end of an IL operation keyword, which modifies the meaning of the operation.

## network

The term network is used in different contexts:

- The means of communication between the target platform and their clients.
- For the execution order of graphic programs, a sequence of connected blocks.
$0-R$


## Original Equipment Manufacturer (OEM)

An organization that makes devices from component parts bought from other organizations.

## OEM parameter

Parameters attached to an I/O module. A parameter is characterized by a type. An OEM parameter is defined by the designer of the controller. It can be a constant, or a variable parameter entered by the user during the I/O connection.

## online mode

Mode in which the application is connected to a target and the application code is executed.

## Open Platform Communications (OPC)

A series of standards and specifications for industrial communication. An industrial automation task force developed the original standard in 1996 under the name OLE for Process Control (Object Linking and Embedding for process control). OPC specifies the communication of real-time plant data between control devices from different manufacturers.

## operator

Basic logical operation such as arithmetic, boolean, comparator, and data conversion.

## output

The direction of a variable or a device. An output variable is connected to an output channel of an output device.

## output parameter

Data that is produced as a direct result of executing an instruction or function.

## overflow

Integer value which corresponds to the number of times the cycle time has been exceeded. Always 0 , if cycle time is 0 .

## parent program

A program which controls other programs, called its children.

## peer-to-peer

One or more Ethernet networks connecting a series of devices together enabling application data to pass between them.

## ping

A signal sent across a communication channel, used as a diagnostic query data command over a link and then receiving a a reply ensuring the link is healthy and that communications can occur. No process data is transfered or modified.

## programmable logic controller (PLC)

An industrial digital computer which has been adapted for reliability in harsh conditions and is dedicated to the control of manufacturing processes.

## plug-in

A module or package integrated into a bigger platform that enables the extension of the application.

## Program Organization Unit (POU)

A set of instructions that are programs, functions or function blocks.

## power rail

Main left and right vertical rails at the extremities of a ladder diagram.

## producer

A controller producing a tag to one or more consumers requesting one.

## program

A program executes a set of instructions in a project.

## project

A set of controllers and links between their programs.

## protocol

A set of communication rules used for communications across devices.

## process safety time (PST)

The process safety time, for the equipment in use, is the length of time a dangerous condition can be present before a hazardous event occurs without the protection of a safety system.

## pulse action

A list of statements executed only once when the corresponding step is activated.

## qualifier

Determines the way the action of a step is executed.

## 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 or a Controller.

## real-time mode

The run time normal execution mode of a controller where target cycles are triggered by the cycle timing. Another execution mode for controllers is cycle-to-cycle mode.

## redundancy

The use of two or more controllers, each carrying out the same function, to improve reliability or availability.

## redundant group

A group containing the same type of I/O modules arranged in parallel for a group of I/O channels. This configuration increases fault tolerance for I/O data.

## retain

The attribute of a variable. The value of a retain variable is saved at each cycle. The value is restored during stops and restarts.

## return

Graphic component of a program representing the conditional end of a program.

## return parameter

An output parameter that uses a method called return-by-value allowing the values to be read by the calling instance (an instance of a program or Function Block).

## rising edge

The transition of a value from low to high. For example, when applied to a Boolean variable it corresponds to a change from FALSE (0) to TRUE (1).

## real-time clock (RTC)

A clock internal to the computer that keeps track of the current time in human units; year, month, date, hours, minutes, and seconds, providing an accurate date and time allowing the computer to regulate the timing and speed of various functions.

## remote terminal unit (RTU)

A computing device used in industrial control systems for remote monitoring and control of various automation devices and systems.

## run-time engine

The embedded software that solves application logic and drive I/O points.

## run-time error

An error that occurs in the program when it is executed.

## rung

The graphic component of an Ladder Diagram program representing a group of circuit elements leading to the activation of a coil in an Ladder Diagram.

## safe state

The condition that an output module maintains in the event of a network communication error, or when the controller enters the program mode. When either of these situations occur, the outputs automatically switch to the configured state (for digitals) or to the value (for analogs). This condition permits the execution of a process demand. The safe state is usually applied after the detection of a fault condition, ensuring the effect of the fault enables rather than disables a process demand.

## safety accuracy

The accuracy of an analog signal guaranteed to have no dangerous faults. The signal is declared faulty when drifting outside the safety range.

## safety critical state

A faulted condition preventing the execution of a process demand.

## scope

The level at which tags and routines may be created. They can be created at the controller level (accessible to all routines), or the program level (accessible to only the routines within a single program).

## security state

The indication of the level of access control that is applied to a given resource; such as a controller, a program organization unit, or a target.

## sequential program

A program that executes according to the dynamic behavior of the programming language and SFC languages, with a time variable explicitly synchronizing operations. A sequential program gives order to operations of a process and conditions between operations.

## sequential function chart (SFC)

A sequential function chart (SFC) is a flowchart that controls your machine or process. An SFC uses steps and transitions to perform specific operations or actions.

## short integer (SINT)

Indicates the data type is a signed integer 8-bit format. Basic type that can be used to define a a variable, a parameter or a controller.

## Safety Instrumented Function (SIF)

A process control that performs specified functions to maintain a safe state for a process when detecting unacceptable or dangerous process conditions.

## Safety Integrity Level (SIL)

Defines the safety integrity performance requirements for a safety instrumented function (SIF). There are four possible SIL levels, defined in IEC 61508 and IEC 61511, for specifying the performance requirements of the safety functions allocated to a safety-related system. Each level corresponds to a range of target likelihood of failures of a safety function. SIL4 has the highest level of safety integrity while SIL1 has the lowest.

## simplex module

A processor or I/O module in a single configuration suitable for non-safety to low demand SIL2 requirements.

## simulation mode

Mode in which virtual machines execute the code of individual controllers and the Windows platform performs aspects such as program execution.

## Safety Network Control Protocol (SNCP)

A safety protocol enabling controllers to exchange data within a Ethernet network system.SNCP is a SIL3 certified protocol providing a safety layer for Ethernet networks.

## Simple Network Time Protocol (SNTP)

A simplified version of Network Time Protocol (NTP) used to synchronize computer clocks on a network.

## spurious trip rate

The rate at which safety functions are activated without the presence of a specified process demand. Spurious trip level (STL) is a discrete level for specifying the spurious trip requirements of safety functions to be allocated to safety systems.

## status Booleans

Boolean variable giving the status information about a signal, process, action, or component. A boolean can be TRUE (1) or FALSE (0).

## stencil

A group of shapes.

## step

A basic graphic component representing a steady situation of the process. A step is referenced by a name. The activity of a step is used to control the execution of the corresponding actions. Actions are used to perform functions such as turning outputs on or off in a step.

## STRING

Data type used to denote a character string. Basic type that can be used to define a variable, a parameter, or a controller.

## structure

A data type composed of a sequence of members. A structure tag occupies a contiguous block of memory in the controller with each member in sequence in memory.

## sub-program

A program called by a parent program. A sub-program is also called a child program.

## symbol table

The file corresponding to the variables and function blocks defined for a controller. 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.

## synchronous

Data communications term describing a serial transmission protocol. A prearranged number of bits is expected to be sent across a line each second. To synchronize the sending and receiving machines, a clocking signal is sent by the transmitting computer. Synchronous communications do not use start and stop bits.

## synchronizing secondary

A chassis that has a partner primary control chassis, and is currently attempting to achieve synchronization. In this state of readiness, a secondary control chassis cannot assume the primary state.

## 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 project. You can read from or write to system variables. These variables are defined in the dsysOdef.h file. For example, the current cycle time is a system variable that can only be read by a program.

## target

The hardware platform onto which you download an application. Includes characteristics such as the memory model and variable types.

## TCP/IP

Transmission Control Protocol/Internet Protocol. A transport layer protocol and a network layer protocol developed by the Department of Defense. This is a commonly used combination for communication within networks and across internetworks.

## TIMER

A structure data type that contains status and control information for timer instructions.

## Triple Modular Redundant (TMR)

A fault tolerant arrangement in which three systems (modules) complete a process and the result is determined by a voting system to provide one output, also called a Triplicated Module.

## 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.

## transition

The change of state from an Off to On condition or vice versa.

## transition step

A basic graphic component representing the condition between different steps. A transition step is referenced by a name. A boolean condition is attached to each transition step.

## trigger cycles

A project property indicating whether a resource cycle executes according to a defined cycle timing.

## triplicated module

One of three processors or I/O modules arranged in parallel configuration. The parallel arrangement enables three paths for command signals and I/O data and increases their fault tolerance, also called Triple Modular Redundant.

## TÜV Certification

Independent third-party certification against a range of international standards, including IEC 61508.

## unsigned double integer (UDINT)

Unsigned double integer 32-bit format. Basic type that can be used to define a variable, a parameter or a controller.

## unsigned integer (UINT)

Unsigned integer 16-bit format. Basic type that can be used to define a variable, a parameter or a controller.

## unsigned long integer (ULINT)

Unsigned integer 64-bit format. Basic type that can be used to define a variable, a parameter or a controller.

## unsigned short integer (USINT)

Unsigned integer 8-bit format. Basic type that can be used to define a variable, a parameter or a controller.

## user-defined data types

Types that the user can define using basic data types or other user-defined data types. User-defined data types can be arrays or structures.

## user-defined function block

A user defined program that can be packaged into an instruction block.

## variable

A unique identifier of elementary data that is used in the programs of a project. Variables can be organized into variable groups.

## variable name

A unique identifier for a storage location containing information used in exchanges between projects.

## voting system

A redundant system ( $m$ out of $n$ ) requiring at least $m$ of the $n$ channels to be in agreement before the system can take action.

## wiring

The property of a variable indicating the I/O channel to which the variable is wired.

## WORD

A storage location in memory consisting of a group of 16 bits or 2 bytes. A grouping or number of bits in a sequence that is treated as a unit.

## write protected

The indication of whether the variable can be written.
Use the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software to create virtual machines that run on hardware components, called targets.

For AADvance applications, use the AADvance-Trusted SIS Workstation software with these types of software licenses:

- Demo License: delivered with the product and available for testing the product. This license provides a 30-day trial of the fully operational version of the AADvance-Trusted SIS Workstation software for AADvance applications.
- Full License: obtained by manually activating an unlicensed version of the product. The Full License is a fully operational version of the product that enables the creation of AADvance projects that contain multiple controllers.

The obtained license is specific to a computer. The Full license persists through all uninstall and reinstall operations. To move the license to another computer, contact the support team to remove the license and then reauthorize it on another computer.

## See also

Obtain an authorized Full license on page 513
Remove an authorized license on page 513

## Obtain an authorized Full license

After the Demo License expires, obtain a Full license to continue using the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software for AADvance applications.

## To obtain a Full license

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required to manage licenses, rightclick the SIS Workstation Windows desktop shortcut and select Run as administrator.
2. From the Help menu, select Licensing AADvance.

The licensing information displays along with three User Codes.
3. Send an email that contains the required license type and all three User Codes to the support team: keymaster@ra.rockwell.com
4. The support team emails you back Registration Keys 1 and 2.
5. From the Help menu, select Licensing AADvance.
6. Enter the registration keys in their respective fields and select Validate.

Tip: The User Codes may have changed since last displayed; the Registration Keys returned by the support team becomes invalid. Request new Registration Keys by sending the displayed User Codes.
The AADvance-Trusted SIS Workstation software is licensed for use with AADvance applications.

## See also

Remove an authorized license on page 513
To move the license to another computer, contact the support team to remove the license and then reauthorize it on another computer.

## To remove an authorized license

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required to manage licenses, rightclick the SIS Workstation Windows desktop shortcut and select Run as administrator.
2. From the Help menu, select Licensing AADvance.

The licensing information displays along with three User Codes.
3. Send an email that contains all three user codes to the support team: keymaster@ra.rockwell.com
4. The support team emails you back Registration Key 1.
5. From the Help menu, select Licensing AADvance.
6. Enter the registration key in Registration Key 1 and select Validate.

A confirmation code appears.
7. Send an email that contains the confirmation code to the support team:
keymaster@ra.rockwell.com
The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software is no longer licensed for AADvance applications.

## See also

Obtain an authorized Full license on page 513

## AADvance Manuals

When setting up AADvance ${ }^{\circledR}$ Control System, read these manuals on the Rockwell Automation ${ }^{\circledR}$ Literature Library, http://www.rockwellautomation.com/literature/:

| Publication Title | Publication Number |
| :--- | :--- |
| AADvance Safety Manual | ICSTT-RM446 |
| AADvance System Build Manual | ICSTT-RM448 |
| AADvance Controller Troubleshooting Manual | ICSTT-RM406 |
| OPC Portal Server User Guide | ICSTT-RM407 |

## See also

Working with AADvance Applications on page 103

# Working with Trusted Applications 

The AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software is the development environment that enables the creation of one comprehensive project to control a Trusted ${ }^{\circledR}$ controller. The AADvance-Trusted SIS Workstation software enables the creation of Trusted applications supporting multiprocess control. Applications consist of virtual machines running on hardware components. The development process consists of creating a project made up of one device, representing the hardware, which is then downloaded to the Trusted controller. At runtime, the device becomes a virtual machine running on the hardware equipment.

Develop projects with these programming languages from the IEC 61131-3 standard: Functional Block Diagram (FBD), Ladder Diagram (LD), and Structured Text (ST). When building, the compiler produces "target independent code" (TIC).

Develop projects on a Windows ${ }^{\circledR}$ development platform. The AADvanceTrusted SIS Workstation software graphically represents and organizes the device and program organization units (POUs) within a project from many views.

Choose to simulate the running of a project, after building a project, with high-level debugging tools, before actually downloading the device to the Trusted controller. Then connect the AADvance-Trusted SIS Workstation software to monitor real-time information and perform updates. Use a standard Ethernet or serial network to connect the AADvance-Trusted SIS Workstation software with the Trusted controller.

## See also

Creating a Trusted Project on page 515

## Creating a Trusted Project

Create projects with the AADvance-Trusted SIS Workstation software. Also import projects created using the Trusted ${ }^{\circledR}$ Toolset.

Templates available for Trusted projects are:

- Trusted, enables creating a Trusted project containing one device in the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software.
- Import Trusted Toolset Project, enables importing a Trusted Toolset project into the AADvance-Trusted SIS Workstation software.

Since the Trusted run-time is a 16-bit application, the quantity of program organization units (POUs), variables, and I/O modules are directly dependent upon that environment.

For projects, the following properties are defined:

| Info |  |
| :--- | :--- |
| Property | Description |
| Name | Name of the project. Project names are limited to 32 characters <br> beginning with a letter or single underscore followed by letters, digits, <br> and single underscores. The last character for a project name must be a <br> letter or digit; project names cannot end with an underscore character. <br> Names cannot be reserved words, defined words, or data types. |
| Password Protected | Indication that the project is protected by a password controlling its <br> access |
| Path | Complete path where the project is stored on the computer. The path is <br> automatically assigned: <br> \%USERPROFILE\% \My DocumentsISIS Workstation <br> <version> |

Project

| Property | Description |
| :--- | :--- |
| Documentation | Free-form text describing the project |

Security

| Property | Description |
| :--- | :--- |
| Permission Set | The Factory Talk Directory Permission Set associated to the Trusted <br> project. |

Perform the following tasks for projects:

- Accessing the properties of projects
- Building projects
- Cleaning projects
- Downloading project code
- Simulating projects
- Connecting projects
- Copying projects

Perform online updates for projects. Control access for projects by setting a password restricting modifications such as adding programs, functions, and function blocks.

## To create a Trusted project

1. From the File menu, click New Project... (or press Ctrl+Shift+N). The New Project dialog box is displayed.
2. In the New Project dialog box, click the Trusted tab.
3. In the Project Types list, click Trusted, and in the Templates list, click the Create New Project template.
4. Specify a name and comment for the project, and then click OK.

## See also

Import a Trusted Toolset project on page 517
Copy a Trusted project on page 519
Access the properties for a Trusted project on page 519

## To import a Trusted Toolset project

Import projects created with the Trusted ${ }^{\circledR}$ Toolset. Choose to rename the project when importing.

## To import a Trusted Toolset project

1. From the File menu, click New Project... (or press Ctrl+Shift+N).

The New Project dialog box displays.
2. In the New Project dialog box, in the Project Types list, click Import.
3. In the Templates list, click Import Trusted Toolset Project.
4. Specify a name for the project, and then click Browse to select the database path.
5. In the Select Project dialog box, locate and select the project database file (appli.hie), and then click Open.
6. A message may display asking to update the database to the current version. After updating the database, a previous version of the Trusted Toolset can no longer open the project. To continue the importation process, click OK.
7. In the New Project dialog box, click OK.

Tip: AADvance-Trusted SIS Workstation software opens one project at a time. After making changes to a project, the system automatically prompts to save changes before closing the project or opening another.

## See also

## Creating a Trusted Project on page 515

## Copy a Trusted project on page 519

Archive a Trusted project to save a snapshot of its Windows' folder for future restoration if there is data loss. The archived project contains all code, comments, modification history, documentation, and compiled binaries created by the project. Archived project files are compressed and saved with the extension ".taf" as Trusted archive files. Projects of up to 1 GB can be archived.

IMPORTANT Archived projects are not encrypted and should not be used to store or transmit sensitive information over unencrypted protocols.
The Global Library is not included in archived projects. Take appropriate steps to ensure that the restored project is compatible with the Global Library that is used to restore the project. For example, the archived project should be shared along with the Global Library program organization unit (POU).

## To archive a project:

1. Open the project to archive.
2. Select Project > Archive.
3. In Archive Project, specify a name and storage location for the archive file, then select Save.

Note: When a project is archived from a folder with a depth level different than the depth level of the Projects location folder, it must be rebuilt to connect to the physical controllers or to go into simulation.

## See also

Restore a Trusted project on page 518

## Restore a Trusted project

Restore a Trusted project to open a snapshot of its Windows' folder at the time of archival. The restored project contains all code, comments, modification history, documentation, and compiled binaries created by the project at the time of archival.

IMPORTANT Archived projects are not encrypted and should not be used to store or transmit sensitive information over unencrypted protocols.
The Global Library is not included in archived projects. Take appropriate steps to ensure that the restored project is compatible with the Global Library that is used to restore the project. For example, the archived project should be shared along with the Global Library program organization unit (POU).

## To restore a project:

1. Select File > New Project.
2. In New Project, from the Trusted tab, select Project Types > Restore.
3. In Templates, select Restore Project from an Archive, then select Browse.
4. In Select Archive File, navigate to the project archive, then select Open.
5. In New Project, select OK.

Note: When restoring a project that was archived from a folder with a depth level different than the depth level of the Projects location folder, it must be rebuilt to connect to the physical controllers or to go into simulation.
When a project is restored from the Projects location folder, AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software downloads, updates, and connects to the controller without having to build the project.

## See also

Archive a Trusted project on page 517

## Copy a Trusted project

## To copy a Trusted project

1. In the Application View, select the project.
2. From the File menu, click Save <Project> As....
3. In the Save Project As dialog box, do the following:
a. Specify a name for the project.
b. (optional) Click Browse... to specify the database path, and then click OK.
c. Click Save.

The selected project is copied and renamed.

## See also

Creating a Trusted Project on page 515
Import a Trusted Toolset project on page 517
Access the properties for a Trusted project on page 519
Modify the Documentation property for projects. All other project properties are read-only.

## To access the properties for a Trusted project

- In the Application View, right-click the project, and then click Properties.
The Properties window displays the properties for the project.


## See also

## Creating a Trusted Project on page 515

This chapter provides an overview of the main security features available for the AADvance-Trusted SIS Workstation software that protect the Trusted project content and Trusted controller equipment:

- FactoryTalk ${ }^{\circledR}$ Security software provides permission-based control on actions that affect the Trusted controller equipment and the Trusted project.
- AADvance-Trusted SIS Workstation software project access control protects the Trusted project.
Create a copy to rename a project.

Permission-based control

## To access the properties for a Trusted project

## Project Security Introduction

FactoryTalk Security integrates a common security model across all FactoryTalk ${ }^{\circledR}$ products. FactoryTalk ${ }^{\circledR}$ Services Platform (FTSP) includes the FactoryTalk ${ }^{\circledR}$ Administration Console that provides the interface for configuring your system. FactoryTalk Services Platform software, available from PCDC, is installed independently of AADvance-Trusted SIS Workstation
software. Use FactoryTalk Security to control actions in your projects with centrally managed, role-based policy enforcement.

IMPORTANT Before you use this feature, read the FactoryTalk Security System Configuration Guide, which is available in the Rockwell Automation Literature Library: rockwellautomation.com/en-us/support/documentation/literature-library.htm|

For AADvance-Trusted SIS Workstation software project security, set the project access control by using a project password.

## See also

Use FactoryTalk Security with AADvance-Trusted SIS Workstation software

AADvance-Trusted SIS Workstation software and FactoryTalk Security Securable actions supported by FactoryTalk Security

## Use FactoryTalk Security with AADvance-Trusted SIS Workstation software

The FactoryTalk Security software allows you to control actions on your projects based on:

- the user who is signed in.
- the project the user is attempting to modify.
- the computer from which the user is attempting to modify the project.

Tip: The security feature is intended to prevent accidental unauthorized action to your project and to Trusted controller equipment. It is important to note that while the feature does provide some protection against intentional unauthorized action, it does not provide protection against action conducted by system hackers. You should put in place additional preventative measures against unwanted actions.

FactoryTalk Security grants or denies action based on:

- User ID
- Computer name
- Action name (that is, the activity the user is trying to perform, such as configure I/O devices, or download an application)

IMPORTANT Before you use this feature, read the FactoryTalk Security System Configuration Guide, which is available in the Rockwell Automation Literature Library: rockwellautomation.com/en-us/support/documentation/literature-library.htm|
Tip: AADvance-Trusted SIS Workstation software startup might open slowly when FactoryTalk Services version 6.31 .00 or later is installed. Users who do not plan to use the permission-based control feature are therefore advised not to install it.
If FactoryTalk Services version 6.31 .00 or later must be installed, users are advised to deactivate the Use single sign-on security policy in the FactoryTalk Administration Console.

## Security policy considerations

## AADvance-Trusted SIS Workstation software and FactoryTalk Security

When you use FactoryTalk Administration Console to configure securable actions, security settings are obtained from the FactoryTalk Network Directory.

When used with the AADvance-Trusted SIS Workstation software application, FactoryTalk Security supports securable actions and permission sets. You can configure these FactoryTalk Security settings in the FactoryTalk Administration Console.

In the AADvance-Trusted SIS Workstation software application:

- FactoryTalk Directory provides the security settings.
- It is not possible to use the FactoryTalk Local directory.
- FactoryTalk Services Platform version 6.31 or later supports associating the project with a specific FactoryTalk Directory.
- You can add, change, or remove the permission set in Trusted projects with AADvance-Trusted SIS Workstation software project properties.
- When you archive and restore a project in AADvance-Trusted SIS Workstation software, the permission set associated with the project is maintained.
- You can log on to or log off of FactoryTalk from AADvance-Trusted SIS Workstation software. When the single sign-on property is enabled in FactoryTalk security policy, AADvance-Trusted SIS Workstation software automatically initiates a sign-in attempt with the Windows user account.

Permission sets identify a set of actions that are granted or denied for user groups or computer groups. Use permission sets to define permissions in the FactoryTalk Administration Console and to apply the same permissions to multiple projects.

IMPORTANT Users could lose (or gain) permission to open or edit a project, if you change the permission set. Always check the Effective Permissions in FactoryTalk Administration Console to ensure that users have the appropriate access. Before you use these features, read the FactoryTalk Security System Configuration Guide, which is available in the Rockwell Automation Literature Library: rockwellautomation.com/en-us/support/documentation/literature-library.html

Users must log off and on to AADvance-Trusted SIS Workstation software again to apply the changes made in FactoryTalk Administration Console.

When a user opens a project that is secured with a permission set, the AADvance-Trusted SIS Workstation software application checks the Security Authority Identifier, which represents the FactoryTalk Directory ID, to see if it matches the ID in the project.

- A matching ID causes the AADvance-Trusted SIS Workstation software to check the FactoryTalk Security server for permission sets associated with the project, and to retrieve the permissions for the current user and computer combination.
- An unmatched ID prevents the project from opening.

IMPORTANT Projects that are secured and bound to a specific Security Authority cannot be recovered if the security authority identifier of the FactoryTalk Network Directory used to secure the project no longer exists. For more information on backing up a FactoryTalk Directory, see Back up a FactoryTalk system in the FactoryTalk Services Platform online help.

## Securable Actions supported by FactoryTalk Security

Configure the securable actions listed here in the Network Directory by using the FactoryTalk Administration Console. For details, see the FactoryTalk Security System Configuration Guide which is available in the Rockwell Automation Literature Library.

## Rockwell Automation Literature Library



Tip: The variable selector normally opens with a data type filter and an empty row at the bottom of the grid. Selecting the empty row clears the filter. The securable actions Project: Modify Global Variables, Project: Modify POUs, and Project: Modify Global Library set to Deny might influence the variable selector behavior. Clear the filter to see the variables in the variable selector. For instructions on how to reset the filter, see Variable Selector on page 40.

These are the FactoryTalk Security securable actions.

| Action | Description |
| :--- | :--- |
| Application: Verify, Build or Clean | Verify a POU syntax, build an application, or clean an <br> application. |
| Controller: Connect | Connect to a controller. |
| Controller: Download | Download an application. |
| Controller: Lock/Unlock Variables | While connected, lock or unlock the value of a variable. |
| Controller: Modify Cycle Timing | Edit the cycle timing. |
| Controller: Open Intelligent Update Manager | Open the Intelligent Update Manager tool. |
| Controller: Open System Configuration Tool | Open the System Configuration Tool. |
| Controller: Override Variables | While connected, force the value of a variable. |
| Controller: Spy Lists | Monitor a list of variables. |
| Controller: Start/Stop | Start or stop an application. |
| Controller: Update | Update an application. |
| Project: Configure I/0 Devices | Edit instances of I/O devices and edit the wiring of a <br> channel. |
| Project: Configure ISaVIEW | Edit an ISaView document. <br> When ISaVIEW securable Action is set to Deny, ISaVIEW <br> objects/shapes properties are not visible from the Property <br> Window and the entire document is read-only. |
| Project: Configure Password | Add, change, or remove a password on a project. |
| Project: Configure Permission Set | Select, or remove a permission set associated with a <br> project. |
| Project: Configure PoUs | Add, delete, rename, reorder, or import a PoU in the device <br> or in the global library. |
| Project: Modify Communication Parameters | Modify communication properties. |
| Project: Modify Compiler Options | Modify compiler options. |
| Project: Modify Conversion Tables | Modify conversion tables. |
| Project: Modify Description | Edit the project description. |
| Project: Modify Global Defined Wobal Library | Edit global defined words. |
| Project: Modify Global Variables | Modify the global library. |


| Action | Description |
| :--- | :--- |
| Project: Modify POUs | Edit a POU in the device or in the global library, including: <br> Local variables <br> Local defined words <br> Editors: Structured Text, Ladder Diagram, or Function <br> Block Diagram. <br> Diary <br> Variables properties <br> Import local variables |
| Project: Modify Runtime Parameters | Modify application run-time options. |
| Project: Modify SOE/Modbus Extended Attributes | Configure SOE or Modbus attributes for a global variable. |
| Project: Open | Open the project. |
| Project: Rename Device | Rename a target device. |

## Trusted Devices

## A device corresponds to a Trusted controller.

Device names are limited to eight characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Names cannot end with an underscore. Names cannot be reserved words, defined words, or data types.

## Perform these tasks for devices:

- Setting the properties of devices
- Renaming devices
- Building devices
- Cleaning devices
- Downloading devices
- Simulating devices
- Connecting devices
- Updating devices
- Displaying diagnostic information for devices

For devices, specify these properties:

| Application Run-time Options | Description |
| :--- | :--- |
| Cycle Timing (ms) | Amount of time, in milliseconds, allocated to each cycle. If a cycle is <br> completed within the cycle timing period, the system waits until this period <br> has elapsed before starting a new cycle. The cycle consists of scanning the <br> physical inputs of the process to drive, executing the POUs of the device, then <br> updating physical outputs. The Trusted controller executes the application <br> code in real time. When the cycle time is set to 0, the controller starts the next <br> cycle immediately after completing the last cycle. <br> The default cycle time is 0 ms. |
| Detect Errors | Store errors. Define the Nb Stored Errors option. |
| Memory for Retain | Size of the memory space reserved for storing the values of retained <br> variables. The values of these variables are stored in this memory at the end <br> of each cycle for use if the target is restarted. |
| Nb Stored Errors | Number of entries, that is, the size of the queue (FIFO) in which detected <br> errors are stored. |
| Compiler Options | Description |
| Generate Debug Information | Generate information required for debugging using step-by-step execution. |
| Miscellaneous | Description |
| Last Compilation Date | Date of the last compilation (build). |


| Last Compiled Version | Version number of the last compiled application image. |
| :--- | :--- |
| Serial Port Connection <br> Information | Description |
| Baud Rate | Data transmission speed, defined in bits per second. Possible values are 0, <br> 600, 1200, 2400, 4800, 9600, and 19200; the default value is 0. A value of 0 <br> indicates no change. |
| Data Bits | Number of data bits in a byte. Possible values are 7 or 8. |
| Hardware Flow Control | AADvance-Trusted SIS Workstation software controls the CTS and DSR lines to <br> enable hardware handshaking during exchanges. |
| Parity | Parity type. Possible values are None, Odd, Even, Mark, or Space; the default <br> value is None. |
| Stop Bits | Length of the stop bit. Possible values are 0,1, 1.5, or 2. |
| Shared Connection | Description |
| Information | Enables the ISA68M target. |
| Enable ISA68M | Enables simulating the application. <br> Enable Simulation <br> Network used for communication. Possible values are Serial and TCP/IP. The <br> detwork <br> Number of trises for communication using a serial connection. <br> RetrySerial port used for communications. Possible values are Simulator, Com1, <br> Com2, Com3, Com4, Com5, Com6, Com7, Com8, Com9, Com10, Coml1, User, or <br> Ip. |
| Serial Port or Port Category |  |
| Communication timeout, expressed in seconds. |  |
| Timeout(S) | Description |
| Information |  |


| Task | Procedure |
| :--- | :--- |
| Set properties for a device | $\bullet$ In the Application View, right-click the device, and then select Properties. |
| Rename a device | Renaming a device activates an automatic saving of the project. <br> $\bullet$ In the Application View, right-click the device, then click Rename, and then type a name for the device. |

## See also

## Generating Code on page 554

Run an Application Online on page 556
Access diagnostic information on page 559
Online Updates on page 567
Trusted Libraries on page 531

## Enable power flow debugging

Use power flow debugging to follow the flow of logic during simulation or online debugging of LD/FBD programs. Power flow debugging provides a quick visual representation of signals.

By default, a red power flow line indicates True, while a blue power flow line indicates False.

Turn on power flow debugging for the Trusted device.

Tip: While power flow debugging is enabled by default for Trusted Toolset projects, it is disabled by default in AADvance-Trusted SIS Workstation software.

## To enable power flow debugging

1. In the Application View, right-click the device and then select Properties.

The Properties window opens.
2. Expand Compiler Options.
3. Set Generate Debug Info to TRUE.
4. In the menu, select File > Save Device.

The power flow setting is saved.

## Programs

## See also

Trusted Devices on page 523
Define programs in the Programs section of the device in the Application View. For programs, specify these properties:

| Property | Description |
| :--- | :--- |
| Comment | Single line of text describing a program. The description displays in the <br> Properties window. |
| Description | Free-form text (RTF) describing a program. The description displays in <br> the Description window. |
| Language | (Read-only) Programming language of the program organization unit <br> (PoU) |
| Name | Name of the program. Program names are limited to eight (8) characters <br> beginning with a letter or single underscore followed by letters, digits, <br> and single underscores. Do not use reserved words, defined words, or <br> data types for names. Use unique names for POUs within a project. |
| Type | (Read-only) Type of POU. Possible values are program, user-defined <br> function, or user-defined function block |

Perform these tasks for programs:

| Task | Procedure |
| :--- | :--- |
| Add programs | - In the Application View, right-click the Programs element for the device, point to Add Programs, and then click one of these: <br> • New FBD: Function Block Diagram <br> • New LD: Ladder Diagram <br> $\bullet$ - New ST: Structured Text |
| Set properties for a program | In the Application View, right-click the program, and then click Properties. |
| Copy and paste programs | Copying and pasting a program includes local variables and local defined words. <br> 1. In the Application View, right-click the program, and then click Copy. <br> 2. Right-click the Programs element, and then click Paste <br> Tip: POU comments are not copied during a copy and paste operation. |
| Rename programs | In the Application View, right-click the program, then click Rename, and then type a name for the program. |
| Delete programs | In the Application View, right-click the program, and then click Delete. <br> Tip: $\quad$When deleting a program, save the project before making subsequent <br> changes. |

## See also

## Verifying POU Syntax on page 555

Export a program on page 526
Add a program from an exchange file on page 526
Name conventions and limitations on page 564
Export programs as exchange files (*.txf) to import them into different projects. Importing exchange files enables reusing programs in different Trusted ${ }^{\circledR}$ projects. Export individual programs.

Tip: Exporting a program, function, or function block exports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\circledR}$ extended attributes are not exported.

## To export a program

1. In the Application View, right-click the program, and then click Export...
2. In the Save Program As dialog box, browse for the location to store the Trusted exchange file, and then click Save. The program is exported as an exchange file.

## See also

Add a program from an exchange file on page 526
Programs on page 525
Import programs previously exported as exchange files (*.txf). Importing exchange files enables reusing programs in different Trusted ${ }^{\circledR}$ projects.

Tip: Importing a program, function, or function block imports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\ominus}$ extended attributes are not imported.

When importing a program organization unit (POU), specify a unique name within the project.

## To add a program from an exchange file

1. In the Application View, right-click Programs, point to Add Programs, and then click Existing Program....
Tip: Alternatively, add an existing program from File > Add > Existing Program...
2. In the Select Exchange File dialog box, locate the Trusted exchange file containing the program, and then click Open. The program is imported into the project.

## See also

## Export a program on page 526

## Programs on page 525

## Functions

Define functions in the Functions section of the device in the Application View.

For functions, specify these properties:

| Property | Description |
| :--- | :--- |
| Comment | Single line of text describing a function. The description displays in the <br> Properties window. |
| Description | Free-form text (RTF) describing a function. The description displays in the <br> Description window. |
| Language | (Read-only) Programming language of the program organization unit (POU) |
| Name | Name of the function. Function names are limited to eight (8) characters <br> beginning with a letter or single underscore followed by letters, digits, and <br> single underscores. Names cannot be reserved words, defined words, or <br> data types. Use unique names for POUs within a project. |
| Type | (Read-only) Type of POU. Possible values are program, user-defined <br> function, or user-defined function block |

Perform these tasks for functions:

| Task | Procedure |
| :--- | :--- |
| Add a function | 1. In the Application View, right-click the Functions element, point to Add Functions, and then click the required programming <br> language for the function. |
| 2. To define the parameters for the function, right-click the function, and then click Parameters. |  |

Tip: When adding functions, define parameters. Functions can have a maximum of 32 parameters ( 31 inputs and one output). When defining parameters, consider these limitations:

- Parameter names are limited to 32 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. Use unique names for the parameters of a function.
- Possible data types for parameters are BOOL, DINT, REAL, TIME, MESSAGE
- For Message type variables, string capacity is limited to 255 characters


## See also

Verifying POU Syntax on page 555
Export a function on page 528
Add a function from an exchange file on page 528
Name conventions and limitations on page 564

## Export a function

## Add a function from an exchange file

Export functions as exchange files (*.txf) to import them into different projects. Importing exchange files enables reusing functions in different Trusted ${ }^{\circledR}$ projects. Export individual functions.

Tip: Exporting a program, function, or function block exports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\circledR}$ extended attributes are not exported.

## To export a function

1. In the Application View, right-click the function, and then click Export....
2. In the Save Function As dialog box, browse for the location to store the Trusted exchange file, and then click Save. The function is exported as an exchange file.

## See also

Add a function from an exchange file on page 528
Functions on page 527
Import functions previously exported as exchange files (*.txf). Importing exchange files enables reusing functions in different Trusted ${ }^{\circledR}$ projects.

Tip: Importing a program, function, or function block imports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\oplus}$ extended attributes are not imported.

When importing a program organization unit (POU), specify a unique name within the project.

## To add a function from an exchange file

1. In the Application View, do one of the following:

- To add a function to the Functions section, right-click Functions, point to Add Functions, and then click Existing Function....
- To add a function to a program or function, right-click the existing program or function, point to Add, and then click Existing Function....
Tip: Alternatively, add an existing function from File $>$ Add $>$ Existing Function...

2. In the Select Exchange File dialog box, locate the Trusted exchange file containing the function, and then click Open. The function is imported into the project.

## See also

Export a function on page 528

## Function Blocks

## Functions on page 527

Define function blocks in the Function Blocks section of the device in the Application View.

For function blocks, specify these properties:

| Property | Description |
| :--- | :--- |
| Comment | Single line of text describing a function block. The description displays <br> in the Properties window. |
| Description | Free-form text (RTF) describing a function block. The description <br> displays in the Description window. |
| In-line | For Ladder Diagram (LD) function blocks, duplicates the code of a <br> function block for each instance used in the application, providing a <br> faster execution time. In-line LD function blocks increase the size of the <br> application TIC code. <br> To use an in-line LD function block in another LD function block, known <br> as nested blocks, set the In-line property for both blocks to True. <br> - Set the In-line property to False for LD function blocks that generate <br> an undeclared identifier error. <br> - Set the Generate debug information property to False if a build error <br> occurs when the In-line property is True for LD function blocks. |
| Language | (Read-only) Programming language of the program organization unit <br> (PoU) |
| Name | Name of the function block. Function block names are limited to eight <br> (8) characters beginning with a letter or single underscore followed by <br> letters, digits, and single underscores. Names cannot be reserved words, <br> defined words, or data types. Use unique names for PoUs within a <br> project. |
| Type | (Read-only) Type of POU. Possible values are program, user-defined <br> function, or user-defined function block. |

Perform these tasks for function blocks:

| Task | Procedure |
| :--- | :--- |
| Add function blocks | 1. In the Application View, right-click the Function Blocks element, point to Add Function Blocks, and then click the required <br> programming language for the function block. <br> 2. To define the parameters for the function block, right-click the function block, and then click Parameters. |
| Set properties of function blocks | In the Application View, right-click the function block, and then click Properties. |
| Copy and paste function blocks | Copying and pasting a function block includes local variables and local defined words. <br> 1. In the Application View, right-click the function block, and then click Copy. <br> 2. Right-click the Function Blocks element, and then click Paste. <br> Tip: PoU comments are not copied during a copy and paste operation. |
| Rename function blocks | In the Application View, right-click the function block, then click Rename, and then type a name for the function block. |
| Delete function blocks | In the Application View, right-click the function block, and then click Delete. <br> Tip: When deleting a function block, save the project before making subsequent <br> changes. |

Tip: When adding function blocks, define parameters. Function blocks can have
a maximum of 32 parameters (inputs and outputs). When defining
parameters, consider these limitations:

- Parameter names are limited to 32 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. Use unique names for the parameters of a function block.
- Possible data types for parameters are BOOL, DINT, REAL, TIME, MESSAGE
- For Message type variables, string capacity is limited to 255 characters


## See also

## Verifying POU Syntax on page 555

Export a function block on page 530
Add a function block from an exchange file on page 530
Name conventions and limitations on page 564

## Export a function block

## Add a function block from an exchange file

Export function blocks as exchange files (*.txf) toimport them into different projects. Importing exchange files enables reusing function blocks in different Trusted ${ }^{\circledR}$ projects. Export individual function blocks.

Tip: Exporting a program, function, or function block exports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\ominus}$ extended attributes are not exported.

## To export a function block

1. In the Application View, right-click the function block, and then click Export....
2. In the Save Function Block As dialog box, browse for the locationto store the Trusted exchange file, and then click Save. The function block is exported as an exchange file.

## See also

Add a function block from an exchange file on page 530
Function Blocks on page 529
Import function blocks previously exported as exchange files (*.txf). Importing exchange files enables reusing function blocks in different Trusted ${ }^{\circledR}$ projects.

Tip:
Importing a program, function, or function block imports its logic, local variables, local defined words, and description. Program, function, and function block comments, Diary files, and Modbus ${ }^{\circledR}$ extended attributes are not imported.

When importing a program organization unit (POU), specify a unique name within the project.

## To add a function block from an exchange file

1. In the Application View, right-click Function Blocks, point to Add Function Blocks, and then click Existing Function Block....
Tip: Alternatively, add an existing program from File > Add > Existing Function Block...
2. In the Select Exchange File dialog box, locate the Trusted exchange file containing the function block, and then click Open. The function block is imported into the project.

## See also

Export a function block on page 530
Function Blocks on page 529

## Variables

Define variables for their scope. For instance, global variables are available for use throughout the programs, functions, and functions blocks of a device. Whereas, variables defined for a program, a function, or a function block are local to that element. Define variables in the Variables grid of the Dictionary. For individual variable scopes, import and export variables data having the Microsoft Excel (*.xls) or comma-separated values (*.csv) format.

Variable names are limited to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Names cannot be reserved words or data types. Names must be unique.

## See also

Variables Grid on page 615
Import and Export Variables Data on page 534
Variables on page 738
Name conventions and limitations on page 564

## Trusted Libraries

Each Trusted ${ }^{\circledR}$ project contains one global library. The global library is made up of functions and function blocks available for reuse throughout a Trusted project. Using the IEC 61131-3 languages, you can write functions (FBD, LD, or ST) and function blocks (FBD, LD, or ST). You cannot remove the global library from a project.

Library functions and function blocks must have unique names within a project. When these have the same names as those defined in a project in which the functions or function blocks are used, only those from the project are recognized.

Library functions and function blocks are compiled with the project. However, you can verify the syntax for the Global library.

Tasks to perform with libraries are:

| Task | Procedure |
| :--- | :--- |
| Add a function or function block to the library | 1. In the Application View, expand Libraries, then Global, and then right-click the Functions or Function Blocks <br> element, and then do one of these: |
|  | -Right-click the Functions element, then point to Add Functions, and then click the required programming language. <br> programming language. <br>  <br>  <br> 2. To define the parameters for the function or function block, right-click the required element, and then click <br> Parameters. |


| Task | Procedure |
| :--- | :--- |
| Copy and paste function or function block | Copy and paste functions and function blocks between the Global library and the project, and vice versa. <br> 1. In the Application View, right-click the function block in the Global library, and then click Copy. <br> 2. Right-click the respective section of the library or project, and then click Paste. |
| Export a function or function block | Export functions or function blocks as exchange files (*.txf) toimport them into projects or global libraries on <br> different workstations. <br> Tip: $\quad$Exporting a function or function block also exports its local <br> variables and local defined words. <br> In the Application View, right-click the function or function block, and then click Export.... |
| 1. In the Save As dialog box, browse for the location to store the Trusted exchange file, and then click Save. The |  |
| function or function block is exported as an exchange file. |  |

## See also

Opening a Trusted Project on page 532
Closing a Project on page 120
Functions on page 527
Function Blocks on page 529
Global Library Functions and Function Blocks on page 823

## Opening a Trusted Project

When opening a project, choose from a list of recently opened projects or browse to locate a required project.

| Task | Procedure |
| :---: | :---: |
| Open a Trusted project from AADvance ${ }^{\circledR}$-Trusted ${ }^{\ominus}$ SIS Workstation software | 1. From the File menu, click Open Project (or press Ctrl+Shift+0). <br> 2. In the Open Project dialog box, click the Trusted tab, and then do one of these: <br> - In the list of recent projects, select the required project, and then click OK. <br> - Click Browse... to locate the project folder, select the required *.sissin file, and then click Open. |
| Open a Trusted project from the Projects directory | - From the Windows ${ }^{\oplus}$ Explorer, access the Projects directory and perform one of these: <br> - Double-click the required *.sissin file. <br> - Drag the *.truproj or *.sissIn file onto the desktop AADvance-Trusted SIS Workstation software icon. |

Tip:
The project might take more time to open when the permission set is used in the project.

Tip: When opening a password-protected project, enter the correct password to gain access to the project.

## See also

Trusted Libraries on page 531
Closing a Project on page 120
Setting Project Access Control on page 533

# Closing a Project 

When closing a project, the AADvance-Trusted SIS Workstation software prompts to save changes made to the project.

## To close a project

1. From the File menu, click Close Project.
2. When prompted to save changes, click Yes.

## See also

## Opening a Trusted Project on page 532

## Setting Project Access Control

For project security, set access control using a password for projects. Password definitions are limited to eight characters and can consist of letters, digits, and symbols. Enter the password to open a password-protected project.

In the Application View, the following indicates the security state:
B A lock is applied to the project.
Edit existing passwords for projects or remove existing passwords. When copying, pasting, and importing projects having access control, password definitions are retained.

| Task | Procedure |
| :--- | :--- |
| Set a password | 1. In the Application View, right-click the project, and then click Set Password. <br> 2. In the Set Password dialog box, enter the required information, and then click OK. <br> 3. In the Password field, type the required password. <br> 4. In the Confirm Password field, re-type the required password. |
| Edit a password | 1. In the Application View, right-click the project, and then click Set Password. <br> 2. In the Change Password dialog box, enter the required information, and then click OK. <br> a. In the Old Password field, type the current password. <br> b. In the New Password field, type the required password. <br> c. In the Confirm Password field, re-type the required password. |
| Remove a password | 1. In the Application View, right-click the required element, and then click Set Password. <br> 2. In the Change Password dialog box, enter the required information, then click OK. <br> - In the Old Password field, type the current password. <br> - The New Password and Confirm Password fields must remain blank. |

## See also

Opening a Trusted Project on page 532

# Import and Export Variables Data 

Imported variables are saved as Microsoft ${ }^{\circledR}$ Excel ${ }^{\circledR}$ spreadsheets (.xls) or comma-separated values (.csv). Exporting variables enables managing variables data including adding, removing, and modifying variables. Import variables data files into other devices and programs in the same project or in other projects.

Tip: When importing a variables data file missing any columns, the AADvance ${ }^{\oplus}$ Trusted ${ }^{\circledR}$ SIS Workstation software uses the default values for the missing data.

When exporting variables, specify the location in which to save the exported files. To develop the contents of variables data files (*.xls or *.csv) in a respective editor, for instance, Microsoft Excel or Notepad, export an empty file serving as a template.

A variables data file includes a header row, a mapping row, a version number, and the variables data. The header row displays the names of the data columns. The column names of the header row are the various variable properties. The mapping row displays the internal names of data columns in brackets used for processing. An automatically generated version number indicates the version of the import/export feature. The individual variable data is placed in the respective columns.


This table indicates the syntax used in the variables data files for the variable properties and the associated internal names:

| Variable Property | Internal Name | Description |
| :--- | :--- | :--- |
| Name | (Name) | Name of the variable. Variable names are limited to 32 characters beginning with a letter or <br> single underscore character followed by letters, digits, and single underscore characters. <br> Names cannot be reserved words, defined words, or data types. Names must be unique for <br> the same type of elements within a scope. |
| Data Type | (DataType) | Data type of the variable |
| String Size | (StringSize) | The maximum character length for string-type (MESSAGE) variables. The string length is <br> limited to 240 characters. |
| Wiring | (Wiring) | Indicates the I/O channel wired to the variable |
| Attribute | (Dtribute) | Read and write access rights <br> For the local variables of a program, indicates the variable is internal. <br> For the local variables of a function or function block, indicates whether the variable is <br> internal, an input, or an output. <br> For the global variables of a device, indicates the variable is internal, for use with I/0 wiring, <br> or defined for an I/O channel. |
| Modbus Address | ModbusAddress) | Modbus address of the variable |


| Variable Property | Internal Name | Description |
| :--- | :--- | :--- |
| Message True | (MsgTrue) | User-defined text that is used by the SOE Collector in the Trusted Sequence of Events and <br> Process Historian Package when the value of a Boolean variable data type is TRUE. <br> Limited to eight Windows-1252 characters excluding: , \% " <br> All leading and trailing spaces are automatically removed. |
| Message False | (MsgFalse) | User-defined text that is used by the SOE Collector in the Trusted Sequence of Events and <br> Process Historian Package when the value of a Boolean variable data type is FALSE. <br> Limited to eight Windows-1252 characters excluding: , \% " <br> All leading and trailing spaces are automatically removed. |
| Retained | Indicates whether the virtual machine saves the value of the variable at each cycle |  |
| Initial Value | (Inetained) | Value held by a variable when the virtual machine starts executing the application |
| Unit | (Unit) | A string identifying the physical unit of measure |
| Comment | (Comment) | Free-format text limited to 60 single-byte characters for variables |

A progress bar indicates the advancement of import and export operations. The progress bar enables canceling import and export operations. For import operations, the process stops after importing the last variable in progress. For export operations, the process does not produce an exported variables file.

| Task | Procedure |
| :--- | :--- |
| Import variables | 1. In the Application View, right-click the destination receiving the Local Variables or Global Variables, and then click Import <br> Variables. <br> Tip: <br> - If you import a variable to which a conversion table applies, that variable will no longer be associated with the conversion <br> table. <br> - If you import a variable for which a Modbus <br> - If you import a variable that is in the SOE collection, that variable will be removed from the SOE collection. |
| 2. From the Import Variables dialog box, select the file type containing the variables, locate the file, and then click Open. |  |

## See also

## Variables Grid on page 615

## Use Modbus Communications

A Trusted ${ }^{\circledR}$ controller supports Modbus ${ }^{\circledR}$ communications and acts as a Modbus master or Modbus slave. A Modbus master can have Modbus master slaves. The Trusted controller uses either Ethernet or serial ports for Modbus communications.

Configure Modbus communications in the T8151 Communications Interface. For more information on configuring Modbus communications, refer to the Product Description PD-T8151B, Trusted Communication Interface.

Tip: Before using Modbus communications, specify the Modbus address format (Hexadecimal or Decimal) in the CAM Trusted settings from the Tools > Options menu. The System Configuration Tool reads Modbus addresses for variables in Decimal format.

T8151 Trusted Communications Interface Module Parameters

## T8151 Communications Interface



When configuring Modbus communications, define these properties:

| Property | Description |
| :--- | :--- |
| TCP/IP | The Ethernet ports are listed as TCP/IP 0 and TCP/IP 1 but are described on the front panel as Ethernet 1 and 2. Each port has an |
|  | IP address and a subnet mask. The subnet mask defines the network address portion of the whole IP address. Devices on one |
| network address cannot communicate with devices on other networks without a gateway. |  |
|  | Set the Ethernet port IP addresses for both ports on a communication interface to separate networks. For example, the IP |
|  | addresses covered by the subnet mask should be different. If port 1 and 2 are on the same network, only one port can |
|  | communicate. |
|  | Examples |
|  | - Subnet Mask = 255.255.255.0 in decimal |
|  | This equals 11111111.11111111.1111111.00000000 in binary. The 1's in the subnet mask indicate the location of the network address |
|  | portion, for example, any number in the first three places. |
|  | This allocates all of the first three numbers in the IP address to the network address. Therefore, IP addresses 1.2.3.123 and |
|  | 1.2.3.231 are on the same network, but 1.2.4.123 is on network 1.2.4.x. |
|  | - Subnet Mask = 255.255.128.0 in decimal |
|  | This equals 1111111.1111111.10000000.00000000 in binary. |
|  | This allocates the first two numbers in the IP address and the highest bit of the next number to the network address. |
|  | Therefore, 1.2.128.1 and 1.2.129.1 are on the same network but 1.2.127.1 is not. |


| Property | Description |
| :---: | :---: |
| Default Gateway | The address of a device that allows access to other networks. Since both ports must be on different networks, only one of the ports can be on the same network as the gateway, and only this port can use the gateway. The gateway allows the port to communicate with IP addresses outside its own network, for example, across a site local area network (LAN) or even onto the Internet, perhaps to allow remote diagnostics. <br> Example <br> A communications interface Ethernet port is connected to network 192.200.11.x. <br> - Subnet Mask $=255.255 .255 .0$ in decimal <br> - Default Gateway $=192.200 .11 .1$ in decimal <br> - Communications interface Ethernet port $=192.200 .11 .202$ in decimal <br> The communications interface can communicate with other devices on the 192.200.11.x network. To communicate with a device on another network such as 192.200 .81 .145 , set Default Gateway to match the network gateway address of 192.200.11.1. The remote device can transmit and receive packets from the communication interface. |
| Multicast | For information on configuring Peer-to-Peer multicasting, refer to the Product Description PD-T8151B, Trusted Communication Interface. |
| Serial Ports | Define the four serial ports for baud rate, data bits, parity, stop bits, and line type. The front panel serial port is port 0 and cannot be defined. <br> The normal pattern uses two bits after each data byte. This pattern is either odd/even parity and one stop bit or no parity and two stop bits. <br> - RS232 is only available on serial ports 1 and 2. The setting for this mode is rs232. <br> - rs485fd configures a four wire point-to-point connection. <br> - rs485hdmux configures a two wire multi-dropped connection. rs485 performs the same function. <br> - rs485fdmux configures a four wire multi-dropped connection. |
| Additional | Enter configuration commands not supported by the System Configuration Tool. |

## Prerequisites

- Before configuring Modbus communications, create variables with Modbus addresses and (optional) extended attributes. Configure Modbus communications in the T8151 Communications Interface.


## To configure Modbus communications

1. From the VIEW menu, click System Configuration Tool.
2. In the Trusted System Configuration Tool window, right-click an empty slot and select T8151: Communications Interface.

The slot displays a communications interface.
3. Click the communications interface in the slot.

The T8151Trusted Communications Interface Module Parameters window appears.
4. Define the properties for the Modbus communications:

- TCP/IP Ethernet ports
- Default gateway
- Multicast for enhanced peer-to-peer
- Serial ports

5. Configure Modbus master and slave protocols for use in the communication services for controllers.
6. Define additional configuration commands not supported by the System Configuration Tool.

## See also

Configure a Modbus Slave on page 538
Configure a Modbus Master on page 541
Configure a Slave for a Modbus Master on page 547
Set extended attributes for Modbus collection on page 622
Specify Trusted settings on page 77

## Configure a Modbus Slave

Configure up to 10 Modbus ${ }^{\circledR}$ slaves to operate in the T8151 Communications Interface. Allocate these slaves to serial or Ethernet ports.

Since a Trusted ${ }^{\circledR}$ system that is Modbus slave has no control over the messages received from the master, the system can have longer than expected delays between messages. Allocate more time for communications processing between the T8151 Communications Interface and the processor by specifying a sleep period between scans in the Trusted system configuration. To set a sleep period between scans, in the T8110 Trusted TMR Processor configuration, set the sleep period to approximately 32 ms longer than the current scan time from the Toolset Debugger window. This longer sleep period provides time on most scans for communications tasks and evens out the scan time. It also lengthens the scan time while ensuring that process safety times are not compromised and the maximum scan time and I/O
module watchdog times are adjusted proportionately to the change in average scan time.


When configuring a Modbus slave, define these properties:

| Property | Description |
| :--- | :--- |
| Slave Address | Slave address for communication with the T8151 Communications Interface. The address is usually 1 on a point-to-point link in the <br> range 1 to 127. |
| Communications | The type of connection between the Modbus slave and the Modbus master: <br> - Initiate TCP/IP Connection, slave initiates the connection to the Modbus master. <br> - Wait TCP/IP Connection, slave waits for an incoming connection with Modbus master (from the remote end) <br> - Serial port. A serial port can only be used for Modbus slave or Modbus master but not both. <br> A slave that appears on an Ethernet port is packaged in Internet Protocol (IP). Establish the IP layer before delivering Modbus data. The <br> possible TCP/IP connections are either the T8151 Communications Interface sets up a TCP/IP connection or the T8151 Communications <br> Interface waits for a call from the remote end. This connection is probably to a terminal server that requires trial and error and a close <br> look at the terminal server configuration to determine whether it expects to set up or wait for connection. The default port used for <br> Modbus on the T8151 Communications Interface is 2000; this port only changes if port 2000 is used elsewhere. If the to determine <br> whether it expects to set up or wait for connection. The default port used for Modbus on the T8151 Communications Interface initiates <br> the connection, enter the IP address of the remote end. |


| Property | Description |
| :--- | :--- |
| Connection Timeout | The time delay to allow Ethernet-based slaves to recover after a lost connection. This property should remain at the default value to <br> disconnect after 300 seconds. This time delay is appropriate for communications where cable breaks are not normally expected, for <br> example, not involving a modem or other link likely to disconnect. <br> The TCP/IP stack does not detect the loss of Modbus communications caused by a pulled connection or cable break. The TCP/IP stack <br> does not detect a broken connection and retains it as a valid connection. When the broken connection is recovered, the Modbus Master <br> may resume communications using the existing connection but is more likely to establish a new connection. Each re-connection <br> consumes one of the available connections. If re-connections occur on a regular basis, all available connections are consumed and <br> connections are no longer possible. On early communications interfaces that do not recognize the connection timeout setting, <br> removing and reinserting the communications interface and rebooting recovers the available connections. |
| SOE Over Modbus | The protocol that allows buffered transfer of events via a T8151 Communications Interface Modbus slave. This protocol provides access <br> to the SOE buffer events, not Process Historian, over a Modbus data link where the default OPC server/SOE collector protocols prove too <br> slow. However, the SOE Over Modbus protocol requires an application in the Modbus master remote end to read, interact with and <br> interpret the buffer, and assign event data to variables. <br> For more information on events and their configuration, see Product Description PD-8013, Trusted SOE and Process Historian Package. <br> For more information on the SOE Over Modbus protocol and data block format, see Product Description PD-T8151B, Trusted <br> Communication Interface. |

SOE Over Modbus is implemented as a service running on a T8151 Communications Interface. When enabled, the service manages a window of Modbus addresses that implement the protocol which is automatically available on all ports of that module that are configured as Modbus slaves.
The settings in the Modbus Slave Configuration window define a block of register addresses that provide a window for accessing events.

## SOE Over Modbus Requirements:

- Configure and activate at least one Modbus slave.
- Configure variables for SOE logging. Set the Enable SOE Logging extended attribute for these variables or connect these variables to an SOE board and have a valid Modbus address.
- Configure the SOE Over Modbus service. Configure each communications module that supports the service. Define redundant configurations with the same information.
- Resolve Modbus address conflicts. The Modbus addresses specified by the address window must be unique and not defined by the application except for certain addresses in redundant configurations.
- Define Modbus addresses for redundant configurations. For redundant configurations across communication modules, the first four addresses in the SOE address window should be defined by the application but not used elsewhere.
- Configure the Modbus master. Ensure the Modbus master is configured with the appropriate addresses and has the SOE Over Modbus protocol implemented.


## SOE Over Modbus Redundant Configurations:

The SOE Over Modbus service runs independently on each T8151 Communications Interface module where it is configured. While the service on each module provides the same event data, the presentation of that data to a Modbus master is not normally synchronized. Use the service in a redundant, fail-over configuration with these steps:

- Configure each T8151 Communications Interface module that is part of the redundant configuration for SOE Over Modbus using the same values for Base Address and Block Count.
- Define four integer variables in the dictionary and assign a Modbus address that corresponds to the first four addresses in the SOE address window, starting at the address in Base Address. The SOE Over Modbus services use these variables on each communication module to synchronize the presentation of event data.
The SOE Over Modbus service has these properties:
- Base Address, the starting address of the window used by the protocol. The base address can be located at any valid Modbus holding register address. For example, Modbus addresses in the 40,000 to 49,984 range. End Address is updated as Base Address is modified.
- Block Count, the number of events that the Modbus master can read in a single Modbus read. Each event block requires four Modbus holding register addresses. End Address is updated as Block Count is modified. Block Count must indicate at least two event blocks.
- End Address, the last address of the Modbus address window for the SOE Over Modbus protocol. Calculate this value as: start address specified in Base Address
+7 (for protocol overhead)
+ (the number of blocks in Block Count * 4)
The Modbus address window indicated for Base Address and End Address should be reserved exclusively for use by the SOE Over Modbus service. Defining a variable with a conflicting Modbus address may lead to loss of events, corrupted event data, or protocol errors that prevent the continued operation of the SOE Over Modbus service.


## To configure a Modbus slave

1. In the T8151 Communications Interface, click Configure Modbus Slave.
2. In the Modbus Slave Configuration window, select the slave number and define the properties.
3. Select the Use this Slave check box and indicate the slave address.
4. Select the connection method for the slave:

- Initiate TCP/IP Connection. Indicate the TCP/IP address and port of the Modbus master.
- Wait TCP/IP Connection. Indicate the port of the incoming connection.
- Serial port. Indicate one of four possible serial ports.

5. Select a connection timeout:

- Never, where the Ethernet connection never times out
- Disconnect after, where the Ethernet connection times out after the indicated number of seconds. This property should remain at the default value to disconnect after 300 seconds.

6. Use the SOE over Modbus service:
a. Select the Enable check box.
b. Indicate these properties:

- Base Address, starting address of the window used by the protocol
- End Address, last address of the Modbus address window used by the protocol
- Block Count, number of events that the Modbus master can read in a single Modbus read

7. Click OK.

## See also

Use Modbus Communications on page 535
Configure a Modbus Master on page 541
Configure a Slave for a Modbus Master on page 547

## Configure a Modbus Master

The $\mathbf{7 8 1 5 1}$ Communications Interface can act as a Modbus ${ }^{\circledR}$ master with the addition of the Processor Interface Adapter T8122 or T8123. On a system without the appropriate Processor Interface Adapter connected, configure the Modbus master and download onto the Trusted ${ }^{\circledR}$ system. However, no communications activity occurs upon running the system. For details, see Product Description PD-T812x, Trusted Processor Interface Adapter.

Modbus master operates on the $\mathbf{7 8 1 5 1}$ Communications Interface. This interface can link to a wide range of external devices using either serial or Ethernet links, supporting the Modbus RTU protocol. Modicon ${ }^{\text {™ }}$ Open Modbus TCP is not supported.

The Modbus master configuration is performed in three stages: configure a master, configure the slaves serving that master, then configure the messages carrying data to and from each slave.

Configure these settings for a Modbus master:

- Configuration settings, communications and status reporting
- Broadcast modes, broadcasting messages to slaves
- Statistics data, recording the time taken to complete a master poll



## To configure a Modbus master

1. In the T8151 Communications Interface, click Configure Modbus Master.

The Modbus Master Configuration for Chassis 01, Slot 01 window appears. For example purposes, this procedure configures communications for the first slot of the first chassis.
2. Click Add Master.

The Master 1 Modbus master appears in the structure window on the left.
3. (optional) Click Change Name and then type a name for the master.
4. On the Configuration tab, define communications and status reporting for the master.
5. On the Broadcast tab, enable Broadcast Mode, and then define broadcast messages for slaves.
6. On the Statistics tab, enable Report Statistics, and then define report statistics data to the application.
7. Click Validate.

## See also

Define Communications and Status Reporting on page 543
Broadcast Messages to Slaves on page 544
Report Statistics Data on page 546
Validate Changes to Modbus Master Configurations on page 553

Define Communications and Status Reporting

The configuration settings for a Modbus ${ }^{\circledR}$ master include communications and status reporting. When configuring a Modbus master, define these properties:


## To define communications and status reporting for a Modbus master

1. Click the Configuration tab.
2. In the Communications Link group, select the type of port and define the properties:

- Ethernet
- Serial

3. In the Timeout area, specify a timeout delay (in milliseconds).
4. (optional) In the Applications Interface area for Slave Control, select Application Controls, and then indicate the Modbus address for the
communications control variable to enable the application to control communications for the Modbus master.
5. (optional) In the Applications Interface group for Status Reporting, select Report Status, and then indicate the Modbus address for the status reporting variable to report the status of the communications link to the application.
6. Click Save.

## See also

## Configure a Modbus Master on page 541

## Broadcast Messages to Slaves

The Modbus ${ }^{\circledR}$ master supports broadcasting messages to slaves. A Modbus broadcast is an unacknowledged write data message sent from the master using slave ID 0 . All slaves on the communications link that support broadcast messages process the message. Since the data transfer is one way only, broadcast messages are restricted to coil and holding register writes only. A broadcast sequence consists of any number of messages. Each message is enabled or disabled individually from the application. Each broadcast message counts towards the maximum number of slave messages allowed for the T8151 Communications Interface.

## Tip: When using broadcast messages, configure at least one non-broadcast message addressed to a specific slave.

For message control, enter the Modbus address of an application variable to control the message depending on the broadcast mode. Where a control variable is defined for either a broadcast or slave message, the message is enabled when the coil or holding register contains a zero value. All other values disable the message. The data type used for a control variable is either Boolean (coil) or integer (holding register).

Use different Modbus addresses for all control and status holding registers, and control variable coils.

| Variable Data Type | Value | Message |
| :--- | :--- | :--- |
| Boolean (coil) | False | Enabled |
|  | True | Disabled |
| Integer (holding register) | 0 | Enabled |
|  | Non-zero | Disabled |

Possible broadcast modes:

| Property | Description |
| :--- | :--- |
| One Shot | Sends a single broadcast sequence upon detecting an edge on the control variable within the application. After sending all enabled <br> broadcast messages, the mode is suspended until detection of a new edge on the control variable. |
| Message Count | Sends the broadcast sequence after sending every number of slave messages. Set the Every number of messages value. At run time, <br> the application cannot change the number of messages. All messages sent to the slaves are counted. Configured messages that are <br> disabled by the application do not count as sent messages. After sending the required number of slave messages, the broadcast <br> sequence starts. If required, configure the control variable to enable and disable the broadcast mode. |
| Fixed Time | Sends the broadcast sequence at every fixed time interval defined in the configuration, in milliseconds. Set the Every time interval. <br> At run time, the application cannot this interval. The time interval measurement starts from the point of sending the first broadcast <br> message and ignores the time taken to send the sequence and any wait times. Therefore, set the time interval to a value greater <br> than the time taken to send the broadcast sequence. If required, configure the control variable to enable and disable the broadcast <br> mode. |

When a broadcast is initiated, enabled messages in the list are sent one after another. Set a wait time interval between each message to allow for processing time of the slave devices.

Slave list control indicates the behavior for message scheduling. The possible values are:

- Restart, restarts with the first defined message for each slave after completing the broadcast sequence
- Suspend, suspends the sequence and continues at the same point after the broadcast

When creating a broadcast message, define these properties:

| Property | Description |
| :--- | :--- |
| Message Type | Type of message to broadcast to slaves. Possible types are Write Coils (multiple Boolean variables) or Write Registers <br> (multiple analog variables). |
| Variable Network Address | Modbus address range of the variables as mapped in the Trusted system. Specify only the first address since the last address <br> is calculated automatically. |
| Modbus Slave Address | Slave's address range for the variables. Enter the first and last addresses. Possible addresses for coils range from 1 to g999 <br> and possible mappings for registers range from 40001 to 49999. Write up to 512 coils or 123 registers in a single message. |
| Message Control | The address of the application variable to control the message. If application does not control the message, leave <br> unchecked. These are the valid values for a message control holding register. Invalid values result in the inactive state. <br> Use different Modbus addresses for all control and status holding registers, and control variable coils. |
|  | Salue |
| State | Description |
|  | Inactive |
|  | 0 |
| Standby | 1 |
|  | Sctive message to inactive |

## To broadcast messages to slaves

1. Click the Broadcast tab, and then select the Use Broadcast Mode check box.
2. In the Mode group, select the broadcast mode:

- One Shot, sends a single broadcast sequence upon detecting an edge on the control variable
- Message Count, sends the broadcast sequence after sending a fixed number of slave messages
- Fixed Time, sends the broadcast sequence at a fixed time interval defined in the configuration

3. (optional) For the Message Count and Fixed Time broadcast modes, in the Mode group, select the Use Control Variable check box and indicate the Modbus address for the message control variable to set a variable to control the broadcast message.
4. To set the Wait Time interval for processing of messages between broadcasts, indicate the interval duration in milliseconds.
5. In the Slave List Control area, indicate one of these for slave list control or message scheduling:

- Restart, restarts with the first defined message for each slave after completing the broadcast sequence
- Suspend, suspends the sequence and continue at the same point after the broadcast

6. Define the messages for broadcast:
a. Click New.

The New Broadcast Message dialog box appears.
b. Choose the message type: Write Coils or Write Registers.
c. In the Variable Network Address area, indicate the starting Modbus address of the variables as mapped in the Trusted system.
d. In the Modbus Slave Address area, indicate the first and last addresses of the address range of the variables for the slave.
e. In the Message Control area, select the Application Controls Message check box and enter the Modbus address of the application variable to use for controlling the message.

## 7. Click Save.

## See also

## Configure a Modbus Master on page 541

## Report Statistics Data

To report statistics for a master, define these properties:

| Property | Description |  |  |
| :---: | :---: | :---: | :---: |
| Statistics Reporting | Modbus addresses for the data transmission variable and control variable. For the data transmission variable, the value returned is an integer which represents the poll time in 1/100th of a second. For the control variable, the data type is either Boolean (coil) or integer (holding register). <br> Use different Modbus addresses for all control and status holding registers, and control variable coils. |  |  |
|  | Variable Data Type | Value | Message |
|  | Boolean (coil) | False | Enabled |
|  |  | True | Disabled |
|  | Integer (holding register) | 0 | Enabled |
|  |  | Non-zero | Disabled |
| Statistics Type | The type of statistics to record for the Modbus master: |  |  |
|  | Statistics Type | Description |  |
|  | Max Scan Rate | The maximum | n time |
|  | Scan Rate | The latest scan |  |
|  | Average | The average so | time |

Depending on the statistics type, the control variable performs a different function. For Max Scan Rate and Average, a falling and rising edge sequence resets the measurement.

| Statistics Type | Control Variable Value | Result |
| :--- | :--- | :--- |
| Max Scan Rate | False $->$ True (Boolean) <br> $0->$ non-zero (Integer) | Max value remains until next cycle completed. <br> Max value then updated with time of next Max Scan Rate complete cycle. |
|  | True $->$ False (Boolean) <br> non-zero $->0$ (Integer) | Triggers a reset |
|  | False $->$ True (Boolean) <br> $0->$ non-zero (Integer) | No effect (data updated every cycle) |
|  | True $->$ False (Boolean) <br> non-zero $->0$ (Integer) | No effect (data updated every cycle) |


| Statistics Type | Control Variable Value | Result |
| :--- | :--- | :--- |
| Average | False $->$ True (Boolean) <br> $0->$ non-zero (Integer) | Average value remains until next cycle completed. <br> Average value then updated with time of Average next complete cycle. <br>  |
|  | Triggers a reset |  |

The master poll scan is defined as the time taken to poll all of the slaves defined for the master. The first slave defined in the poll list is taken as the start and end point of the timing sequence.

| Sequence Number | Message |  |
| :--- | :--- | :--- |
| 1 | Slave 1 Msg 1 | Master |
| 2 | Slave 2 Msg 1 | Poll 1 |
| 3 | Slave 3 Msg 1 |  |
| 4 | Slave 1 Msg 2 | Master |
| 5 | Slave 2 Msg 2 | Poll 2 |
| 6 | Slave 3 Msg 2 |  |
| 7 | Slave 1 Msg 3 | Master 3 |
| 8 | Slave 2 Msg 1 |  |
| 9 | Slave 3 Msg 3 | Master |
| 10 | Slave 1 Msg 4 | Poll 4 |
| 11 | Slave 2 Msg 2 | Slave 3 Msg 1 |

## To report statistics data for a Modbus master

1. Click the Statistics tab, and then select the Report Statistics check box.
2. In the Statistics Reporting area, define the Modbus addresses for these variables:

- Data, application variable sending the data to the application
- Control, application variable performing a different function depending on the statistics type

3. In the Statistics Type area, select a statistics type:

- Max Scan Rate, the maximum scan time
- Scan Rate, the latest scan time
- Average, the average scan time

4. Click Save.

## See also

Configure a Modbus Master on page 541

## Configure a Slave for a <br> Modbus Master

- Statistics data, time to complete a slave polling



## To configure a slave for a Modbus master

1. In the tree structure, select the Modbus master, and then click Add Slave.
2. (optional) Click Change Name and enter a name for the slave.
3. Configure the properties for the slave:
a. To configure the configuration settings, click the Configuration tab, and then define the properties.
b. To configure the messages, click the Messages tab, and then define the properties.
c. To configure the statistical data, click the Statistics tab, and then define the properties.
4. Click Validate.

## See also

Configuration Settings on page 549
Messages on page 550
Report Statistics Data on page 552

## Validate Changes to Modbus Master Configurations on page 553

## Configuration Settings

The configuration settings for a slave for a Modbus ${ }^{\circledR}$ master include communications and status reporting. When configuring a slave for a Modbus master, define these properties:

| Property | Description |
| :--- | :--- |
| Communications | Station ID, Modbus communications identity for the slave. This value must be unique for all slaves connected to a single Modbus <br> master. For Ethernet-based slaves, possible communications are only single point-to-point. In this case, usually use an ID of 1. <br> Retries, number of attempts the Modbus master makes to communicate with a slave before declaring a communications failure. <br> Min Packet Gap, time the Modbus master waits between messages sent to the slave, in milliseconds. For slaves that must not be <br> accessed by the Modbus master at full communications speed. |
| Pinging | Method used by the Modbus master to test communications with an individual slave. The Modbus master pings a non- <br> communicating slave at regular intervals. Once the non-communicating slave responds to a ping, then polling of the slave restarts <br> on the next poll cycle. When a slave returns to the polling sequence, the first message that is sent to the slave is the first message <br> defined in the INI file for that slave. <br> Repeat Rate, time delay the Modbus master waits between each attempt to communicate with the slave. <br> Command, method the Modbus master uses for testing communications. Possible methods are: <br> $\bullet$ Use Modbus 08 Command, (Query Data), subcommand 0. Compatible slaves echo this command exactly, similar to an Ethernet <br> ping. |
| •Use Slave Register, reads data from a slave register to test communications to the slave, when the slave does not support Modbus |  |
| 08 Command. When using a register for testing communications, discards the data read from the register. |  |

## To define communications and status reporting for a slave of a

 Modbus master1. Click the Configuration tab.
2. In the Communications area, specify these properties:

- Station ID, Modbus communications identity for the slave
- Retries, number of attempts the Modbus mastermakes to communicate with a slave
- Min Packet Gap, time the Modbus master waits between messages sent to the slave

3. In the Pinging area, define the pinging method the Modbus master uses to test communications with slaves:
a. Indicate the Repeat Rate time delay the Modbus master waits between communication attempts with slaves.
b. Specify the Command method the Modbus master uses for testing communications:

- Use Modbus 08 Command
- Use Slave Register

4. (optional) In the Applications Interface area for Slave Control, select the Application Controls check box, and then indicate the Modbus address for the communications control variable to enable the application to control communications for the slave.
5. (optional) In the Applications Interface area for Status Reporting, select the Report Status check box, and then indicate the Modbus address for the status reporting variable to report the status of the communications link to the application.

## See also

Configure a Slave for a Modbus Master on page 547
Messages
The messages list displays all of the messages defined for a slave device in the order that these are executed by the Modbus ${ }^{\circledR}$ master. Perform these tasks to manage the messages list:

- Add a new message at the bottom of the messages list
- Insert a message above a selected message in the messages list
- Edit an existing message
- Delete a message from the messages list and edit messages
- Rearrange the order of messages in the list, by dragging and dropping

For slave messages, define these properties:

| Property | Description |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Message Type | Use the appropriate Modbus command depending on the address range and the number of addresses. The T8151 Communications Interface supports these commands: |  |  |  |  |  |
|  | Address Range |  | Size | Direction | Modbus Command |  |
|  | Low | High |  |  | Type | Code |
|  | 1 | 9999 | 1 | Read | Read Coil Status (Read Coils) | 0x01 |
|  | 1 | 9999 | 1 | Write | Force single Coil (Write Coils) | Ox05 |
|  | 1 | 9999 | 2-512 | Read | Read Coil Status (Read Coils) | 0x01 |
|  | 1 | 9999 | 2-512 | Write | Force Multiple Coils (Write Coils) | 0x15 |
|  | 10001 | 19999 | 1-512 | Read | Read Input Status (Read Digital Inputs) | 0x02 |
|  | 30001 | 39999 | 1-125 | Read | Read Analog Inputs (Read IP Registers) | 0x04 |
|  | 40001 | 49999 | 1-125 | Read | Read Holding Register (Read Registers) | Ox03 |
|  | 40001 | 49999 | 1 | Write | Preset Single Register (Write Registers) | Ox06 |
|  | 40001 | 49999 | 2-123 | Write | Preset Multiple Registers (Write Registers) | 0x16 |
| Variable Network Address | Modbus address range of the variables as mapped in the Trusted ${ }^{\ominus}$ system. Requires only the starting address; automatically calculates the end address. |  |  |  |  |  |
| Modbus Slave Address | Address range of variables for the slave. Enter the start and end addresses. See Message Type for available address ranges and maximum number of addresses in a single message. |  |  |  |  |  |


| Property | Description |  |  |
| :---: | :---: | :---: | :---: |
| Message Control | Modbus address of an application variable used to control the message. Where a control variable is defined for either a broadcast or slave message, the message is enabled when the coil or holding register contains a zero value. These are the valid values for a message control coil or holding register. All other values disable the message. The data type used for control variable can be either Boolean (coil) or integer (holding register). <br> Use different Modbus addresses for all control and status holding registers, and control variable coils. |  |  |
|  | Boolean (coil) | Value | Message |
|  |  | False | Enabled |
|  |  | True | Disabled |
|  | Integer (holding register) | 0 | Enabled |
|  |  | Non-zero | Disabled |

## To manage the messages list from a Modbus master to a slave

1. Add a new message to the messages list:
a. Click New.
b. In the New Message dialog box, define the properties for the message:

- Message Type, Modbus command message type
- Variable Network Address, Modbus address range of the variables as mapped in the Trusted ${ }^{\circledR}$ system
- Modbus Slave Address, start and end of slave address range for the variables
c. (optional) In the Message Control area, select the Application Controls Message check box, and then specify the Modbus address of the application variable used to control the message.

2. Insert a new message in the messages list:
a. In the messages list, select the message above which to insert the message.
b. Click Insert.
c. In the Insert Message dialog box, define the properties for the message. See step 1 for message properties to define.

The message appears in the list above the selected message.
3. Edit an existing message:
a. In the messages list, select the message to edit and click Edit.
b. In the Edit Message dialog box, make the necessary changes.
4. To delete a message from the messages list, select the message to remove and click Delete.
5. To rearrange the order of a message in the messages list, select the message, and then drag and drop the message to its new position in the list.

## See also

## Report Statistics Data

Enable reporting statistics to record the statistics data for a slave for a Modbus ${ }^{\circledR}$ master. The statistics data records the time to complete a slave polling.

To report statistics for a Modbus slave, define these properties:

| Property | Description |
| :--- | :--- |
| Statistics Reporting | Modbus addresses for the data transmission variable and control variable. The data transmission reports to the application using a <br> Modbus data variable. For the data transmission variable, the value returned is an integer which represents the poll time in $1 / 100$ th of <br> a second. For the control variable, the possible data type values are Boolean (coil) or integer (holding register). <br> Use different Modbus addresses for all control and status holding registers, and control variable coils. |


|  | Variable Data Type |  |  |
| :---: | :---: | :---: | :---: |
|  |  | Value | Message |
|  | Boolean (coil) | False | Enabled |
|  |  | True | Disabled |
|  | Integer (holding register) | 0 | Enabled |
|  |  | Non-zero | Disabled |
| Statistics Type | The type of statistics to record for the Modbus slave: |  |  |
|  | Statistics Type | Description |  |
|  | Max Scan Rate | The maximum scan time |  |
|  | Scan Rate | The latest scan time |  |
|  | Average | The average scan time |  |

Depending on the statistics type, the control variable performs a different function. For Max Scan Rate and Average, a falling and rising edge sequence resets the measurement.

| Statistics Type | Control Variable Value | Result |
| :--- | :--- | :--- |
| Max Scan Rate | False $->$ True (Boolean) <br> $0 \rightarrow>$ non-zero (Integer) | Max value remains until next cycle completed. <br> Max value then updated with time of next complete cycle. |
|  | True $->$ False (Boolean) <br> non-zero $->0$ (Integer) | Triggers a reset |
|  | False $->$ True (Boolean) <br> $0 \rightarrow$ non-zero (Integer) | No effect (data updated every cycle) |
|  | True $->$ False (Boolean) <br> non-zero $->0$ (Integer) | No effect (data updated every cycle) |
| Average | False $->$ True (Boolean) <br> $0->$ non-zero (Integer) | Average value remains until next cycle completed. <br> Average value then updated with time of next complete cycle. |
|  | True $->$ False (Boolean) <br> non-zero $\rightarrow 0$ (Integer) | Triggers a reset |

The slave poll scan is the time taken to poll a slave for all messages in the poll list. The first message defined in the poll list is taken as the start and end point of the timing sequence.

| Sequence Number | Message |  |  | Slave 1 |
| :---: | :---: | :---: | :---: | :---: |
| 1 | Slave 1 Msg 1 | Slave 2 |  | Tstart |
| 2 | Slave 2 Msg 1 | Tstart | Slave 3 |  |
| 3 | Slave 3 Msg 1 |  | Tstart |  |
| 4 | Slave 1 Msg 2 |  |  |  |
| 5 | Slave 2 Msg 2 |  |  |  |
| 6 | Slave 3 Msg 2 |  |  |  |
| 7 | Slave 1 Msg 3 | Slave 2 |  |  |
| 8 | Slave 2 Msg 1 | Tstop |  |  |
| 9 | Slave 3 Msg 3 |  | Slave 3 |  |
| 10 | Slave 1 Msg 4 |  |  |  |
| 11 | Slave 2 Msg 2 |  |  |  |


| Sequence Number | Message | Slave 1 |  |
| :--- | :--- | :--- | :--- |
| 12 | Slave 3 Msg 1 |  |  |
|  |  | Tstop |  |
| 13 | Slave 1 Msg 1 |  | Slave 1 |
| 14 | Slave 2 Msg 1 | Tstop |  |
| 15 | Slave 3 Msg 2 |  |  |

## To report statistics data

1. Click the Statistics tab, and then select the Report Statistics check box.
2. In the Statistics Reporting area, define the Modbus addresses for these variables:

- Data, application variable sending the data to the application
- Control, application variable performing a different function depending on the statistics type

3. In the Statistics Type area, select a statistics type:

- Max Scan Rate, the maximum scan time
- Scan Rate, the latest scan time
- Average, the average scan time

4. Click Save.

## See also

Configure a Slave for a Modbus Master on page 547

Validate Changes to Modbus Master Configurations

Validate changes to Modbus ${ }^{\circledR}$ master configurations after saving changes. Validation runs automatically after saving changes and exiting a Modbus master configuration.


## To validate changes to Modbus master configurations

1. Make necessary configuration changes to any of a Modbus master properties:

- Communications and status reporting (Configurations tab)
- Broadcast messages to slaves (Broadcast tab)
- Statistics data (Statistics tab)


## 2. Click Validate.

The Validating Modbus Master Configuration window appears with the validation results.

## See also

## Configure a Modbus Master on page 541

## Generating Code

Before downloading code onto the target systems, build the code for the whole project. This operation verifies and builds the code for all libraries, devices, programs, functions, and function blocks and builds information used to recognize the systems on networks. Once a project is built, subsequent build operations only recompile the parts of the project needing regeneration.

Choose to build the project or the device. Also verify the syntax of POUs or the Global library. Building or verifying a selected project element, only verifies or builds the code for the selected element.

Choose to clean projects and devices. Cleaning projects and devices deletes the intermediate and output files generated during the last build operation. After cleaning, regular updates cannot be performed. To retain the capacity to perform regular updates, perform subsequent builds for a project or device rather than cleaning then building it. However, intelligent updates may be performed after cleaning a project or device.

## See also

## Build a Trusted Project on page 554

Verifying POU Syntax on page 555

## Cleaning a Trusted Project on page 556

## Build a Trusted Project

Choose to generate code and the necessary files for a project. Also choose to build only a selected element within a project such as the device, function, or function block. Once a project has been built, subsequent builds only recompile the parts of the project needing recompiling.

When building projects or devices, view the progress of the build operation in the Output window. When the build operation is complete, view generated errors in the Error List.

| Task | Procedure |
| :--- | :--- |
| Build a project | $\bullet$ •In the Application View, right-click the project element, and then click Build All (or press Ctrl+Shift+B). |
| Build the device within a project | 1. In the Application View, click the device. <br> 2. From the Build menu, click Build <Device>. |


| Task | Procedure |
| :--- | :--- |
| View build progress and generated <br> errors | 1. From the Tools menu, click Options. <br> 2. In the Options dialog box, expand Projects, click General, select the following options, and then click OK. <br> - Always show Error List if build finishes with errors <br> - Show Output window when build starts |
| 3. Build the required project. |  |

The AADvance-Trusted SIS Workstation software rebuilds the device and increases the compilation version number in the following conditions:

- The device has been modified since the last compilation
- A dependent library is not compiled, was compiled after the device, or was modified since the last compilation
- A POU has been modified since the last compilation
- The device contains variables in binding consumer links or variables in CIP consumer links.

The compiler generates different code for simulation than for targets. Therefore, specify the code for simulation in the device properties before building.

## See also

Verifying POU Syntax on page 555
Cleaning a Trusted Project on page 556
Run an Application Online on page 556
Download code to Trusted controllers on page 557
Choose to verify the syntax for programs, functions, and function blocks. Also verify the syntax of functions and function blocks belonging to a library. When verifying the syntax, the AADvance-Trusted SIS Workstation software only verifies the programming syntax without producing code. View the verification progress in the Output window.

## To verify POU syntax

1. In the Application View, right-click the program, function, or function block, and then click Verify <POU>.
2. (optional) To view the verification progress and generated errors, from the Tools menu, click Options.

- In the Options dialog box, expand Projects, click General, select these options, and then click OK.
- Always show Error List if build finishes with errors
- Show Output window when build starts


## See also

Programs on page 525
Functions on page 527
Function Blocks on page 529

## Cleaning a Trusted Project

Delete intermediate and output files belonging to projects or devices. Cleaning projects, or deleting these files, enables the generation of new files during the next build operation. When cleaning the device, the intermediate and output files are deleted for the device only.

After cleaning a project or device, regular updates cannot be performed. To retain the capacity to perform regular updates, perform a subsequent build rather than cleaning and then building. intelligent updates may still be performed after cleaning a project or device.

| Task | Procedure |
| :--- | :--- |
| Clean a project | In the Application View, right-click the project to clean, and then click Clean All. |
| Clean a device | In the Application View, right-click the device to clean, and then click Clean. |

## See also

Build a Trusted Project on page 554
Verifying POU Syntax on page 555
Intelligent Online Updates on page 569

## Running an Application Online

Running online means an application is connected to the physical Trusted ${ }^{\circledR}$ controller. AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software ensures the execution of the application code. While running online, force the values of variables and monitor the values of variables.

Before running an application on a Trusted controller, build the project code and download the application code onto the target.

For debugging purposes, simulate the running of an application. The compiler generates different code for simulation than when online.

Status information for the Trusted controller appears in the Application View as a lightning bolt icon:

> When the icon is yellow, the Trusted controller is connected or in simulation mode.

From the Application View, view detailed status information for the Trusted controller, including the identity of the AADvance-Trusted SIS Workstation software locking the controller, by hovering over the controller status icons.
When connected, the Application View displays states for the device:

|  | The device is running on the target. The device is in the RUN, STOP, or ERROR state. |
| :--- | :--- |
| $!$ | The device is not running on the target or no code is available on the target. The device is in the <br> DISCONNECTED or NO APPLICATION state. |

The state of the device appears next to the device icon in the Application View:

| Device State | Description |
| :--- | :--- |
| RUN | The device is running in real-time mode. The state of a device when online. |
| DISCONNECTED | Unable to establish communication with the target device. <br> Verify communication settings such as IP address. |
| NO APPLICATION | The device is not running on the target. Possible reasons: <br> - The device is stopped. Possible operation is to start the device. <br> - The device is in error. Correct the error and download code to the target <br> device. <br> - No code is available on the target. Download code to the target device. |

## See also

Build a Trusted Project on page 554
Download code to Trusted controllers on page 557
Connect with Trusted controllers on page 558
Disconnect from Trusted controllers on page 561
Simulate an Application on page 561

## Download code to Trusted controllers

Performing a download operation verifies that the latest build reflects the current project. Perform download operations for projects with code to send to physical Trusted controllers. When simulating, do not perform download operations.

## Prerequisites

- Connect the computer where the AADvance-Trusted SIS Workstation software is installed to the hardware controller through a Serial or Ethernet network.


## To download project code

1. In the Properties window for the device, configure these properties:

- Enable ISA68M
- Host Address
- Host Socket Port

Tip: The compiler generates different code for simulation than for targets.
2. Build the project code.
3. Set the maintenance enable key switch on the Trusted controller to the Maintain position.
4. From the Application View, right-click the project element, and then select Download.

## See also

Build a Trusted Project on page 554
Connect with Trusted controllers on page 558
Disconnect from Trusted controllers on page 561
Start or stop a Trusted controller on page 560
Connect with the physical exact et Trusted ${ }^{\circledR}$ controller when developing an application. While connected, the Trusted controller executes the application in real-time mode. When connecting an application, the AADvance-Trusted SIS Workstation software verifies the coherency between the current application definitions and the compiled code on the Trusted controllers.

A Trusted controller supports a maximum of one connection from a AADvance-Trusted SIS Workstation software.

## Prerequisites

- Build the application code and download the code to the target.

Important: Modifications other than these to a project may affect the accuracy or availability of monitored values when connecting without rebuilding and downloading:

- Minor graphical changes not affecting the execution order
- Descriptive text such as comments, element descriptions, and diary entries
- Migration from a legacy project
- Addition to the global library of functions and function blocks not used in the project
- Addition or removal of spy lists or ISaVIEW screens


## To connect an application

1. In the Properties window for the device, configure the Host Address and Host Socket Port properties.
2. From the Project menu, select Connect Mode > Online. The Online menu displays online command options.
3. Build the project code.
4. Download the project code to the physical controller.
5. In the Application View, select the Project element, and then select Online > Connect. The AADvance-Trusted SIS Workstation software saves the application state.

Tips: - While connected, perform tasks that include:

- monitor values of variables, diagnostic information, and ISaVIEW screens
- lock and unlock the I/O channels of an I/O device
- modify values of variables and cycle time
- Upon disconnecting, the AADvance-Trusted SIS Workstation software returns the device to the state previous to connecting.


## See also

## Disconnect from Trusted controllers on page 561

Forcing the Values of Variables on page 562
Download code to Trusted controllers on page 557

## Online Updates on page 567

Trusted Devices on page 523

## Access diagnostic information

## To access the Diagnostic tool

From the Application View, right-click the device and then click Diagnostic.
The Diagnostic tool displays a high-level overview of the Trusted ${ }^{\circledR}$ application. Use Diagnostic tool to access diagnostic information like the application state, last application error, various values for the cycle time, and the target and AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software versions. Communications between the AADvance-Trusted SIS Workstation software and target must be established to display the diagnostic information. When connected to the application, refresh the Diagnostic tool to update the displayed information.


Diagnostic information that displays in the Diagnostic tool:

| Information | Description |
| :--- | :--- |
| Target type | The type of target. For Trusted applications, the possible value is ISA68M. When simulating, the <br> possible value is ISA86M. |
| Application name | The name of the Trusted application. |
| Application state | The state of the Trusted application. Possible values are Running, Stopped, or Error. |
| Application execution state | The execution state of the Trusted application. Possible values are RealTime or CycleToCycle. |
| Last application error | Displays the last application error message or none if no application error occurred. <br> Tip: If the Output window displays the application error before you open the <br> Diagnostic tool, the Last application error is none. |
| Application identification |  |
|  | Target |


| Information | Description |  |
| :--- | :--- | :--- |
|  | Workbench | Displays the version, the generation date and time of the code, and the CRC symbols value. |
| Cycle timing information from target |  |  |
|  | Programmed cycle time | The cycle time defined for the device, in milliseconds. Define the programmed cycle time using the <br> Cycle Timing(ms) property in the Properties window. |
|  | Last measured cycle time | The real cycle time of the last application cycle, in milliseconds. |
|  | Maximum measured cycle time | The longest period of time used for a cycle since connecting with the Trusted application. |
|  | The number of cycles having exceeded the programmed cycle time. |  |

Refresh the information that displays in the Diagnostic tool . When communications between the AADvance-Trusted SIS Workstation software and target are established, the Diagnostic tool displays the last readings from the target. When connected to the application, refresh the Diagnostic tool to update the displayed information.

Icons that display status of the data in the Diagnostic tool :

|  | Refresh is in progress. |
| :--- | :--- |
| $\underbrace{\text { a }}$ | Refresh was successful. |
|  | There was an error refreshing the information. <br> The error is usually due to a connect attempt failure. |

## To refresh the information displayed in the Diagnostic tool

- From the Diagnostic tool, click है


## See also

Trusted Devices on page 523
Connect with Trusted controllers on page 558

## Start or stop a Trusted controller

Stop or start the application running the Trusted controller from the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software.

- Stopping the execution of the application causes the Trusted controller to go into the configured safe state.
- Starting the execution of the application causes the output to energize depending on the programmed logic conditions.
- Starting or stopping the application is only available when online.


## Start or stop a Trusted controller

1. Connect to the application.
2. From the Online menu, click Start application or Stop application.

The application running on the Trusted controller stops or starts.

## See also

Connect with Trusted controllers on page 558
Disconnect from Trusted controllers on page 561

## Disconnect from Trusted controllers

## Simulate an Application

Before downloading code to Trusted controllers, disconnect them.

## To disconnect Trusted controllers

- In the Application View, select the Project element, then from the Online menu, click Disconnect.


## See also

Download code to Trusted controllers on page 557
Connect with Trusted controllers on page 558
Start or stop a Trusted controller on page 560
Simulating running an application signifies that virtual machines execute the code of the device and the Windows platform performs aspects like program organization unit (POU) execution. Virtual machines ignore inputs and outputs.

The compiler generates different code for simulation than for online.

## Prerequisite

Build the project code before simulating an application. Simulation executes code in real-time mode. The programmed cycle timing triggers target cycles.

## To simulate an application

1. Select the device, then set the Enable Simulation property to True from the list in the Properties window.
2. For applications using functions and function blocks from the global library, set the global library for simulation.
a. In the Application View, expand Libraries, then Global.
b. Select Functions, then set the Targets property to Simulate from the list in the Properties window.
c. Select Function Blocks, then set the Targets property to Simulate from the list in the Properties window.
3. From the Project menu, point to Connect Mode, and then click Simulation to set the connection mode for the project.
The Simulation menu category appears with the simulation command options.
4. In the Application View, select the project.
5. Build the project code, then select Simulation $>$ Start Simulation.

## See also

Forcing the Values of Variables on page 562
Build a Trusted Project on page 554

# Forcing the Values of Variables 

While connected or simulating, force or override the values of variables. These variables can be user-defined or directly represented. The behavior of a variable is defined by its logical value, physical value, lock state, and direction. When forcing the values of variables, the value to overwrite depends on the direction of the variable. Force the values of variables from these views:

- Local Variable dictionary
- Global Variable dictionary
- IEC 61131-3 language editors
- I/O wiring
- ISaVIEW
- Spy list

For locked variables, the values displayed in the Logical Value and Physical Value columns differ depending on their direction. Variable direction is determined from the direct representation definition for the I/O wiring.

Input Variable (Read) Behavior


Example: To force the temperature reading from a sensor.

Output Variable (Write) Behavior


Example: To force the closing of an actuator valve.
For variables with the write access value of false, lock access before forcing the values. For variables with the write access value of true:

- From the desired variable view, double-click the variable's corresponding cell in the Logical Value column, then in the dialog box, enter the required value and click Write.
- From the Ladder Diagram (LD) and Function Block Diagram (FBD) editors, double-click the variable, then in the dialog box, enter the required value and click Write.
- From the Structured Text (ST) editor, click the red underline and then click . In the Write Logical Value dialog box, enter the required value and click Write.

While connected, force the values of directly-represented variables.

## To force the value of a variable

1. From the Dictionary instance, locate the required variable.
2. Write the required value in the respective value column:

- For an input variable, write the value in the Logical Value column.
- For an output variable, write the value in the Physical Value column if the variable is locked and write the value in the Logical Value column if the variable is unlocked.


## See also

Monitoring the Values of Variables on page 563
Dictionary on page 611
Connect with Trusted controllers on page 558
1/O Wiring on page 638
ISaVIEW Screens on page 709

# Monitoring the Values of Variables 

While the AADvance-Trusted SIS Workstation software is in online or simulation mode, monitor the values of variables by using these views:

- Local Variable dictionary
- Global Variable dictionary
- IEC 61131-3 language editors
- I/O wiring
- ISaVIEW
- Spy list

Generating monitoring information increases the size of the TIC code created.
For dictionary instances and spy lists, the logical values, physical values, and lock status of variables are displayed in their respective columns. For graphical program organization units (POUs), values are displayed differently depending on their type:

- Values of variables having the 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.
- Values of variables having the DINT, REAL, MESSAGE, and TIME types are displayed as a numeric or textual value.

Depending on the view, the logical and physical values display these messages when variables are unavailable:

| View | Message |
| :---: | :---: |
| Dictionary | - Offline, indication that the local variable is not used in a program or it is a function block instance input. <br> - WAIT, indication that the variable is either: <br> - in online mode and attempting to connect to the target, or <br> - in simulation mode and attempting to connect to the simulator. |
| Spy list, <br> ISaVIEW | - No message, indication that the local variable is not used in a program or it is a function block instance input. <br> - WAIT, indication that the variable is either: <br> - in online mode and attempting to connect to the target, or <br> - in simulation mode and attempting to connect to the simulator. |
| IEC 61131-3 language editors | WAIT, indication that the variable is either: <br> - in online mode and attempting to connect to the target, or <br> - in simulation mode and attempting to connect to the simulator. |

When variables are unavailable, no message displays and the value is not editable in the Write Logical Value dialog box.

Tip: Increase the refresh rate of logical values, physical values, and locked status in the active view by closing other views that display this data, such as ISaVIEW screens and dictionaries.

## To generate monitoring information

1. In the Application View, select the device for which to generate monitoring information.
2. In the Properties window, set the Generate debug information property to True.

## See also

Forcing the Values of Variables on page 562
Connect with Trusted controllers on page 558
Simulate an Application on page 561
Run an Application Online on page 556
Trusted Devices on page 523

## Name conventions and limitations

| Conventions | Limitations |
| :---: | :---: |
| Projects |  |
| Project names | Project names have a maximum of 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. The last character for a project name must be a letter or digit; project names cannot end with an underscore character. Do not use reserved words, defined words, or data types for names. |
| Device quantity | Projects contain one device. |
| Device names | Device names have a maximum of eight (8) characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Names cannot end with an underscore. Do not use reserved words, defined words, or data types for names. |
| Program Organization Units (POUs) (Programs, Functions, and Function Blocks) |  |
| POU names | POU names have a maximum of eight (8) characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. Use unique names within a project. |
| POUs per project | Projects can contain up to 254 POUs. <br> The maximum number of variables is directly dependent on the Trusted ${ }^{\oplus}$ runtime 16 -bit application. |
| Hierarchical levels | The maximum hierarchical levels for POUs is 20 |
| Operator parameters | The addition (+), multiplication (*), AND (\&), CAT, and OR (>=1) operators have a maximum of 32 input parameters. |
| Function parameters | Functions have a maximum of 32 parameters ( 31 inputs and one output). |
| Function parameter names | Function parameter names have a maximum of 32 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. Use unique names for the parameters of a function. |
| Function block parameters | Function blocks have a maximum of 32 parameters (inputs and outputs). |
| Function block parameter names | Function block parameter names have a maximum of 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. Use unique names for the parameters of a function block. |
| Variables |  |
| Variable quantity | The maximum number of variables depends directly on the Trusted runtime 16-bit application. |
| Variable names | Variable names have a maximum of 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Do not use reserved words, defined words, or data types for names. Use unique names. <br> The names of variables having a defined Modbus address, initial value, or retained property are calculated by combining the variable and POU names for a maximum of 32 characters also beginning with a letter or single underscore followed by letters, digits, and single underscores. |
| Variable comments | Variable comments have a maximum of 60 single-byte characters. |
| Boolean variables | Boolean variables have the Boolean value TRUE (1) or FALSE (0) and can have an internal, constant, input, or output attribute. <br> The Boolean variable values Message True and Message False are limited to eight Windows-1252 characters excluding, \% " and without leading and trailing spaces. |


| Conventions | Limitations |
| :---: | :---: |
| Projects |  |
| DINT variables | DINT variable integer values range from -2147483647 to +2147483647 and can have an internal, constant, input, or output attribute. Integer literals must begin with a prefix identifying the base: <br> - DECIMAL values have no prefix <br> - HEXADECIMAL values have the prefix "16\#" <br> - OCTAL values have the prefix "8\#" <br> - BINARY values have the prefix "2\#" |
| Real variables | Real variables have six significant digits. For larger values, the maximum possible value is $\pm 3.402823466 \mathrm{E}+38$ while for smaller values, the minimum possible value is $\pm 1.175494351 \mathrm{E}-38$. Real literal values can be written with either decimal or scientific representation. The exponent part of a real scientific expression must be a signed integer value ranging from -37 to +37 . The scientific representation uses the 'E' or 'F' letter to separate the mantissa part and the exponent. Real variables can have an internal, constant, input, or output attribute. |
| Time variables | Time variables can have positive values ranging from 0 to 23h59m59s999ms. The time literal value must begin with the "T\#" or "TIME\#" prefix. Time variables can have an internal or constant attribute. |
| MESSAGE variables | MESSAGE variable string capacity has a maximum of 252 characters excluding the terminating null character ( 0 ) , a byte for the current length of the string, and a byte for the maximum length of the string. Characters must be preceded and followed by single quote (') characters. When placing single quote (') characters within a message literal, these characters must be preceded by the dollar (\$) character. |
| Modbus Address | Modbus addresses depend on the data type and extended attributes of a variable. These are the available address ranges for the possible data types: <br> - Coil (BOOL) from 1 to 10000 <br> - Discrete input (BOOL) from 10001 to 20000 <br> - Input register (DINT or REAL) from 30001 to 39999 <br> - Holding register (DINT or REAL) from 40001 to 49999 |
| Defined Words |  |
| Defined Word names | Defined word names have a maximum of 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Do not use reserved words, defined words, or data types for names. Use unique names within a project. |
| Defined word equivalents | Defined word equivalents have a maximum of 255 characters. |
| Defined word quantity | Trusted programs have a maximum of 255 defined words. Trusted projects have a maximum of 3554 defined words. |
| Defined word comments | Defined word comments have a maximum of 60 characters. |
| 1/0 Wiring |  |
| 1/0 modules per project | The maximum number of $1 / 0$ module instances depends directly on the Trusted runtime 16-bit application. |
| Hardware racks | A hardware rack has a maximum of $2551 / 0$ boards |
| I/O boards per project | A project has a maximum of 255 single $1 / 0$ boards. |
| 1/0 channels | Each I/0 board has a maximum of 128 I/0 channels. These channels can be inputs or outputs. |
| 1/0 module order | The $1 / 0$ module order ranges from 0 to 254 |
| Conversion table names | Conversion table names have a maximum of 16 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Do not use reserved words, defined words, or data types for names. |

## Function Block Diagram (FBD) Programs

| Conventions | Limitations |
| :--- | :--- |
| Projects | Label elements have a maximum of 32 characters beginning with a <br> letter or single underscore followed by letters, digits, and single <br> underscores. |
| Label elements | Label elements have a maximum of 455 characters beginning with a <br> letter followed by letters, digits, and single underscores. |
| Ladder Diagram (LD) Programs | Rung comments have a maximum of 251 characters. |
| Label elements | ST statements (one line of code) are recommended to have less than <br> Rung comments <br> Structured Text (ST) Programs <br> ST statements <br> Instruction List (IL) Programs <br> Instruction lists <br> Flow Chart (FC) Programs <br> Flow charts <br> Sequential Function Chart (SFC) Programs <br> Sequential function charts |

# File System Paths 

## See also

## Creating a Trusted Project on page 515

Online updates enable making online changes to a running application. These online updates include implementing a modification, downloading that modification to the system, then having the system seamlessly switch to the modified application. A seamless switch means no interruptions to the application process. Online changes are necessary for processes where any interruption may jeopardize production or safety.
Performing updates while the application is running can affect the Safety
Function of the system. Updates should only be performed when absolutely
required since the safety integrity of the system may be reduced during the
modification.
Before performing updates while the device is connected, set up alternative
safety measures lasting the duration of the update.
Performing an update on a Safety Implemented System is the responsibility of
the user. The running application may stop if modifications are performed
incorrectly.

Online updates are available in two types:

- Regular Online Updates
- Intelligent Online Updates

| Important: | Online updates may interrupt Modbus ${ }^{\oplus}$ communications. The <br>  <br>  <br>  <br>  <br>  <br>  <br> uodbus task stops and restarts (less than one second) during an |
| :--- | :--- |

Regular online updates are always enabled and occur when making minor modifications to an application, for example, without changing the data memory space (CRC). Intelligent online updates must be enabled and occur when making major changes to an application. Intelligent updates allow modifications to the data memory space.

An update is split into three phases:

1. Downloads the update to the device
2. Confirms the download is error free and verifies if the device will accept the update
3. Realizes the update

Tip: $\quad$ Perform an update after building a project. Updates are unavailable after cleaning a project. To perform intelligent updates, enable the option from the Intelligent Update Manager.

## To perform an update

1. For intelligent updates, ensure the device contains sufficient memory space.
2. Set the maintenance enable key switch on the Trusted controller to the Maintain position.
3. From the Application View, right-click the project or device for which to perform the update, and then click Update.

## See also

Regular Online Updates on page 568
Intelligent Online Updates on page 569

## Regular Online Updates

Regular online updates are always enabled and occur when making minor modifications to an application, for example, without changing the data memory space (CRC). Regular online updates support these minor modifications:

- Modifying the initial value of a variable if an initial value has previously been specified
- Modifying the retain setting of a variable. Previously retained values are no longer available.
- Modifying the Modbus address of a variable
- Modifying the comment of a variable
- Modifying the contents of a conversion table

Regular updates do not support adding or deleting global variables or function blocks. To ensure not changing the data memory space, declare spare variables, function blocks, and programs in advance.

## See also

## Intelligent Online Updates on page 569

Intelligent Online Updates

Intelligent online updates are performed by the Intelligent Update Manager. Intelligent updates are only performed when the modifications exceed those supported by a regular update. Before performing intelligent online updates, enable the feature to support making major changes to an application affecting the data memory space (CRC).

For intelligent online updates to succeed, the application size must be less than 800 KB to ensure that the size does not exceed 1 MB after an update.

General rules and limitations for intelligent updates:

- Intelligent updates were enabled before the previous application download, generating a baseline target snapshot.
- The target must be ISA68M and set to False the Enable Simulation compiler option from the device properties.
- Do not delete files contained in the project folder. The Intelligent Update Manager stores compilation information used for determining the need for intelligent updates in databases included in this folder.
Tip: If the Intelligent Update Manager determines that an intelligent update is not required, performs a regular update. The Intelligent Update Manager also performs mixed updates (regular and intelligent consecutively). Only regular updates support changes to Real in-line constants.
- The number and type of I/O devices and I/O boards must remain the same.
- The OEM parameter data for an I/O device must remain the same.

Important: Changes to OEM parameters on I/O module complex equipment (or peer-to-peer board) take effect after stopping and restarting the application (or re-powering the system). The effect is the same for regular or intelligent online update.

- Variables with values to retain across an update must have the same name, scope, and base type (for example, Boolean, analog, timer, or message). The Intelligent Update Manager omits unused local variables. Refer to the table below listing possible modifications for variables.
- Function blocks (standard or user-defined) must have the same calling parameters or inputs. Certain other restrictions for function blocks are noted above. Refer to the table below listing possible modifications for function blocks.
- When forcing the values of variables or power flow monitoring, use the same application version as that currently running on the system.

The Intelligent Update Manager supports these major modifications:

| Element | Modification |
| :---: | :---: |
| Variables | - Adding or deleting a global, local, or I/0 variable. The I/O variable must be connected to an I/O channel. <br> - Modifying the name of a variable. The current value of the variable is lost when renaming the variable during an update. <br> - Modifying the data type of a variable between DINT and Real. The current value of the variable is lost during the modification. <br> - Modifying the initial value, direction (internal, input, or output), Modbus address, or comment of a variable. <br> - Modifying the retain setting of a variable. Previously retained values are no longer available. <br> - Modifying the extended attributes of a global variable. |
| Functions and Function Blocks | - Adding or deleting a function block instance. <br> - Adding, deleting, or modifying a function or function block. <br> - Modifying the parameters of a function or function block. The internal value of the function block is lost during the update. <br> - Modifying the local variables of a function block. <br> - Modifying the name of a function or function block. For function blocks, the data values are lost during the update. |
| Programs | - Adding, deleting, or modifying a program. <br> - Modifying the name of a program. The data values are lost during the update. <br> - Modifying the program hierarchy. |
| Conversion Table | - Adding, deleting, or modifying a conversion table. <br> - Assigning, modifying, or removing the conversion table for a variable. |

During an update, existing variable values are maintained (unless specified otherwise above). New variables use their initial value during an update. Function data is not maintained across updates.

Each time function blocks are affected by logic changes, the instance data of a local variable increases. The Intelligent Update Manager is unable to parse instance data over 9999. Therefore, even though no modification occurs, an online update is performed because the manager is unable to match the instance data. Unmatched function blocks lose their internal states.

Tip: $\quad$ To work around the limit of 9999 instances per program, cut the required program then save and close the language container. Open the language editor and then paste the cut program. The instance data numbering is reset.

## To enable intelligent updates

1. From the Build menu, click Intelligent Update Manager.
2. From the Intelligent Update Manager dialog box, click Options...
3. In the Options dialog box, select Enable Intelligent On-line Updates.
4. (optional) Select the following options (selecting all options is recommended):

- Inform user when On-line update is not possible
- Inform user when Intelligent Update will be used
- Always display log when building download image
- Always display log when updating the target snapshot or history file
- Show temporary variables in log when building download image
- Display Match Function Blocks information dialog

5. In the Number of non-protected historical snapshots to keep, set the required value.
6. Click OK.

## See also

Intelligent Update Manager on page 571
Intelligent Update Options on page 572
Manual Function Block Matching on page 574

## Intelligent Update Manager

To access the Intelligent Update Manager

- From the Build menu, click Intelligent Update Manager.

When intelligent updates are enabled, the Intelligent Update Manager dialog box displays a list of target snapshots after the initial compilation. Snapshots are copies of application versions taken after a download. Set the number of historical snapshots saved by the Intelligent Update Manager using the Intelligent Update Options dialog box. When the maximum number of historical snapshots is exceeded, the oldest snapshot is deleted. Protect snapshots. Protected snapshots are not included in the maximum number of historical snapshots. Also delete snapshots.


Unless specified otherwise, the newest snapshot is set as the current target snapshot. The current target snapshot has a gray background and is the topmost item in the list. A snapshot is necessary for intelligent updates to function properly.

Tip: Obtain application version information from the TIC Version Checker or from the Diagnostic Tool.

The Intelligent Update Manager compares the most recently compiled application with the target snapshot version currently running on the target
and attempts to match data points and values between both. The Intelligent Update Manager generates the list of variable values to transfer and the device loads these values when performing an update. The Intelligent Update Manager runs each time an application is updated (compiled and downloaded) requiring intelligent updates.

| Task | Procedure |
| :--- | :--- |
| Generate a new target snapshot | Generate a target snapshot based on the latest valid compiled application. Before performing an update to a running system <br> where intelligent online updates were not previously enabled, generate a target snapshot for use as a baseline of the current <br> version. <br> 1. Compile the required project. <br> 2. From the Intelligent Update Manager dialog box, click New Target Snapshot. |
| Protect a snapshot | Protect a selected historical snapshot from deletion. Protected snapshots do not count towards the number of unprotected <br> snapshots. <br> - From the Intelligent Update Manager dialog box, select the required snapshot in the list, and then click Protect. |
| Unprotect a snapshot | Remove protection from a protected historical snapshot. <br> • From the Intelligent Update Manager dialog box, select the protected snapshot in the list, and then click Unprotect. |
| Set a target snapshot | Set a historical snapshot as the target snapshot when a device is running an earlier application version needing modification. <br> • From the Intelligent Update Manager dialog box, select the required snapshot in the list, and then click Set Target Snapshot. |
| Delete a snapshot | Delete an unprotected target snapshot. <br> • From the Intelligent Update Manager dialog box, select the required unprotected snapshot in the list, and then click Delete. |

## See also

Intelligent Online Updates on page 569
Intelligent Update Options on page 572
Intelligent Update Manager Report on page 576
TIC Version Checker on page 636
Access diagnostic information on page 559
Intelligent Update Options
Enable intelligent online updates from the Intelligent Update Manager Options dialog box. By default, intelligent updates are disabled for new
projects. Imported projects retain their Enable Intelligent On-line Updates setting.

```
Options
Enable Intelligent On-line Updates
```

x
$\square$ Inform user when On-line update is not possibleInform user when Intelligent Update will be usedAlways display $\log$ when building download imageAlways display $\log$ when updating the target snapshot or history fileShow temporary variables in $\log$ when building download image
1 Number of non-protected historical snapshots to keep

Display Match Function Blocks information dialog

When the intelligent online update feature is enabled, choose to enable these options:

Tip: It is recommended to enable all intelligent update options.

- Inform user when online update is not possible. The Intelligent Update Manager determines whether an online update is not possible.
- Inform user when Intelligent Update will be used. The Intelligent Update Manager determines whether an online update is possible and that an intelligent update is required since the data memory space has changed.
- Always display log when building download image. Displays the log generated when building the download image with modifications requiring an intelligent update.
- Always display log when updating the target snapshot or history file. Displays the log generated when updating the target snapshot or history file (at download/update time). When not checked, the log is only displayed when an error occurs.
- Show temporary variables in log when building download image. Displays the log generated when building the download image automatically including generated variables in the list of changed variables. For temporary variables, may reset function block internal variables to their initial values.
- Number of non-protected historical snapshots to keep. The number of historical snapshots retained in the snapshot list. Protected snapshots and the current snapshot are not included in the maximum user-defined quantity. Exceeding the defined number of snapshots
deletes the oldest snapshot. The default value is 20 unprotected snapshots.
- Display Match Function Blocks information dialog. Displays an informative dialog box before displaying the Match Function Blocks dialog. This dialog warns that some function blocks are not matched and that unmatched function blocks lose previously stored data.


## To enable intelligent update options

1. From the Build menu, click Intelligent Update Manager.
2. From the Intelligent Update Manager dialog box, click Options...
3. In the Options dialog box, select Enable Intelligent On-line Updates.
4. (optional) Select the following options (selecting all options is recommended):

- Inform user when On-line update is not possible
- Inform user when Intelligent Update will be used
- Always display log when building download image
- Always display log when updating the target snapshot or history file
- Show temporary variables in log when building download image
- Display Match Function Blocks information dialog

5. In the Number of non-protected historical snapshots to keep, set the required value.
6. Click OK.

## See also

Intelligent Online Updates on page 569
Intelligent Update Manager on page 571
Manual Function Block Matching on page 574

# Manual Function Block Matching 

The Intelligent Update Manager identifies function blocks using the names of the variables wired to their inputs. When the inputs match, the Intelligent Update Manager seamlessly transfers the values of local variables inside the function block to the new version of the application.

The Intelligent Update Manager is unable to match inputs wired from other blocks or corner connections since these wires are assigned temporary names possibly changing at each compilation. When the manager identifies instances
of unmatched function blocks, the Match Function Blocks dialog box displays showing these instances.


Use the Match Function Blocks dialog box to manually match the snapshot version of the function block with the modified version. The leftmost column displays the program and its function blocks needing matching. Select the function block to match in this first column. Use the Previous and Next buttons to scroll between the different programs with function blocks that need matching.

The Target column (second column) displays the Target version function block while the third column displays the calling parameters (inputs) of the Target function block. The Workbench column (fifth column) displays the potentially matching version of the function block while the fourth column displays the calling parameters of the Workbench function block. Match the blocks by selecting the equivalent instances in the Target and Workbench columns. The columns displaying the calling parameters help identify an equivalent match. Upon making a correct selection, click the Match button to inform the Intelligent Update Manager that these blocks are equivalent. A green checkmark displays beside the matched instances. You can also unmatch selections.

Exit manual function block matching. An online update is no longer possible after exiting the Match Function Block dialog box. After exiting manual function block matching, recompile the project.

## Tip: When the Intelligent Update Manager detects unmatched blocks at download, the Match Function Block dialog box displays.

## To manually match function blocks

1. From the Match Function Blocks dialog box, in the first column, select the function block needing matching.

The Target and Workbench columns (columns two and five) display the unmatched instances of the function block.
2. In the Target and Workbench columns, select the matching instances of the function block. Use the third and fourth columns, displaying the inputs connected to the function block, as a reference to ensure the correct pair is matched.
3. Click Match.

The matched function block instances have a green check-mark beside them.
4. Click Next and Previous to scroll through the different programs having unmatched function blocks.
Tip: Abort the matching process to cancel all matches. Aborting eliminates the possibility of online updates. Matching function blocks becomes available after recompiling the application.
5. Repeat steps 1 through 4 until all function blocks are matched, then click Save and Continue.

The matching process is complete.

## See also

Intelligent Online Updates on page 569

Intelligent Update Manager Report

The Intelligent Update Manager creates an action log each time an intelligent update is required. The log displays when an error occurs or at each compilation when the Always display log... options are enabled. Since the report only details information about intelligent updates, instances of NO CHANGE display when no change occurred or a minor modification occurred only requiring a regular update. The report does not include changes to variable attributes.

## See also

Intelligent Update Manager on page 571
Intelligent Update Options on page 572
To access the System Configuration Tool
From the View menu, click System Configuration Tool.
To use Trusted ${ }^{\circledR}$ controllers, configure the system using the System Configuration Tool. Use the System Configuration Tool to create the SYSTEM.INI text file used to configure the operational parameters within a Trusted controller.

From the System Configuration Tool, load existing SYSTEM.INI files, save new or modified files, erase the buffer, and import I/O connection files.

| Task | Procedure |
| :--- | :--- |
| Load the buffer from a file | 1. From the System Configuration Tool, in the File menu, click Load Buffer From File. <br> 2. In the Load.INI File dialog box, browse and select the required *.ini file, and then click Open. |
| Save the buffer to a file | 1. From the System Configuration Tool, in the File menu, click Save Buffer To File. <br> 2. In the Save Buffer dialog box, define the required name and location for the file, and then click Save. |
| Erase the buffer | From the System Configuration Tool, in the File menu, click Erase Buffer. |
| Import an I/O connection file | 1. From the System Configuration Tool, in the File menu, click Import.... <br> 2. From the Select file to Import dialog box, browse and select the required appli.cnx connection table source code file, <br> and then click Open. |



Tip: Even if a controller contains two TMR processors, the System Configuration Tool only displays one.

The System Configuration Tool displays the system layout of the Trusted controller chassis. The layout displayed in the System Configuration Tool must match the physical controller.

To configure the Trusted controller, perform the following from the System Configuration Tool:

- Create and configure system templates
- Configure the Trusted TMR Processor
- Assign modules to the Trusted System
- Configure I/O modules
- Configure the Communications Interface module
- Set the Communications port
- Generate the SYSTEM.INI
- Download the SYSTEM.INI
- Upload the SYSTEM.INI

Tip: $\quad$ To configure the controller, the maintenance enable key switch must be in the Maintain position. You must connect your workstation to the Trusted TMR Processor front panel diagnostic port using the TC-304-01 Serial Maintenance Cable or via Ethernet using an 8153 Communications Interface Adapter located on the back of the T8151B Communication Interface.

## See also

Managing Templates on page 578
Configuring the SYSTEM.INI File on page 598
Generating the SYSTEM.INI file on page 603
Setting the Communications Port on page 609

## Managing Templates

## To access the System Template Editor

In the System Configuration Tool, from the Template menu, click Manage Templates.

Before adding I/O modules to the system, configure the templates to be used for the required modules. Templates contain segments of configuration. Add, delete, modify, duplicate, archive, and restore templates from the System Template Editor. Each template performs a specific function and can only be assigned to one module type. A specific template can be used by multiple modules of the assigned type.

| Task | Procedure |
| :--- | :--- |
| Add a template | 1. From the System Template Editor, click New. <br> 2. From the Template Creation dialog box, enter the following information: <br> - Template Name <br> - Module Type <br> - Template Type <br> 3. To define the template, click Edit Template. |
|  | 4. To accept the template design, click OK. <br> 5. From the Template Creation dialog box, enter a description and then click OK. <br> 6. To close the System Template Manage, click OK. |
| Modify a template | 1. In the Template Creation dialog box, select the template to modify in the Existing Templates list, and then click <br> Edit. |
|  | 2. From the Template Creation dialog box, edit the required parameters, and then click OK. <br> 3. To close the System Template Manager, click OK. |
| Duplicate a template | 1. In the Template Creation dialog box, select the template to duplicate in the Existing Templates list, and then click <br> Duplicate. |
|  | 2. From the Template Creation dialog box, edit the required parameters, and then click OK. <br> 3. To close the System Template Manager, click OK. |
| Delete a template | 1. In the Template Creation dialog box, select the template to remove in the Existing Templates list, and then click <br> Delete. |
| 2. Click Yes to confirm the removal. |  |

Create these types of templates with the System Template Editor:

| Template | Modules | Description |
| :--- | :--- | :--- |
| Threshold template | $8402,8403,8423,8424,8431,8432$, | Defines the state of selected I/O module channels that depend on the input signal to the |
|  | $843,, 8448,8449$ | channel. |
| LED template | $8403,8423,8424,8431,8433,8442$, <br>  <br>  <br>  <br>  <br> $8444,8448,8449,8451,8461,8471$, <br> $8472,8473,8480$ | Defines the color and mode (flashing, steady, or off) of the channel LEDs for the different <br> channel states. Possible values are Off, Red, Green, Red Flash, or Green Flash. |


| Template | Modules | Description |
| :---: | :---: | :---: |
| Force template | $\begin{aligned} & 8402,8403,8423,8424,8431,8432, \\ & 8433,8444,8448,8449,8451,8461, \\ & 8471,8472,8473,8480 \end{aligned}$ | Defines the state the unconnected channels must be forced into to provide a healthy (unalarmed) system state. |
| System template | $\begin{aligned} & 8402,8403,8423,8424,8431,8432, \\ & \text { 8433, } 8442,8444,8448,8449,8451, \\ & \text { 8461, 8471, 8472, 8473, 8480, All } \\ & \text { Module Types } \end{aligned}$ | Defines the timeout intervals for the Watchdog and IMB (Inter-Modular-Bus) of the I/O module. |
| Flags template | $\begin{aligned} & 8402,8403,8423,8424,8431,8432, \\ & 8433,8442,8444,8448,8449,8451, \\ & 8461,8471,8472,8473,8480 \\ & \hline \end{aligned}$ | Defines the flag settings for the I/O module. |
| Filter template | $\begin{aligned} & 8402,8403,8423,8424,8431,8432, \\ & 8433 \end{aligned}$ | Defines the internal filtering values for the I/O module. |
| Additional CLI | 8402, 8403, 8423, 8424, 8431, 8432, 8433, 8442, 8444, 8448, 8449, 8451, 8461, 8471, 8472, 8473, 8480, All Module Types | Defines additional CLI (Command Line Interface) entries that are external to the configuration data. |
| Speed Monitor template | 8442 | Defines the operating parameters required by the speed monitor. |
| Shutdown template | $\begin{array}{\|l} 8444,8448,8449,8451,8461,8471, \\ 8472,8473,8480 \\ \hline \end{array}$ | Defines the output action required when the module enters the shutdown state. |
| Channel Type template | 8448 | Defines the direction (input or output) for the channels of the module. |
| De-energized Short Circuit template | $\begin{aligned} & 8448,8449,8451,8461,8471,8472, \\ & 8473 \end{aligned}$ | Enables short circuit detection for de-energized outputs. |

Tip: Templates created with the All Module Types are not restricted to a specific module type; apply the template to any I/O module.

From the System Template Editor, view the list of existing templates. The Quick View pane displays the details of an existing template.


## See also

Threshold Templates on page 580
LED Templates on page 582
Force Templates on page 586
System Templates on page 587
Additional CLI Templates on page 593

## Threshold Templates

The threshold template displays the current voltage thresholds for the module. When adding or modifying a threshold template, apply the same value to all channels of a module or define individual values for each channel.

Input modules monitor and calculate the voltage level from the field at each channel to determine the appropriate state to report to the TMR Processor. After calculating the input channel voltage, a state is then determined based on the channel threshold settings. There are eight possible states ( 0 to 7 ). The following example displays state thresholds for a line monitored digital input switch configuration:


Tip: $\quad$ States 0 to 6 are based on the calculated voltage. State 7 is reported when the module completely fails.

There are fixed minimum (Tmin) and maximum (Tmax) thresholds for the module. Each channel has eight configurable thresholds. Each state transition has a hysteresis, as seen in gray in the example above.

Apply the following rules when configuring thresholds:

- Each threshold is greater than or equal to the previous threshold, for example, T 2 is greater or equal to $\mathrm{T} 1, \mathrm{~T} 3^{3} \mathrm{~T} 2$ etc...
- The state associated with each threshold is between the threshold settings, except that the lower threshold is higher than Tmin (in the valid OC state range), and the upper threshold is lower than Tmax (in the valid SC state range).
- Threshold values are defined as fractions of a volt. For example, in the 8403 Digital Input Module, thresholds are represented as 512 counts per volt; for example, a value of 8400 represents 16.406 volts (8400/512). The various Product Descriptions define the threshold fractions used for applicable modules.

When using the Threshold Template Editor, use the following syntax to define the same threshold values for all channels:
default = <T1>, <T2>, <T3>, <T4>, <T5>, <T6>, <T7>, <T8>
Use the following syntax to define individual channel thresholds:
channel\# = <default> | <T1>, <T2>, <T3>, <T4>, <T5>, <T6>, <T7>, <T8>


The bar graph displays the current voltage thresholds. Modify the values by either dragging the horizontal lines on the bar graph or by entering the voltage values. Use the scroll bar to display the threshold settings.

| Task | Procedure |
| :--- | :--- |
| Define the threshold values for all channels of <br> a module | The displayed default threshold values are dependent on the module type. <br> 1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the <br> Threshold Template Type, and then click Edit Template. <br> 2. To modify the default values for the thresholds, in the ThreshForm dialog box, set the required Volts values. <br> 3. To apply the threshold values to all channels of the module, click Set All, and then click OK. |
| Define the threshold value for a channel | 1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the <br> Threshold Template Type, and then click Edit Template. |
|  | 2. To modify the default values for the thresholds, in the ThreshForm dialog box, set the required Volts values. <br> 3. To apply the threshold values to one channel of the module, in the Channel Action section, select the required <br> channel number, then click Set. <br> 4. Enter threshold values for all required channels, and then click OK. |


| Task | Procedure |
| :--- | :--- |
| Remove the threshold values | 1. If the same value was applied to all channels, click Remove, and then OK. <br> 2. If individual values were applied to each channel, perform the following: <br> a. In the Channel Action section, select the required channel number, and then click Remove. <br> b. Remove all channels as required, and then click OK. |

## LED Templates

## See also

LED Templates for Input Modules on page 582
LED Templates for Output Modules on page 584

## LED Templates for Input Modules

LED templates define the mapping between the front panel indicators for the specific module and the input or output state. LED templates are available for the following:

- Input Modules
- Output Modules
- Zone Interface Modules


## LED Templates for Zone Interface Modules on page 585

The LED template defines the mapping between the input state and the front panel indicators for the specific module. Use the LED State Editor to map the required operation for all input channels. An LED mapping entry is made using
the following syntax:
state\# $=<$ OFF | RED | GREEN [+FLASH]>
If no entries are made in this section, the module will operate with the default LED mapping. The following example displays the default values for the 8403 Digital Input Module, which are suitable for inputs without line monitoring devices installed.

| State for Input <br> Modules | State for 8449 <br> Module | Default Value | Description |
| :--- | :--- | :--- | :--- |
| 0 | 8 | $=$ Red | Out of range |
| 1 | 9 | $=0 \mathrm{ff}$ | Open circuit |
| 2 | 10 | $=0 \mathrm{ff}$ | Open field contact |
| 3 | 11 | $=0 \mathrm{ff}$ | Indeterminate contact state |
| 4 | 12 | $=$ Green | Closed field contact |
| 5 | 13 | $=$ Green | Short circuit |
| 6 | 14 | $=$ Green | Over range |
| 7 | 15 | $=$ Off |  |
| 8 | - | $=$ Red |  |

Tip: $\quad$ State 7 represents a module fault. States 8 through 15 all represent channel fault states (except for the 8449 Valve Monitor module). For the 8449 module, states 8 through 15 are used differently since the valve outputs and position inputs are grouped in pairs.

LED templates define specific LED states. For example, if inputs are installed with line monitoring devices, the LED mapping entries can be made to flash red for the corresponding states that indicate a line fault condition, such as Open circuit, Indeterminate and Short circuit.

## To configure the color and operating modes of the LED front panel indicators

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the LED Template Type, and then click Edit Template.

| LED State Editor |  |
| :--- | :--- | :--- |
| State 0 | State 2 |

The LED State Editor displays.
2. To modify the color and operating mode of the channel, click the required state LED icon, and then click one of these:

- Off
- Red
- Green
- Red Flash
- Green Flash

The modified channel LED of the input module displays.
3. In the LED State Editor, configure the required input channel states, and then click $\mathbf{O K}$.

## See also

LED Templates for Zone Interface Modules on page 585

## LED Templates for Output Modules

The LED template defines the mapping between the output state and the front panel indicators for the specific module. Use the LED State Editor to map the required operation for all output channels. An LED mapping entry is made using the following syntax:

```
state# = <OFF | RED | GREEN [+FLASH]>
```

If no entries are made in this section, the module will operate with the default LED mapping. The following example displays the default values for the 8451 Digital Output Module, which are suitable for outputs without line monitoring devices installed.

| State for Output <br> Modules | State for 8449 Module | Default Value | Description |
| :--- | :--- | :--- | :--- |
| 0 | 0 | $=$ Off | Not used |
| 1 | 1 | $=$ Red | No field supply voltage |
| 2 | 2 | $=$ Off | Output de-energized |
| 3 | 3 | $=$ Green | Line open circuit |
| 4 | 4 | $=$ Green | Output energized (on) |
| 5 | 5 | $=$ Red | Field short circuit |
| 6 | 6 | $=0 \mathrm{ff}$ | Not used |
| 7 | 7 | $=0 \mathrm{ff}$ | Not used |
| 8 | - | $=$ Red |  |

Tip: $\quad$ State 7 represents a module fault. States 8 through 15 all represent channel fault states (except for the 8449 Valve Monitor module). For the 8449 module, states 8 through 15 are used differently since the valve outputs and position inputs are grouped in pairs.

## To configure the color and operating modes of the LED front panel indicators

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the LED Template Type, and then click Edit Template.


The LED State Editor displays.
2. To modify the color and operating mode of the channel, click the required state LED icon, and then click one of these:

- Off
- Red
- Green
- Red Flash
- Green Flash

The modified channel LED of the output module displays.
3. In the LED State Editor, configure the required output channel states, and then click OK.

## See also

LED Templates for Input Modules on page 582
LED Templates for Zone Interface Modules on page 585

LED Templates for Zone
Interface Modules

Trusted ${ }^{\circledR} 8448$ Zone Interface Modules can have each channel defined as an input or an output using the Channel Type Templates. The LED State Editor enables configuring individual channels with an LED state/color map.

When configuring the channels, apply the defined default values for the channel or set different values. Channels using the default LED mapped state are identified with $\mathrm{D}_{\text {. }}$.

| Task | Procedure |
| :---: | :---: |
| Configure the LED states for the channels of the 8448 Zone Interface module | 1. From the Template Creation dialog box, enter the template name, select the 8448 Module Type, then select the LED Template Type, and then click Edit Template. <br> The LED State Editor displays. <br> 2. To define the default LED state values, select Default in the list of channels, then for each state click the required LED icon, and then click one of the following: <br> - Off <br> - Red <br> - Green <br> - Amber <br> - Red Flash <br> - Green Flash <br> - Amber Flash <br> The default LED color values are mapped for each state. <br> 3. To apply the default mapped state values to channels, select the required channel and then select Use Default from the Channel Options. <br> 4. To apply specific mapped state values to a channel, perform the following: <br> a. Select the required channel and then select Use Editor from the Channel Options. <br> b. For each state, click the required LED icon, and then click one of the following: <br> - Off <br> - Red <br> - Green <br> - Amber <br> - Red Flash <br> - Green Flash <br> - Amber Flash <br> 5. Configure all required channels and then click OK. |
| Assign a specific channel mapping as the default mapping | - Select the channel having the specific mapping, and then click Default <- Current. |
| Assign the default mapping states to a channel retained as specific | - To assign the default mapping state values to a channel having specific mapping, select the required channel, and the click Current <- Default. <br> The channel still has a specific LED mapping but now uses the default states. |

## See also

## Channel Type Templates on page 596

## Force Templates

The Force template enables forcing the input or output channel states onboard the module to a specific value irrespective the current value. Channels may be forced to a specified state on a per-channel basis. The states that exist on a module are specific to that module. Use the Forcing Editor to add, remove, or modify Force templates.


Example: channels on the 8431 Analog Input Module can be in any one of the following states:

| State | Description |
| :--- | :--- |
| 0 | Under range |
| 1 | Low-low |



In this example, channels 1, 11, and 22 are forced to the High state (4).

## To configure the Force Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Force Template Type, and then click Edit Template.

The Forcing Editor displays.
2. To add a channel force to the template, select the required channel number in the Channel Select window, then enter the required channel state in the Channel State window, and then click Add.
3. To remove a channel from the template, select the channel, and then click Remove. A warning message confirms the removal of the correct force.
4. To modify the state of a channel, select the required channel, then enter the new value in the Channel State window, and then click Change.
5. Configure all required channels, and then click OK.

## See also

Trusted TMR System Safety Manual

## System Templates

The System template enables configuring the following information:

- No-Load Threshold
- Inter-Modular Bus Timeout
- Watchdog Timeout
- Power Fail Timeout
- Bypass Timeout


## To configure the System Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the System Template Type, and then click Edit Template.


The Module dialog box displays.
2. In the Module dialog box, do the following:
a. To add an IMB Timeout, enter the required value in milliseconds, and then select Add.
b. To add a Watchdog Timeout, enter the required value in milliseconds, and then select Add.
c. To add a no-load threshold, in the Miscellaneous section, enter the required value by using the following syntax:

```
nlthresh = < group1 >, < group2 >,< group3 >,< group4
```

$>$ [,<group5 >]
d. To add a Power Fail timeout, in the Miscellaneous section, enter the required value by using the following syntax:

```
PWRFAILTO = < value >
```

e. To add a Bypass timeout, in the Miscellaneous section, enter the required value by using the following syntax:

```
BYPASSTO = < value >
```

3. To complete the configuration, click OK.

## See also

No-Load Threshold on page 589
Inter-Modular Bus Timeout on page 590
Watchdog Timeout on page 590

## Power Fail Timeout on page 591

Bypass Timeout on page 591

## No-Load Threshold

The no-load threshold template is used for output modules and uses the following syntax:

```
nlthresh = < group1 >, < group2 >,< group3 >,< group4 >
[,< group5 >]
```

The template sets the current per slice (in milliamps) below which an output signals state 3 (no load), and each of the parameters sets the threshold for an entire power group of eight digital outputs. Without this template, the defaults are 10 mA . For example, to recognize the load requires at least 30 mA .

Outputs do not share current evenly across the three TMR slices and there may be a significant difference between slices. As an example, the 8451 output module has a minimum on state load current of 50 mA per channel. With a load of 51 mA , the load may be spread across the slices as follows:

| Channel 1 | Slice A | 26 mA |
| :--- | :--- | :--- |
| Channel 2 | Slice B | 16 mA |
| Channel 3 | Slice C | 9 mA |

While the channel total is greater than the nominal minimum, slice $C$ is below the default no-load threshold (10mA) and signals a no-load state while the other two slices record an acceptable load. This causes a discrepancy alarm. In earlier firmware versions this may lead to a module shutdown.

It is recommended to group comparatively high and low power loads on different power groups, preventing potential noise from a higher powered output affecting the lower powered signals. The following no-load threshold settings are recommended for each group:

| 25 to 80 mA | nlthresh $=5$ |
| :--- | :--- |
| 80 to 150 mA | nlthresh $=10$ |
| 150 to 300 mA | nlthresh $=15$ |
| 300 mA to limit | nlthresh $=20$ |

As an example, if groups 1 and 2 (channels 1 to 16 ) are approximately 100 mA per channel and the rest are 50 mA per channel, the no-load thresholds should be set as:
nlthresh $=10,10,5,5,5$
Output modules such as the 8471 ( 120 Vdc ) module only have four output groups and the template is written, for example, as:
nlthresh $=5,5,5,5$
While some output modules have a minimum on state load current of 20 or 25 mA per channel, is it still recommended to use the nlthresh template.

For the 8473 module, you are recommended to set nlthresh = 20, 20 because the module is unable to accurately detect the no-load condition below this value.

## See also

System Templates on page 587

## Inter-Modular Bus Timeout

The Inter-Modular Bus Timeout (IMBTO) is a timer run internally within each I/O module from start-up. The following syntax is used for the Inter-Modular Bus Timeout:

```
IMBTO = < value >
```

If the TMR Processor does not scan the module within the programmed time, the module adopts the shutdown mode. In the majority of Trusted Systems, omit using this timeout and instead use the default value. In systems that are very large, or for test purposes, for example to pause application program, adjust the timeout period. The timeout period is defined in milliseconds; a value of zero disables the timeout.

## Tip: This value MUST NOT be set to zero for operational systems.

For the IMB timeout, it is recommended to poll the module at least four times at the normal scan time within the timeout, taking scheduled polling into account. Example: if the current scan time is 200 milliseconds and the module is set for scheduled polling at every 5th scan, set the timeout to $200 \times 5 \times 4=$ 4000 ms minimum. Ensure that the timeout is still safe by setting the timeout to less than half the process safety time.

## See also

System Templates on page 587

## Watchdog Timeout

The WDOGTO (System Watchdog Timeout) is a timer run internally within each I/O module from start-up. The following syntax is used for the Watchdog Timeout:

```
WDOGTO = < value >
```

If the TMR Processor does not pet the module's watchdog within the programmed time, the module adopts the shutdown mode. However this parameter adjusts the system watchdog timeout interval. It is not normally required to adjust this parameter unless the system is very large or to allow certain tests to be performed. The timeout value is entered in milliseconds, and a value of zero disables the timeout.

## Tip: This value MUST NOT be set to zero for operational systems.

For the Watchdog timeout, it is recommended to poll the module at least four times at the normal scan time within the timeout, taking scheduled polling into account. For example, if the current scan time is 200 milliseconds and the module is set for scheduled polling at every 5th scan, then the timeout needs to be set to $200 \times 5 \times 4=4000 \mathrm{~ms}$ minimum. Ensure that the timeout is still safe. For example, the timeout is less than half the process safety time.

## See also

## System Templates on page 587

## Power Fail Timeout

## Bypass Timeout

## Flags Templates

## See also

System Templates on page 587
The Flags Template enables configuring the following operations for input modules:

| Operation | Syntax | Description |
| :--- | :--- | :--- |
| Currentsink | Currentsink = < TRUE \| FALSE (default) > | Defines if the inputs are high or low side switching. For example, switch to $+V$ or OV <br> (return). The default option is for high-side switch (to $+V$ ). Setting this option to TRUE <br> selects the switch operation to OV (return). |
| Ratiometric | Ratiometric = < TRUE \| FALSE (default) > | Enables the input thresholds to be defined either in terms of their absolute voltage or <br> normalized to the nominal field supply voltage. The default operation is to define the <br> thresholds in absolute voltage. <br> When the ratiometric threshold operation is enabled, the field supply must be <br> connected to the monitoring terminals on the field termination unit. In ratiometric <br> mode the range of voltages for the thresholds is 0 to $24 V$, the value is re-scaled <br> according to the current field supply voltage. For example, a threshold value of 12 V with <br> an actual field supply of $30 V$ results in an actual threshold of $15 V$. If the field supply <br> voltage is out of range, a line fault condition is generated. |

The Flags Template enables configuring the following operation for output modules:

| Operation | Syntax | Description |
| :--- | :--- | :--- |
| Logical | LOGICAL = < TRUE \| FALSE (default) > | Defines how the module timestamps changes of state for the output channel. When set to <br> True, the module timestamps an output change when it receives a change of state to the <br> logical output control signal. When set to False (default) the module timestamps an output <br> change of state when the measured voltage and current reflect an actual change of state in <br> the output control circuit. Propagation delay between receiving the logic control signal and <br> driving the output control signal is typically less than 1 millisecond. For most applications, it <br> is usually unnecessary to change this setting. |

## To configure the Flags Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Flags Template Type, and then click Edit Template.


The CLI Entry dialog box displays.
2. In the CLI Entry dialog box, do the following:
a. To add an instruction, in the Please Enter CLI Instruction text box, type the required operation syntax, and then click Add Entry.

## b. To delete an instruction, select the entry to remove and then click Remove Entry.

3. To complete the Flags template configuration, click OK.

## See also

## Filter Templates on page 592

## Filter Templates

The Module Filter template configures the filter type to use for each of the 40 channels on an input module. There are two types of entries for this template: default and channel settings.

The default entry sets the default filter type for all channels on the module and uses the following syntax:

```
Default = < 0 | 1 >
```

Also configure input filtering for individual channels by using the following syntax:

```
channel# = < default > | < 0 | 1 >
```

The module supports two input filter types, relating to the numbers in the syntax for individual channels:

| Filter <br> Type | Filter Size | Description |
| :--- | :--- | :--- |
| 0 | 4 ms filter | The samples are passed straight through with only a 4ms signal averaging filter. The sampling frequency depends on <br> whether another filter has been selected for any other channels. The default sampling frequency is 1.0ms. |
| 1 | $4 m s$ filter, same as 0 | The samples are passed straight through with only a 4ms signal averaging filter. The sampling frequency depends on <br> whether another filter has been selected for any other channels. The default sampling frequency is $1.0 \mathrm{ms}$. |

Since the propagation delays of the filters are a known constant, when a change of input state passes through the filter, the corresponding SOE event time stamp is adjusted by the propagation delay.

## To configure the Filter Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Filter Template Type, and then click Edit Template.


The CLI Entry dialog box displays.
2. In the CLI Entry dialog box, do the following:
a. To add an instruction, in the Please Enter CLI Instruction text box, type the required operation syntax, and then click Add Entry.
b. To delete an instruction, in the Current CLI Entries list, select the entry to remove and then click Remove Entry.
3. To complete the Filter template configuration, click OK.

## See also

Additional CLI Templates on page 593

## Additional CLI Templates

## To configure an Additional CLI template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Additional CLI Type, and then click Edit Template.


The CLI Entry dialog box displays.
2. In the CLI Entry dialog box, do the following:
a. To add an instruction, in the Please Enter CLI Instruction text box, type the required operation syntax, and then click Add Entry.
b. To delete an instruction, in the Current CLI Entries list, select the entry to remove and then click Remove Entry.
3. To complete the Additional CLI template configuration, click OK.

## See also

## Managing Templates on page 578

## Speed Monitor Templates

To use the T8442 TMR Speed monitor module, configure the Speed Monitor template.

[^10]
## To display the Speed Monitor Configuration

- From the Template Creation dialog box, enter the template name, select the 8442 Module Type, then select the Speed Monitor template Type, and then click Edit Template.



## See also

## Trusted Product Descriptions and Manuals on page 860

## Shutdown Templates

The shutdown template enables configuring individual shutdown states for each channel of an output module. Output channels are driven to the default fail-safe state when the module enters the shutdown mode. Typically the module enters the shutdown mode when any of the system timeouts occur, or when commanded by the TMR Processor. A channel can be configured to enter a specific shutdown state other than the default using the Shutdown template.

A shutdown entry is entered using the following syntax:

```
channel#=< ON | OFF | HOLD >
```

The available shutdown modes are:

| Mode | Description |
| :--- | :--- |
| ON | Energize the output |
| OFF | De-energize the output |
| HOLD | Hold, or maintain the output in the last commanded state |

All output channels used for safety critical applications must be configured with a shutdown mode that places the process in the safe state.

## To configure the Shutdown Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Shutdown Template Type, and then click Edit Template.

| Shutdown Editor |  |
| :---: | :---: |
|  | $\qquad$ <br> Cancel Energised De-energised Hold Unconfigured $\square$ Default |

The Shutdown Editor displays.
2. In the Shutdown Editor, to define the default shutdown value, rightclick Default in the list of channels, and then click one of the following:

- On
- Off
- Hold
- Unconfigured

The default shutdown value is defined.
3. In the Shutdown Editor, to define the shutdown value for a channel, right-click the required channel and then click one of the following:

- On
- Off
- Hold
- Unconfigured
- Default

4. Configure all required channels and then click OK.

## See also

The Channel Type template defines the Trusted ${ }^{\star} 8448$ Zone Interface Module channel configuration. Each channel on a Zone Interface Module can either be an input or output. The default configuration is with all channels set as inputs.

## To configure the channel type for a Trusted 8448 Zone Interface Module

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the Channel Type Template Type, and then click Edit Template.


The Channel Type window displays.
2. In the Channel Type window, to change the channel type, right-click the required channel and then click Input or Output.

Input channels are identified with $\mathbf{\square}$ and output channels with $\mathbf{0}$.
3. Configure all required channels and then click OK.

## See also

## System Templates on page 587

## De-energized Short Circuit Templates

By default, output modules do not detect for short circuit faults when deenergized. Using the De-energized Short Circuit template, configure output modules to detect de-energized short circuit faults on some or all channels. The impedance below which a de-energized short circuit fault is reported is 40 ohms. The cold resistance of small incandescent lamps may be less than this threshold.

[^11]
## To configure the De-energized Short Circuit Template

1. From the Template Creation dialog box, enter the template name, select an applicable Module Type, then select the De-energised Short Circuit Template Type, and then click Edit Template.

| De-energised Short Circuit Detection |  |  |  |
| :---: | :---: | :---: | :---: |
| Channel | Level (0hms) | - |  |
| Charnel 1 | Off |  | $\checkmark$ OK |
| Channel 2 | Off |  |  |
| Channel 3 | Off |  | X Cancel |
| Channel 4 | Off |  |  |
| Channel 5 | Off |  |  |
| Channel 6 | Off |  | ? Help |
| Channel 7 | Off | 三 |  |
| Channel 8 | Off |  | Select All |
| Channel 9 | Off |  |  |
| Channel 10 | Off |  |  |
| Channel 11 | Off |  |  |
| Channel 12 | Off |  |  |
| Channel 13 | Off |  |  |
| Channel 14 | Off |  |  |
| Channel 15 | Off |  |  |
| Channel 16 | Off |  |  |
| Channel 17 | Off |  |  |
| Channel 18 | Off |  |  |
| Channel 19 | Off |  |  |
| Channel 20 | Off |  |  |
| Channel 21 | Off |  |  |
| Channel 22 | Off |  |  |
| Channel 23 | Off |  |  |
| Channel 24 | Off | - |  |

The De-energised Short Circuit Detection window is displayed.
2. In the De-energised Short Circuit Detection window, do the following:
a. To modify the channel line monitoring, right-click the required channel, and then click Off or $\mathbf{4 0}$.

Also select all channels to set the same value for all channels.
b. To set the same value for all channels, click Select All, right-click in the channel list, and then click Off or 40.
3. Configure all required channels and then click OK.

## See also

## Channel Type Templates on page 596

Configuring the SYSTEM.INI File

The SYSTEM.INI text file configures the operational parameters of a Trusted ${ }^{\circledR}$ controller. To configure the SYSTEM.INI file, you need to have created all required system templates. You also need to configure the communication port to download the SYSTEM.INI file to the TMR Processor. To configure the SYSTEM.INI file, you need to do the following:

- Configure the Trusted TMR Processor
- Assigning modules and adding the expander chassis
- Configuring I/O modules
- Configure the 8151 Communications Interface module


## See also

Configuring the Trusted TMR Processor on page 599
Assigning I/O Modules to the Trusted System on page 600
Configuring I/O Modules for Trusted Applications on page 601
Configuring the 8151 Communications Interface Module on page 602
Generating the SYSTEM.INI file on page 603

## Configuring the Trusted TMR Processor

Access the module to configure the 8110 TMR Processor.
Tip: $\quad$ For information on configuring the Trusted ${ }^{\circledR}$ TMR Processor, refer to the T8110B Trusted TMR Processor product description.

## To access the 8110 TMR Processor configuration

- From the System Configuration Tool, in the System Layout, click the 8110 TMR Processor module in the first slot.



## See also

Trusted TMR Processor

Assigning I/O Modules to the Trusted System

To configure the SYSTEM.INI file, add I/O modules to the system layout of the Trusted ${ }^{\circledR}$ controller. At least one 8151 Communication Interface module must also be added. If additional slots are required for the system, add an expander chassis. When adding an expander chassis, a pair of 8311 Expander Interface modules must also be included in the controller chassis. All Expander Chassis must be connected to the Expander Interface modules.

When adding modules, the layout that displays in the System Configuration Tool must match the layout of the physical Trusted controller and expander chassis.

It is recommended to disable slots with no configured modules and that will never be used as companion slots or hot swaps. Disabling these slots removes them from the scanning process and reduces the processing times. Disabling a slot disables all communications for that slot; you can no longer hot swap a module or perform diagnostics on any module moved to the disabled slot. For this reason, do not disable any companion slots or smart slots and leave one slot active as a diagnostic slot.

| Task | Procedure |
| :---: | :---: |
| Add a module to a slot | 1. From the System Configuration Tool, right-click an empty slot in the chassis. <br> The Replace Item In Slot... dialog box display. <br> 2. In the Replace Item In Slot... dialog box, select the required module and then click OK. The module is added to the system layout and requires configuration. |
| Empty a slot | 1. From the System Configuration Tool, right-click the slot to empty in the System Layout. The Replace Item In Slot... dialog box displays. <br> 2. In the Replace Item In Slot... dialog box, select Empty Slot and then click OK. |
| Add and connect an expander chassis | 1. In the System Configuration Tool, right-click the empty space. <br> The Insert New Chassis dialog box displays. <br> 2. In the Insert New Chassis dialog box, select the required Logical Chassis Number and then click OK. <br> Tip: Add a maximum of 27 expander chassis. The number of added chassis must not exceed the maximum number defined in the parameters of the Processor. <br> The expander chassis, including the two 8310 Expander Processor modules, display in the System Layout. <br> 3. To connect the Expander chassis and controller chassis, add a pair of 8311 Expander Interface modules to the controller chassis. It is recommended to use slot 1 and 2. <br> 4. Click the edge of the expanded chassis, then from the Chassis Connection dialog box, select a slot having an Expander Interface module, and then click OK. Selecting slot 0 disconnects the chassis. <br> Tip: If no Expander Interface modules are added, an Expander Interface module is automatically created at the slot defined in the Chassis Connection dialog box. |
| Connect or disconnect an expander chassis | 1. In the System Configuration Tool, click the required 8311 Expander Interface module. <br> The Expander Interface Parameters dialog box displays. <br> 2. To connect the chassis, from the Unconnected Chassis section, drag the unconnected chassis icon to a Not Connected channel in the Physical Connections section. <br> The connected slot number displays beside the channel number. <br> 3. To disconnect the chassis, from the Physical Connections section, drag the connected chassis number to the Unconnected Chassis section. |
| Display the switch configuration for a connected expander chassis | 1. In the System Configuration Tool, click the edge of the expanded chassis. <br> The Chassis Connection dialog box displays. <br> 2. From the Chassis Connection dialog box, click Switch Config.... <br> The Switch Configuration window displays the positioning of the DIP switches located on the back of the Expander Chassis. If the switches are not correctly positioned, the expander communication link fails. Refer to the 8300 Trusted TMR Expander Chassis for more information. |
| Disable an unused slot | 1. From the System Configuration Tool, click an empty slot in the chassis. The Empty Slot dialog box displays. <br> 2. In the Empty Slot dialog box, select the Disable scanning of this slot parameter, and then click OK. |

## See also

Trusted Product Descriptions and Manuals on page 860
Managing Templates on page 578
Configuring I/O Modules for Trusted Applications on page 601

## Configuring I/O Modules for Trusted Applications

After assigning modules to their required slots in the Trusted chassis, all I/O modules for the Trusted system must be configured. When configuring the I/O module, the Module Definition dialog box displays the supported templates to add to the definition. Templates must be created before adding them to the I/O module definition. Scheduled polling, simulation, and a description for the module ay also be enabled.

When enabling scheduled polling, specify the following parameters:

| Parameter | Definition |
| :--- | :--- |
| Poll Interval | The scan rate of the I/O module. Possible values range from o to 20. For example, a <br> value of 5 means the I/O module is only scanned every 5 application scans. |
| Poll Offset | Defines the application scan that scans the I/O module, enabling an even <br> distribution of scans. The offset value must be less than the defined Poll Interval. <br> For example, if two modules have the Poll Interval set to 5, and one has the Poll <br> Offset set to 3 and the other to 4, the first module is polled in the third scan and <br> the other is polled in the fourth scan, so that they do not get polled in the same <br> application scan. |

Tip: All input information that is used for safety-critical functions must have an update rate less than one-half the Process Safety Time.

Enabling simulation allows the system to start up without the primary module installed. Software simulation of the module is invoked if the module is not present. Simulation provides a set of default values for the field inputs corresponding to the default safe-state (logic ' 0 '). If the module is present the system uses the actual module instead of the simulation model. Simulation is useful during system integration as it allows the normal applications to be loaded and tested without a full complement of I/O modules.

When simulating, the Partner Chassis and Partner Slot must be specified for the System to start without the primary module fitted. The secondary module position must always be the adjacent slot to the right of that occupied by the primary module when using the Companion Slot configuration. When starting up a system with a secondary module, the simulation model is first opened (inputs signals all report default safe-state) and the system performs an Active/Standby changeover between the simulated model and the actual secondary module.

The Partner Chassis and Partner Slot parameters are used to identify the position for a unique Companion Slot position for a standby module. The Partner Chassis and Partner Slot parameters should not be specified for a SmartSlot system since more than one module may use the same smart slot. If the same partner chassis/slot is specified for different modules, the MP removes the partnering. Therefore, SmartSlot systems should have Partner Chassis and Partner Slot set to 0 .

[^12]
## To configure an I/O module

1. From the System Configuration Tool, in the required chassis, click the I/O module to configure.


The Module Definition dialog box displays for the specific I/O module.
2. In the Module Definition dialog box, select the required templates in the Supported Templates list and then click Add to Module.

The templates display in the Templates Included with this module list.
3. To remove a template from the I/O module definition, select the template in the Templates Included with this module list and then click Remove.
4. To enable polling, select Enable Scheduled Polling and then enter the required values for Poll Interval and Poll Offset.
5. To enable simulation, select Simulate and then enter the required values for Partner Chassis and Partner Slot.
6. To add a description, type the required information in the User Description text box.
7. To complete the I/O module configuration, click OK.

## Configuring the 8151 Communications Interface Module

## See also

## Managing Templates on page 578

Assigning I/O Modules to the Trusted System on page 600
Access the $\mathbf{8 1 5 1}$ Communications Interface Module Parameters to configure the Communications Interface module.

Tip: Refer to the 8151B Communications Interface module product description for configuration details.

## To access the 8151 Communications Interface module parameters

- From the System Configuration Tool, in the controller chassis, click an existing 8151 Communications Interface module.


The $\mathbf{8 1 5 1}$ Communications Interface Module Parameters dialog box displays.

## See also

## Trusted Communications Interface

## Generating the SYSTEM.INI file

After configuring the required information in the System Configuration Tool, generate the SYSTEM.INI file.

After generating the SYSTEM.INI file, view the current contents of the file in the buffer. Modify the text font and text colors that display in the SYSTEM.INI and associate all *.ini files with the current application.

Also generate a diagram of the Trusted ${ }^{\circledR}$ system from the SYSTEM.INI file. Regenerating the graphic from the INI buffer updates the display of the System Layout displayed in the System Configuration Tool.

Print the SYSTEM.INI text file and diagram.

| Task | Procedure |
| :---: | :---: |
| Generate the SYSTEM.INI file | 1. From the System Configuration Tool, click <br> The Version Control Information Editor displays. <br> 2. In the Version Control Information Editor, enter the Project Title, Author, Date, Time, and if required a Note, and then click OK. <br> The .INI Buffer now matches the diagram displayed in the System Layout. |


| Task | Procedure |
| :---: | :---: |
| Display the contents of the SYSTEM.INI file | 1. From the System Configuration Tool, click 9 . <br> The contents of the SYSTEM.INI display in the ini Buffer Viewer. <br> 2. To associate all *.ini files with the current file: <br> a. In the ini Buffer Viewer, from the Options menu, click Preferences.... <br> b. In the OptionsFrm dialog box, in the General Preferences tab, select Associate '.ini' files with this application and then click OK. <br> 3. To modify the color preferences: <br> a. In the ini Buffer Viewer, from the Options menu, click Preferences.... <br> b. In the OptionsFrm dialog box, in the Colours tab, select the type of text to modify and then click Colour Picker... <br> c. From the Color dialog box, select the required color, then click OK. <br> d. Modify all required colors and then click OK in the OptionFrm dialog box. <br> 4. To save the SYSTEM.INI file, from the File menu, click Save... <br> 5. To load a SYSTEM.INI file, from the File menu, click Load... |
| Generate the SYSTEM.INI file diagram | - From the System Configuration Tool, click The graphic diagram displayed in the System Layout is regenerated. |
| Print the SYSTEM.INI | - To print the text file, from the File menu, click Print Buffer.... <br> - To print the diagram, from the File menu, click Print Graphic.... |

## See also

Example SYSTEM.INI file on page 604
Configuring the SYSTEM.INI File on page 598

## Example SYSTEM.INI file

```
;
; Trusted - System .INI File
;
;
;
[templates\system\sys.inc]
cli='DCL "IMBTO=1000"'
cli='DCL "WDOGTO=1000"'
cli='DCL "BYPASSTO=30000"'
;
[templates\Led\LED.inc]
cli='DCL "0=red flash"'
cli='DCL "1=red"'
cli='DCL "2=off"'
cli='DCL "3=off"'
cli='DCL "4=green"'
cli='DCL "5=green flash"'
cli='DCL "6=red flash"'
cli='DCL "7=red flash"'
cli='DCL "8=red"'
```

```
cli='DCL "9=red"'
cli='DCL "10=red"'
cli='DCL "11=red"'
cli='DCL "12=red"'
cli='DCL "13=red"'
cli='DCL "14=red"'
cli='DCL "15=red"'
;
[templates\threshold\Threshold.inc]
cli='DCL "DEFAULT=
1536,2048,6144,6656,8192,8704,11264,11776"'
;
;
; Trusted - Processor Chassis
;
;
; Trusted TMR Processor
;
[UPDATER]
;
;
; Communications Interface - Coil/Register Offsets
;
; InterGroupDelay - coil/register group delays.
;
InterGroupDelay = 50
;
;
; Block Write Protection Of Modbus Variables.
;
AutoProtect = 0
;
[ISAGRAF_CONFIG]
;
ISA_SLEEP_PERIOD = 32
MAX_SCAN_TIME = 1000
;
[IMB]
```

$\mathrm{SC}=2,2, \mathrm{NIO}$
SC=2, 1, NIO
$\mathrm{SC}=2,13, \mathrm{XPM}$
$\mathrm{SC}=2,14, \mathrm{XPM}$
CC=2,2,1
SC=1,8, PIM
SC=1,7, PIM
$S C=1,2, X I M$
SC=1,1,XIM
$\mathrm{SC}=1,15, \mathrm{MP}$
;
;
; ICS 2000 Configuration
;
[I2K_SYSTEM]
$a=a$
$\mathrm{b}=\mathrm{a}$
$\mathrm{c}=\mathrm{a}$
;
;
[COMMS]
serial1=19200, 8, Odd,1,rs232
serial2=19200, 8, Odd,1,rs485fdmux
serial3=19200,8, Odd,1,rs485fdmux
;
[MODBUS]
;
[IRIG]
Enabled=0
Mode=1
LEDMonitoring=0
;
[CHASSIS]
max_racks $=2$
;
;
[CHASSIS_01_SLOT_01]
module_type $=8311$

```
;
[CHASSIS_01_SLOT_02]
module_type = 8311
;
[CHASSIS_01_SLOT_07]
module_type=8151
serial1=19200,8,Odd,1,rs485fdmux
serial2=19200,8,Odd,1,rs485fdmux
serial3=19200,8,Odd,1,rs485fdmux
serial4=19200,8,Odd,1,rs485fdmux
tcp_ip0=10.10.10.2,255.255.255.0,11.10.9.8
tcp_ip1=9.9.9.2,255.255.255.0,11.10.9.8
;
; Modbus slave settings
;
mb_port_timeout=300
;
[CHASSIS_01_SLOT_08]
module_type=8151
serial1=19200,8,Odd,1,rs485fdmux
serial2=19200,8,Odd,1,rs485fdmux
serial3=19200,8,Odd,1,rs485fdmux
serial4=19200,8,Odd,1,rs485fdmux
tcp_ip0=10.10.10.1,255.255.255.0,11.10.9.8
tcp_ip1=9.9.9.1,255.255.255.0,11.10.9.8
;
; Modbus slave settings
;
mb_port_timeout=300
;
; Trusted - Expander Chassis
;
; Chassis Number 2
;
;
[CHASSIS_02_SLOT_01]
module_type = 8403
description=TMR 24Vdc Digital Input
```

```
simulate= 0
scan_interval = 0
;
cli='DCB'
cli='DCL "[SYSTEM]"'
$[templates\system\sys.inc]
cli='DCL "[LED]"'
$[templates\Led\LED.inc]
cli='DCL "[THRESHOLD]"'
$[templates\threshold\Threshold.inc]
cli='DCE'
;
[CHASSIS_02_SLOT_02]
module_type = 8451
description=TMR 24Vdc Digital Output
simulate= 0
scan_interval = 0
;
cli='DCB'
cli='DCL "[SYSTEM]"'
$[templates\system\sys.inc]
cli='DCL "[LED]"'
$[templates\Led\LED.inc]
cli='DCE'
;
[VCI]
;
;VERSION CONTROL INFORMATION
;The <CRC> variable is calculated on everything
;excluding the CRC line
;
Author = NickO
ProjectTitle = Demonstration File
Date = 18/02/2017
Time = 12:32:01
Note =
```


## See also

## Generating the SYSTEM.INI file on page 603

## Setting the Communications Port

Configuring the communications port enables downloading the SYSTEM.INI file to the TMR Processor. The workstation must be connected to the Trusted ${ }^{\circledR}$ System either via a serial port to the front panel socket of the TMR processor using Maintenance Cable TC-304-01 or via Ethernet to a Communications Module.

## To set the communication port

1. From the System Configuration Tool, in the Communications menu, click Configure Port.
2. In the Com Port Options dialog box, select the Serial or Ethernet tab.
3. Set the required communication parameters, and then click OK.

## See also

Serial Communications on page 609
Ethernet Communications on page 610
Trusted Product Descriptions and Manuals on page 860
Configure serial communications using the Serial tab in the Com Port Options dialog box. The available serial ports display in the Communication Ports drop-down combo box. Leave the remaining settings at their defaults, unless the Trusted ${ }^{\circledR}$ System communications configuration has been changed. Selecting a serial port defines that port as the communication channel, regardless of the settings on the Ethernet tab.


## See also

## Ethernet Communications

## Download the SYSTEM.INI file to the TMR Processor

Configure Ethernet communications using the Ethernet tab in the Com Port Options dialog box. If the Trusted ${ }^{\circledR}$ System has previously been configured for Ethernet communications (using the SYSTEM.INI configuration) then further communications are made quicker by Ethernet. On the Ethernet tab, enter the TCP/IP address assigned to the socket used on the communications module. Leave the port defined as 6000 unless changed in the SYSTEM.INI configuration.


## See also

## Serial Communications on page 609

Download the SYSTEM.INI file to the Trusted TMR Processor to configure the Trusted ${ }^{\circledR}$ system. The SYSTEM.INI file transmits in packets of data.

## Prerequisites

- Configure the communication port.
- Set the enable key switch on the TMR Processor to Maintain.
- Enter design mode.


## To download the SYSTEM.INI file to the TMR Processor

1. From the System Configuration Tool, click Write to TMR Processor or select File > Write to TMR Processor. The TX LED in the Communications in progress window flashes as each packet of data downloads. The RX flashes to indicate the successful download of a packet.
2. After the successful completion of the download, in the Communications in progress window, click OK.
Tip: After downloading the new or modified SYSTEM.INI, either reboot the Trusted ${ }^{\circledR}$ TMR
Processor to implement the file, or stop and then start the application.

## See also

Setting the Communications Port on page 609
Generating the SYSTEM.INI file on page 603
Upload the SYSTEM.INI file from the TMR Processor on page 611

## Upload the SYSTEM.INI file from the TMR Processor

## Prerequisite

- Enter design mode.


## To upload the SYSTEM.INI file from the TMR Processor

1. From the System Configuration Tool, click 䊆 Read from TMR Processor or select File > Read from TMR Processor. The TX LED in the Communications in progress window flashes as each packet of data uploads. The RX flashes to indicate the successful upload of a packet.
2. After the successful completion of the upload, in the Communications in progress window, click OK.
Tip: After successfully downloading of the new or modified SYSTEM.INI, either reboot the Trusted ${ }^{\circledR}$ TMR Processor to implement the file, or stop then start the application.

## See also

Setting the Communications Port on page 609
Generating the SYSTEM.INI file on page 603
Download the SYSTEM.INI file to the TMR Processor on page 610
Use the Dictionary to manage variables and defined words. Create, edit, or delete variables and defined words from the dictionary instances. The grids of the Dictionary include:

- Defined Words grid -- manages the defined words for a project. For example, create the defined word $O K$ to replace the boolean value TRUE in program organization units (POUs).
- Variables grid -- manages the variables for POUs and the device. Each device and POU has its instance of the grid. For the device, the grid displays global variables. For POUs, the grid displays local variables.

Open the dictionary grids from the Application View. The grids each display the properties for the local variables, global variables, or defined words. Open multiple grids simultaneously.

Customize the Dictionary environment by arranging the columns to display and setting the display colors.

## To arrange the columns to display

Important: To retain customized display settings, save the Dictionary instance before closing.

1. To move a column, drag the column header to another location.

When dragging a column header, arrows indicate the current position of the header.
2. To hide a column, right-click a column header, and then click Hide Column.
3. To show a column, right-click any column header, point to Show Column, and then click the required column name.

## See also

Access a Dictionary grid instance on page 612
Defined Words Grid on page 612
Variables Grid on page 615

## Access a Dictionary grid instance

Manage global variables, local variables, or defined words from the Dictionary instances to configure the program organization units (POUs) in a project.

## To access a Dictionary grid instance

1. From the Application View, expand the device node and double-click Global Variables. The Dictionary instance appears and contains the global variables belonging to the device.
2. For the variables of a POU, expand the required POU node and doubleclick Local Variables. The Dictionary instance appears and contains the variables belonging to the POU.
3. For the defined words of a project, double-click Defined Words. The Defined Words grid appears.

## See also

Dictionary on page 611
Defined Words Grid on page 612
Variables Grid on page 615
The Defined Words grid displays global or local defined words. Use the Defined Words grid of the Dictionary to:

- Create defined words
- Edit existing defined words
- Delete defined words
- Sort defined words in the grid
- Filter defined words in the grid


## See also

Create a defined word on page 613
Edit an existing defined word on page 613
Delete a defined word on page 614
Sort defined words on page 614
Access a Dictionary grid instance on page 612

## Create a defined word

## Edit an existing defined word

Create defined words to use in projects. Defined words are identifier names that replace literal expressions, boolean expressions, reserved keywords, or complex ST expressions.

## To create a defined word

1. From the Application View, double-click Defined Words.
2. In the Defined Words grid, define the required properties:

- Name -- Name of the defined word. Limited to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Names cannot be reserved words, defined words, or data types. Names must be unique.
- Equivalent -- String replacing the defined word during compilation. For example, the defined word "PI" is replaced by its equivalent "3.14159". Limited to 255 characters.
Tip: Do not use spaces for the equivalent string. Using a space prevents saving of the new defined word.
- Comment -- Comment in free-format text (up to 60 characters) for the defined word.

3. Press Enter.

## See also

Defined Words Grid on page 612
Delete a defined word on page 614
Modify the name, equivalent value, or comment for an existing defined word to accommodate the requirements of the project. For example, change the value in a literal expression.

## To edit an existing defined word

1. From the Application View, double-click Defined Words.
2. In the Defined Words grid, change the required properties:

- Name -- Name of the defined word. Limited to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Names cannot be reserved words, defined words, or data types. Names must be unique.
- Equivalent -- String replacing the defined word during compilation. For example, the defined word "PI" is replaced by its equivalent "3.14159". Limited to 255 characters.
Tip: Do not use spaces for the equivalent string. Using a space prevents saving of the new defined word.
- Comment -- Comment in free-format text (up to 60 characters) for the defined word.

3. Press Enter.

## See also

Defined Words Grid on page 612
Sort defined words on page 614
Filter defined words on page 615
Delete a defined word on page 614
Delete defined words from the Defined Words grid to no longer use them in program organization units (POUs). Deleting a defined word also deletes its instance in existing POUs.

## To delete a defined word

1. From the Application View, double-click Defined Words.
2. In the Defined Words grid, right-click the defined word, and then click Delete.

## See also

Sort defined words on page 614
Filter defined words on page 615
To locate specific information, sort the defined words in the dictionary grid by sorting columns in ascending or descending order.

## To sort defined words

1. From the Application View, double-click Defined Words.
2. In the Defined Words grid, select a column header. The sort is in ascending order by default.
Tip: An arrow indicating the current order appear in the column header.
3. Select the column header again to toggle between ascending and descending order.

## See also

## Filter defined words on page 615

## Filter defined words

## Variables Grid

Filter defined words in the Defined Words grid to locate specific information. Filtering displays only the defined words containing the specified characters.

## To filter defined words

1. From the Application View, double-click Defined Words.
2. In the filter row at the top of the Defined Words grid, click the cell that contains the content by which to sort.
3. Perform the sort by either:

- Typing the alphabetical and numerical characters by which to sort the defined words in the selected column.
- Selecting a defined word from the drop-down combo box.

Defined words matching the criteria appear in the grid.

## See also

## Sort defined words on page 614

Use the variables grid of the Dictionary to manage the variables for the device or program organization units (POUs). Each device and POU has an instance of the grid. For the device, the grid displays global variables. For POUs, the grid displays local variables. Perform these tasks from the variables grid:

- Create variables
- Edit existing variables
- Drag variables
- Delete variables
- Sort variables in the grid
- Filter variables in the grid

For global or local variables, the properties are the following:

| Column | Description | Possible Values |
| :--- | :--- | :--- |
| Name | Name of the variable | Limited to 32 characters beginning with a letter or single underscore <br> followed by letters, digits, and single underscore characters. Names <br> cannot be reserved words, defined words, or data types. Names must <br> be unique. |
| Logical Value | Available while running online, monitoring, and simulating <br> applications. Displays the value used by code being <br> executed on the Trusted® controller. This cell enables <br> forcing the value of variables. | Values display according to the variable data type <br> Physical ValueAvailable while running online and monitoring <br> applications. Displays the value sent to and received from <br> the drivers. Enables forcing the value of variables. | | Values display according to the variable data type |
| :--- |
| Lock |
| Available while running online, monitoring, and simulating <br> applications. The indication of whether the value of the <br> I/O variable is locked. |
| Yes or No |
| Data Type |

Chapter 3 Working with Trusted Applications

| Column | Description | Possible Values |  |  |
| :---: | :---: | :---: | :---: | :---: |
| String Size | Maximum length for String-type variables | String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string |  |  |
| Wiring | (Read-only) Generated by the I/0 wiring tool indicating the 1/0 channel wired to the variable | Uses the syntax of Directly Represented Variables |  |  |
| Attribute | Read and write access rights of a variable | Possible values are Read or Read/Write. The possible values depend on the direction of the variable. When wiring a variable, the attribute changes to match the direction required by the I/O device. |  |  |
| Direction | For the local variables of a program, indicates the variable is internal. | Possible value is Var |  |  |
|  | For the local variables of a function or function block, indicates whether the variable is internal, an input, or an output. | Possible values are Var, Varlnput, or VarOutput |  |  |
|  | For the global variables of a device, indicates the variable is internal, for use with I/O wiring, or defined for an I/0 channel. | For global variables, the possible value is VarGlobal. Wired variables must have the VarGlobal direction. When adding I/O devices, I/0 channels are assigned the VarDirectlyRepresented direction. |  |  |
| Modbus Address | Modbus address of the variable | Possible Modbus address depends on the data type and extended attributes: |  |  |
|  |  | Modbus Type | Modbus Address | Data Type |
|  |  | Coil | 1-10000 | BOOL |
|  |  | Discrete Input | 10001-20000 | BOOL |
|  |  | Input Register | $30001-39999$ | DINT, REAL |
|  |  | Holding Register | 40001-49999 | DINT, REAL |
| Message True | User-defined text that is used by the SOE Collector in the Trusted Sequence of Events and Process Historian Package when the value of a Boolean variable data type is TRUE. | Text limited to eight Windows-1252 characters excluding: , \% " All leading and trailing spaces are automatically removed. |  |  |
| Message False | User-defined text that is used by the SOE Collector in the Trusted Sequence of Events and Process Historian Package when the value of a Boolean variable data type is FALSE. | Text limited to eight Windows-1252 characters excluding: , \% " All leading and trailing spaces are automatically removed. |  |  |
| Retained | Indication of whether the Trusted controller saves the value of the variable at each cycle. For details on retaining or backing up variables, refer to the SYSTEM operator. | Yes or No |  |  |
| Initial Value | Value held by a variable when the Trusted controller starts the execution of the code | The initial value of a variable. Possible values are the default value, a value assigned by the user when defining the variable, or the value of the retain variable after the Trusted controller has stopped. |  |  |
| Unit | Unit of measure of the logical and physical values | Free format |  |  |
| Comment | User-defined text | Free-format text limited to 60 single-byte characters for variables |  |  |

Customize the Dictionary environment by arranging the columns to display.

## See also

## Create a variable on page 616

Create multiple variables on page 620
Change the properties of a variable on page 617
Delete a variable on page 619
Access a Dictionary grid instance on page 612
Create global and local variables for use in Trusted ${ }^{\circledR}$ projects. Create variables for use in program organization units (POUs) as inputs, outputs, or internal variables. Create variables before defining POUs.

[^13]
## To create a variable

1. From the Application View, double-click the Dictionary instance for the device or POU.
2. In an empty row of the Variables Grid, define the properties for the variable:

- Name
- Data Type
- String Size
- Attribute
- Direction
- Modbus Address
- Message True
- Message False
- Retained
- Initial Value
- Unit
- Comment

3. Press Enter.

## See also

Variables Grid on page 615
Create multiple variables on page 620
Drag a variable on page 618
Delete a variable on page 619

Change the properties of a variable

Change the properties of an existing variable to accommodate the requirements of program organization units (POUs), I/O wiring, and Modbus ${ }^{\circledR}$ communication. For example, change the Direction property for I/O wiring.

Tip: Update each instance of a function or function block after adding, removing, renaming, changing data type, or changing direction of parameters in all POUs using the instance. Identify these POUs with the Cross Reference Browser, then open each POU and reselect the modified instance using the Block Selector.

## To change the properties of a variable

1. From the Application View, double-click the Dictionary instance for the device or POU.
2. In the Variables Grid, change the properties for the variable:

- Name
- Data Type
- String Size
- Attribute
- Direction
- Modbus Address
- Message True
- Message False
- Retained
- Initial Value
- Unit
- Comment


## See also

## Access a Dictionary grid instance on page 612

Variables Grid on page 615
Drag variables from a Dictionary instance to multiple locations within a project. These locations include other Dictionary instances and elements within a language container.

Tip: To help prevent error messages, avoid dragging variables from a:

- global Dictionary instance to a local instance
- local instance to a global Dictionary instance.

Cut and paste or import and export variables instead.
When dragging one or more variables to another Dictionary instance, place the variables anywhere in the grid. When dragging a variable into a language container, place the variable anywhere in the language container. To retain changes made to Dictionary instances and language containers, save the respective instance or program organization unit (POU) before closing.

## To drag a variable

1. From the Application View, access the Dictionary instance containing the variable and the destination for the variable.
2. From the Dictionary instance containing the variable, in the Variables Grid, select the variable by clicking the cell in the left-most column.
The selection indicator $\quad$ appears in the leftmost column.
3. Drag $\downarrow$, placing the variable in the grid or open language container.

The variable appears at the destination.

## See also

Access a Dictionary grid instance on page 612
Variables Grid on page 615
Cut, Copy, and Paste Objects on page 722

## Import and Export Variables Data on page 534

Delete a variable

## Sort variables

## Filter variables

Delete variables from Dictionary instances to no longer use them in program organization units (POUs). Deleting a variable also deletes its instance in existing POUs.

Tip: Update each instance of a function or function block after adding,
removing, renaming, changing data type, or changing direction of parameters in all POUs using the instance. Identify these POUs with the Cross Reference Browser, then open each POU and reselect the modified instance using the Block Selector.

## To delete a variable

1. From the Application View, access the Dictionary instance for the device or program organization unit (POU).
2. Right-click the variable and click Delete.

## See also

Sort variables on page 619
Filter variables on page 619
To locate specific information, sort the variables in the dictionary grid using an ascending or descending order for the individual columns.

## To sort variables

1. From the Application View, double-click the dictionary instance for the device or program organization unit (POU).
2. In the Variables Grid, select a column header. The sort is in ascending order by default.
Tip: An arrow indicating the current order appear in the column header.
3. Select the column header again to toggle between ascending and descending order.

## See also

Variables Grid on page 615
Access a Dictionary grid instance on page 612
Filter variables on page 619
Filter variables in variables grid instances. Filtering creates a view that displays only the variables containing specified characters.

The filter row is the top row of the grid. Filter variables by typing alphabetical and numerical characters in the cells of the filter row or select an option from the drop-down combo box. Matching variables automatically display.

## To filter variables

1. From the Application View, access the Dictionary instance for the device or program organization unit (POU).
2. In the filter row of the Variables Grid, click the required cell, then do one of the following:

- Type the characters to use in the filtering operation
- Select the required defined word from the drop-down combo box


## See also

Variables Grid on page 615
Access a Dictionary grid instance on page 612
Sort variables on page 619

## Create multiple variables

Simultaneously create multiple local or global variables with Quick Declaration. Access Quick Declaration from a dictionary instance.

## To create multiple variables using Quick Declaration

1. In the dictionary instance, right click an empty row, and then click Quick Declaration.
2. Configure these attributes in Quick Declaration.

- Numbering -- The range of values for the variables. The default Digits option is auto. Change the Digits option to alter the quantity of displayed digits.
- Name -- The variable name contains a Prefix and Suffix. The Prefix appears before the number value and can contain letters, digits, and single underscores. The Suffix appears after the number value and can contain letters, digits, and single underscores. Neither can contain two consecutive underscores.
- Data Type -- Displays the variable types. Possible values include elementary IEC 61131-3 types (BOOL, DINT, MESSAGE, REAL, or TIME).
- Direction -- Indicates whether the variable is internal, an input, an output, or for use with I/O wiring. Possible values include Var, VarInput, VarOutput, or VarGlobal. The possible values depend on the selected context.
- String Length -- Defined length only applying to the MESSAGE variable. Accepted values are 1 to 252.

3. View a preview of the variable from the top-right of Quick Declaration, and then click OK.

## See also

Variables Grid on page 615
Create a variable on page 616

## Variable extended attributes

Extended attributes allow the collection of appropriate data from the controller via the TMR Communications Interface module.

- In the SOE tab, generate a time-stamped log of all faults, field trips, output actions etc. recorded by the 8000 Series System. For details, see the Trusted SOE \& Process Historian Software Package, publication ICSTT-RM243.

Tip:

- You do not need to download code to the controller or update the running application if you change only the Message True and Message False variable properties in the global dictionary. When you connect with the controller, a confirmation message that indicates a mismatch appears. Select Yes to continue connecting with the controller.
- Download code to the controller or update the running application when you change only the Message True and Message False variable properties by using the variables data file.
- In the Modbus ${ }^{\circledR}$ tab, control the manner in which data is controlled and interpreted over Modbus communications.


## See also

Select Boolean and Analog variables for SOE collection on page 621
Set extended attributes for Modbus collection on page 622
Trusted Product Descriptions and Manuals on page 860

## Select Boolean and Analog variables for SOE collection

Collect a time-stamped log of all faults, field trips, output actions and more recorded by the 8000 Series System:

- Inherent Sequence of Events (SOE) logging of all Boolean type input states
- Time resolution to 1 ms
- User-defined tag, description, state and display colors
- Text-based log file for import into other programs

Tip: For details, see the Trusted SOE \& Process Historian Software Package
Product Description, publication ICSTT-RM243.

## To select Boolean and Analog variables for SOE collection

1. Create a variable in the Dictionary.
2. Wire the variable through the I/O Wiring tool.
3. In Application View, right-click the dictionary instance, then select Extended Attributes.
4. In Extended Attributes, in the SOE tab, click Add Variable.
5. In the Variable Selector, select the variable to include in the SOE log.
6. Select OK.

## See also

Variable extended attributes on page 621
Create a variable on page 616

Variable Selector on page 40
Wire the channels of an I/O device on page 643
Trusted Product Descriptions and Manuals on page 860

## Set an Intermediate variable for SOE collection without extended variables

An alternate method is available to collect data from the controller without using SOE extended attributes.

Tip: For details, see the Trusted SOE \& Process Historian Software Package, publication ICSTT-RM243.
Important: Do not select a variable already using the SOE extended attributes.

## To set an Intermediate variable for SOE collection without extended variables

1. Open the I/O Wiring tool.
2. Add a SOE board.
3. Double-click an unwired channel.
4. In the Variable Selector, select the variable for the channel.
5. Click OK.

The channel's Name field indicates the wired variable.

## See also

1/O Wiring on page 638
Select Boolean and Analog variables for SOE collection on page 621
Variable extended attributes on page 621
Control the manner in which data is controlled and interpreted over Modbus ${ }^{\circledR}$ communications.

Tip: Extended attributes are only available for variables with a Modbus address. Specify the Modbus address format (Hexadecimal or Decimal) in the CAM Trusted settings from the Tools > Options menu. The System Configuration Tool reads Modbus addresses for variables in Decimal format.

- Write Protect: Inhibits external communications from overwriting a data point. This applies to both Integer/Real and Boolean variables.

To force compatibility of read/write capabilities of coils and registers for the current 8110 Trusted TMR Processor with previous firmware versions, when configuring the processor, in the Updater section, select the Auto Protect Network Variables check box. The compatible read/write capabilities of the coils and registers are the following:

| Address | Element | Number of Elements | Capability |
| :--- | :--- | :--- | :--- |
| $00001-7500$ | Coils | $(7500$ coils $)$ | Read only |
| $07501-10000$ | Coils | (2500 coils) | Read/write |
| $40001-47500$ | Holding registers | (2500 holding registers) | Read only |
| $47501-50000$ | Holding registers | (2500 holding registers) | Read $/$ write |

- Registers as Unsigned Integer: Reads and writes Modbus data as unsigned data. Integer data is normally treated as 32 -bit signed within the application (for signed data, the most significant bit is used to distinguish positive and negative numbers).
- Use Two Registers (32-bits): Uses two contiguous Modbus registers to transfer all 32 bits. By default, only the lower 16 bits of an analog data point are transferred to and from the Modbus communications.
- Least Significant Word First: Reverses the words on the Modbus addresses. If Use Two Registers is selected, by default the most significant 16 -bit word is addressed first on Modbus communications.

For details on extended attributes for Modbus variables, refer to the Product Description PD-8082, Trusted Toolset Suite.

## To set extended attributes for Modbus collection

1. In the Dictionary, select the variable, then set Modbus Address.

Tip: The T8151 Communications Interface does not support duplicate Modbus addresses.
2. Click OK.
3. In the Application View, right-click the dictionary instance and click Extended Attributes.
4. In the Appli.vxa window, select the Modbus tab, then select one or more of the extended attributes for the variable.
Tip: $\quad$ The Least Significant Word First option is grayed out unless the Use Two Registers option is selected.

## See also

Variable extended attributes on page 621
Change the properties of a variable on page 617
Access a Dictionary grid instance on page 612

## Cross Reference Browser

The Cross Reference Browser displays the variables, blocks, and defined words in a Trusted ${ }^{\circledR}$ project. The displayed cross reference information varies depending on the selected view. Access the individual references of a specific variable, block, or defined word.

To locate specific information, search or filter the cross reference data displayed in the Elements pane. The View combo box enables switching between these views:

- Variable Cross References
- Block Cross References


## - Defined Word Cross References




| Item | Name | Description |
| :---: | :---: | :---: |
| 1 | Elements pane | Displays the quantity of cross references for variables, blocks, or defined words. |
| 2 | References pane | Displays the associated cross references for the element selected in the Elements pane. <br> Double-click a cross reference to view the instance in the language container or to view the $1 / 0$ channel in the $1 / 0$ Wiring. |
| 1 | View | Changes the view to: <br> - Variables <br> - Blocks <br> - Defined Words |
| (2) | Cross reference data status | Updates and indicates the status of cross reference data: <br> Update cross reference data <br> Cross reference data is up-to-date <br> Cross reference date is out-of-date |
| (3) | Search Keyword box | Searches for an element and displays the number of instances of the element. Results appear highlighted in the Elements pane. |
| (4) | Previous/Next | Navigates to the previous and next instance of search results displayed in the Elements pane. |
| (5) | Filter | Displays the filter row for each column. Limit the listed elements by filtering a column using alphanumerical characters. |
| 6 | Expand All/Collapse All | When displaying cross references of blocks in the tree view, expands and collapses the displayed information. |
| 7 | Previous Reference/Next Reference | Selects the previous or next cross reference in the References pane. The instance of the cross reference appears in the language container or the I/O Wiring. |
| 8 | Show List/Show Tree | Toggles between showing cross references in the list or tree in the References pane. |


| Task | Procedure |
| :--- | :--- |
| Display the Cross Reference Browser | $\bullet$ From the View menu, click Cross Reference Browser (or press Ctrl $+\mathrm{W}, \mathrm{CtrI}+\mathrm{C}$ ). |


| Task | Procedure |
| :--- | :--- |
| Search through the Elements pane | When performing a search operation, the matching text is highlighted in yellow. Also press Ctrl+F to access the search field. <br> 1. In the search field, enter the text you need to find in the Elements pane. The matching text is highlighted as each letter is <br> typed. The search field displays the number of located instances (2 of 12). |
| 2. Click to select the next instance or to select the previous instance. The cross references associated to the |  |
| selected element are displayed in the References pane. |  |

## See also

Update the cross reference data on page 625
View cross references of a variable on page 625
View cross references of a block on page 627
View cross references of a defined word on page 629

# Update the cross reference data 

Update the Cross Reference Browser to display current cross references in the project.

## To update cross reference data

1. In the Cross Reference Browser, click Update Cross Reference Data. The cross references update.
Tip: Pressing Ctrl+B updates the cross references.
2. To view cross reference data for different elements, in the View box, select an option:

- Variables
- Blocks
- Defined Words


## See also

Cross Reference Browser on page 623
View cross references of a variable

Update the Cross Reference Browser to display current cross references in the project.

The properties of variable cross references appear in the Elements pane of the Cross Reference Browser:

| Property | Description |
| :--- | :--- |
| Name | Name of the variable. |
| Count | Total number of cross references for a variable in the program organization units (POUs) and in <br> the I/O Wiring of a project. Accepted values are 0 or more. |
| Scope | Scope of a declaration. Accepted values are global or local to one POU. |


| Property | Description |
| :--- | :--- |
| Alias | A user-defined name for use with a Ladder Diagram and a Functional Block Diagram. |
| Data Type | Data type of the variable. |
| Wiring | The directly represented variable (DRV) identifier. |
| Project | Projector library using the variable. |
| Device | The name of the device using the variable. |
| Resource | The name of the device using the variable. |
| Comment | Free-format text describing the variable. |

## To view cross references of a variable

1. In the Cross Reference Browser, select the Variables view, and then click ${ }^{\text {B }}$ Update Cross Reference Data. The cross references update.
Tip: While the focus remains on the Cross Reference Browser, the status bar displays the time and date of the most recent update.
2. In the Elements pane, select the variable. The cross references for the selected variable appear in the References pane.
3. Right-click the cross reference and click Go to Reference. The language container or the I/O Wiring displays the cross reference of the variable.

## See also

Change the view of variable cross references on page 626
Update the cross reference data on page 625
View cross references of a block on page 627
View cross references of a defined word on page 629
Change the view of variable cross references

The References pane of the Cross Reference Browser displays additional information for variable cross references graphically in the Tree view or textually in the List view.

| Item | POU Description | I/O Wiring Description |
| :--- | :--- | :--- |
| Read action | The variable instance of the cross reference reads from another <br> element. | The variable reads from the channel. |
| Usage to action | The variable instance of the cross reference writes to another <br> element. | In List view, the element affected by the variable instance of the <br> cross reference such as a variable, block input, or block output. |
| Location List view, the I/O Wiring. |  |  |

## To change the view of variable cross references

1. In the Cross Reference Browser, display the updated cross references for variables.
2. To view the cross references graphically in the References pane, click Show Tree.
3. To view the cross references textually in the References pane, click Show List.

Tips: - Tree view displays the common path and number of affected elements above the Reference pane. Tree view includes the action, variable instance of the cross reference, location, and affected element within the pane.

- List view displays the variable instance of the cross reference above the Reference pane and indicates the action, usage including the affected element, and location within the pane.


## See also

View cross references of a variable on page 625
Update the cross reference data on page 625

View cross references of a block

Update the Cross Reference Browser to display current cross references in the project.

The Blocks view of the Cross Reference Browser displays cross reference data for functions, function blocks, and operators.

| Property | Description |
| :--- | :--- |
| Name | Name of the function, function block, or operator. |
| Count | Total number of cross references for a block in the program organization units (POUs) of a <br> project. Accepted values are 0 or more. |
| Category | Type of function, function block, or operator. Accepted values are standard or user-defined. |
| Language | Programming language of the function, function block, or operator. |
| Alias | A user-defined name for use with Ladder Diagram program organization units (POUs). |
| Project | Project using the block |
| Device | The name of the device using the block |
| Resource | The name of the device using the block |
| Comment | Free-format text describing the block |

## To view cross references of a block

1. In the Cross Reference Browser, select the Blocks view, and then click娄 Update Cross Reference Data.
The cross references update.
2. In the Elements pane, select the block. The cross references for the selected block appear in the References pane.
Tip: From the Elements pane, expand individual blocks to view the inputs and outputs. Select user-defined functions and function blocks to view the cross reference data for each parameter.
3. Right-click the cross reference and click Go to Reference. The language container displays the cross reference of the block.

## See also

Change the view of block cross references on page 628
List view for block cross references on page 628
Tree view for block cross references on page 628
Update the cross reference data on page 625

## Change the view of block cross references

## List view for block cross references

## See also

List view for block cross references on page 628
Tree view for block cross references on page 628
The References pane displays additional information for the cross references of a block textually from the List view.

| Column | Description |
| :--- | :--- |
| Action | The action of the cross reference. Accepted value is Called by. |
| Usage | The element affected by the block instance of the cross reference. |
| Location | Location of the cross reference in the grid. |

## See also

Change the view of block cross references on page 628
Tree view for block cross references on page 628

Tree view for block cross references

The References pane displays additional information for the cross references of a block graphically from the Tree view.


| Item | Name | Description |
| :--- | :--- | :--- |
| $\mathbf{1}$ | Common path | The common path of the block selected in the Elements pane and the <br> number of cross references. |
| $\mathbf{2}$ | Program organization unit <br> (POU) | The name of the POU using the block and the quantity of cross <br> references. |
| $\mathbf{3}$ | Called by | The defined word is used by the POU. |
| $\mathbf{4}$ | Block | Name of the function, function block, or operator. |
| $\mathbf{5}$ | Location | Location of the cross reference in the language editor grid. |

## See also

Change the view of block cross references on page 628
List view for block cross references on page 628

View cross references of a defined word

Update the Cross Reference Browser to display current cross references in the project.

The properties of defined word cross references appear in the Elements pane of the Cross Reference Browser:

| Property | Description |
| :--- | :--- |
| Name | Name of the defined word. |
| Count | Total number of cross references existing for a defined word in the project. |
| Equivalent | String replacing the defined word during compilation. |
| Scope | Project using the defined word. |
| Comment | Free-format text |

## To view cross references of a defined word

1. In the Cross Reference Browser, select the Defined Words view, and then click Update Cross Reference Data. The cross references update.
2. In the Elements pane, select the defined word. Cross references for the selected defined word appear in the References pane.
3. Right-click the cross reference and click Go to Reference. The language container displays the cross reference of the defined word.

## See also

Change the view of defined word cross references on page 630
List view for defined word cross references on page 630
Tree view for defined word cross references on page 630
Update the cross reference data on page 625

## Change the view of defined word cross references

## To change the view of defined word cross references

1. In the Cross Reference Browser, display the updated cross references for defined words.
2. To view the cross references graphically in the References pane, click Show Tree.
3. To view the cross references textually in the References pane, click Show List.

## See also

List view for defined word cross references on page 630
Tree view for defined word cross references on page 630
The References pane displays additional information for the cross references of a defined word textually from the List view.

| Column | Description |
| :--- | :--- |
| Action | The action of the cross reference. Possible value is Used by. |
| Usage | The element affected by the defined word instance of the cross reference. |
| Location | Location of the cross reference in the grid. |

## See also

Change the view of defined word cross references on page 630
Tree view for defined word cross references on page 630

## Tree view for defined word cross references

The References pane displays additional information for the cross references of a defined word graphically from the Tree view.


| Item | Name | Description |
| :--- | :--- | :--- |
| $\mathbf{1}$ | Common path | The common path of the defined word selected in the Elements pane <br> and the number of cross references. |
| $\mathbf{2}$ | Program organization unit <br> (POU) | The name of the POU using the defined word and the quantity of cross <br> references. |
| $\mathbf{3}$ | Used by | The defined word is used by the POU. |
| $\mathbf{4}$ | Defined word | Name of the defined word. |
| $\mathbf{5}$ | Location | Location of the cross reference in the language editor grid. |
| $\mathbf{6}$ | Element | Element affected by the cross reference. |

## See also

Change the view of defined word cross references on page 630
List view for defined word cross references on page 630
When creating Trusted ${ }^{\circledR}$ projects, use various validators to verify the application program code generated by the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software. The validators process various database files produced by the AADvance-Trusted SIS Workstation software to identify dependencies between programs within an application and the differences between versions of an application. For Trusted applications, use the following four validators:

- Cross Reference Checker
- TIC Dependency Checker
- TIC Difference Checker
- TIC Version Checker

When verifying the generated application code, the following files are used by the validators:

- APPLI.HIS, text file containing the history of the application
- appli.tst, file containing the symbolic information for variables and programs
- appli.X6M, file containing the generated target code. Set the Enable ISA68M property to True.
- The following table displays the application files verified by each validator:

| Validator | APPLI.HIS | appli.tst | appli.X6M |
| :--- | :--- | :--- | :--- |
| Cross Reference Checker | X |  |  |
| TIC Dependency Checker | X | $X$ | $X$ |
| TIC Difference Checker |  | $X$ | $X$ |
| TIC Version Checker |  |  | $X$ |

## See also

## Cross Reference Checker on page 632

TIC Dependency Checker on page 633
TIC Difference Checker on page 634
TIC Version Checker on page 636
Errors and Warnings on page 637

## Cross Reference Checker

To access the Cross Reference Checker
From the View menu, point to Validators, and then click Cross Reference Checker.

The Cross Reference Checker validates the cross reference file. This file is generated when updating the cross reference data displayed in the Cross Reference Browser. The validator processes the file and graphically displays variable usage and program organization unit (POU) dependencies. Both shared and unshared variables and their associated POUs display in the tree hierarchy.

The Cross Reference Checker displays:

- POUs using variables
- The number of used variables
- The number of unused variables
- POUs with shared variables
- Shared variables
- Lines processed

POUs sharing one or more variables are dependent on each other. Modifications to one POU can affect the behavior of a dependent POU. For SIL3 safety applications, verify POU dependencies to ensure the target integrity of the application.

For a POU, the Cross Reference Checker identifies the POUs calling the POU and the POUs called by the POU, and shows the POUs that need verification. The validator also identifies the variable type and the quantity of POUs using a variable.

The Cross Reference Checker is available from the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software or as a stand-alone application. Update the cross reference data from the Cross Reference Browser to use the Cross Reference Checker.

When viewing statistical information, display information for all POUs or only specified POUs or variables.

When viewing the graphical display of POU and variable dependencies, the icons in the tree hierarchy represent:

|  | POU with no shared variables. Expand to display the non-shared variables. |
| :--- | :--- |
|  | POU with shared variables. Expand to display both shared and unshared variables. |
|  | Unshared variable. Expand to display the associated POU name. |
|  | Shared variable. Expand to display the names of all POUs sharing the variable. |

Perform these tasks with the Cross Reference Checker:

| Task | Procedure |
| :---: | :---: |
| Validate the cross references data | 1. From the View menu, click Cross Reference Browser. <br> 2. In the Cross Reference Browser, click (or press Ctrl+B). <br> 3. The cross references are updated and the Variables view displays. <br> 4. From the View menu, point to Validators, and then click Cross Reference Checker. <br> 5. From the Cross Reference Checker dialog box, click Browse to locate the required Cross Reference File (*.XRF). <br> 6. Select the required POUs and variables to validate in the All Programs \& Variables Found and Related Items sections. POU dependencies display. To ensure the integrity of the safety application, verify all dependent POUs. |
| Expand or collapse the nodes in the POU dependency tree | - To expand the nodes in the POU dependency tree, click $\boxplus$. <br> - To collapse the nodes in the POU dependency tree, click |
| Print the validator results | Set the default printer to use the A4 paper format. <br> - From the File menu, click Print. |

## See also

## Validators on page 631

TIC Dependency Checker on page 633

## Errors and Warnings on page 637

## TIC Dependency Checker

To access the TIC Dependency Checker
From the View menu, point to Validators, and then click TIC Dependency Checker.

The TIC Dependency Checker validates variable usage and program organization unit (POU) dependencies, and specifies POUs that need verification. Unlike the Cross Reference Checker that also performs this validation, the TIC Dependency Checker generates the dependency data from the downloadable TIC code and application symbols database files generated after compilation.

The TIC Dependency Checker displays these statistics:

- Total number of POUs
- POUs using variables
- Total number of variables
- POUs with shared variables
- Shared variables
- Total TIC codes

Perform an additional consistency check by comparing the displayed data using the Cross Reference Checker and the TIC Dependency Checker.

For a POU, the TIC Dependency Checker identifies the POUs calling the POU and the POUs called by the POU, and shows the POUs that need verification. The validator also identifies the variable type and the quantity of POUs using a variable.

The TIC Dependency Checker is available from the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software or as a stand-alone application. Successfully build the project to use the TIC Dependency Checker.

When viewing the statistical information, display information for all POUs or only specified POUs or variables.

When viewing the graphical display of POU and variable dependencies, the icons in the tree hierarchy represent:

|  | POU having no shared variables. When expanded, displays the non-shared variables. |
| :--- | :--- |
|  | POU having shared variables. When expanded, displays both shared and unshared variables. |
|  | Unshared variable. When expanded, displays the associated POU name. |
|  | Shared variable. When expanded, displays the names of all POUs sharing the variable. |

Perform these tasks with the TIC Dependency Checker:

| Task | Procedure |
| :--- | :--- |
| Validate the POU dependencies | 1. From the Build menu, click Build All. <br> 2. From the View menu, point to Validators, and then click TIC Dependency Checker. <br> 3. From the TIC Dependency Checker dialog box, click Browse to locate the required Application Directory. <br> 4. Select the required programs and variables to validate in the All Programs \& Variables Found and Related Items <br> sections. <br> POU dependencies display. Verify all dependent POUs to ensure the integrity of the safety application. |
| Expand or collapse the nodes in the POU <br> dependency tree | - To expand the nodes in the POU dependency tree, click- To collapse the nodes in the POU dependency tree, click <br> Print the validator resultsSet the default printer to use the A4 paper format. <br> - From the File menu, click Print. |

## See also

Validators on page 631
Cross Reference Checker on page 632
Errors and Warnings on page 637

## TIC Difference Checker

To access the TIC Difference Checker
From the View menu, point to Validators, and then click TIC Difference Checker.

The TIC Difference Checker compares the compiled TIC code of two different project databases. This validator identifies differences between two versions of the same application.

The TIC Difference Checker displays the application details specifying the matching POUs with identical TIC code and unmatching POUs with different TIC code. These conditions cause unmatching POUs:

- Modifications made by a user
- Unexpected changes caused by the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software

To compare versions of an application, create a copy of the Trusted ${ }^{\circledR}$ project before modifying the code, then modify the original project.

For a POU, the TIC Difference Checker identifies the POUs calling the POU and the POUs called by the POU, and shows the POUs that need verification. The validator displays the dependency tree of matching POUs or of a specific application.

The TIC Difference Checker is available from the AADvance-Trusted SIS Workstation software or as a stand-alone application. Successfully build modified and copied projects to use the TIC Difference Checker.

The TIC Difference Checker displays these application details:

| Name | The name of the POU |
| :--- | :--- |
| Version | The version of the project |
| Generation Date | The compilation date of the application code |
| Size (bytes) | The size, in bytes, of the application |
| Symbols CRC | Cyclic redundancy checking of the symbols database |
| Total Programs | The number of POUs in each project |
| Matched Programs | The number of POUs that are identical in both applications |
| Unmatched Programs | The number of POUs differing between both applications |


| Icon | Description |
| :--- | :--- |
|  | POU having no shared variables. When expanded, displays the non-shared variables. |
|  | POU having shared variables. When expanded, displays both shared and unshared variables. |
|  | Unshared variable. When expanded, displays the associated POU name. |
|  | Shared variable. When expanded, displays the names of all POUs sharing the variable. |

Perform these tasks with the TIC Difference Checker:

| Task | Procedure |
| :---: | :---: |
| Compare application TIC code with the TIC Difference Checker | To compare versions of an application, create a copy of the Trusted project before modifying the code, then modify the original project. <br> 1. For the required application, from the Build menu, click Build All. <br> 2. Save and then create a copy of the project. <br> 3. Perform the required modifications to the project, then from the Build menu, click Build All. <br> 4. From the View menu, point to Validators, and then click TIC Difference Checker. <br> 5. From the TIC Difference Checker dialog box, click Browse to locate the Application Directories for the applications to compare. <br> 6. From the drop-down combo box, select one of these to view the POU tree hierarchy: <br> - Matched programs <br> - Programs - Application 1 <br> - Programs - Application 2 <br> The TIC Difference Checker displays the application details of the compared projects. Ensure all unmatching programs are due to intended modifications. |
| Expand or collapse the nodes in the POU dependency tree | - To expand the nodes in the program dependency tree, click $\boxplus$ <br> - To collapse the nodes in the program dependency tree, click |
| Print the validator results | Set the default printer to use the A4 paper format. <br> - From the File menu, click Print. |

## See also

Validators on page 631
Errors and Warnings on page 637

## TIC Version Checker

## To access the TIC Version Checker

From the View menu, point to Validators, and then click TIC Version Checker.
The TIC Version Checker compares the compiled TIC code generated by the AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software with the application downloaded onto the Trusted processor. The TIC Version Checker validates that the running application and the compiled TIC code are identical by displaying any differences between the TIC code of the applications.

These conditions cause differences between the Symbols CRC, Application CRC, and Total Size of an application:

- Modifications made by a user
- Unexpected changes caused by the AADvance-Trusted SIS Workstation software
- Recompiling the project, for the Application CRC only

To compare versions of an application, create a copy of the project running on the processor. The TIC Version Checker compares the copy of the project with the running application. The TIC Version Checker displays differences caused by modifying the project application.

The TIC Version Checker is available from the AADvance-Trusted SIS Workstation software or as a stand-alone application.

The TIC Version Checker displays these details:

| Program Name | The name of the project or application. |
| :--- | :--- |
| Version | The version of the application extracted from the downloaded database file. |
| Date | The last compilation date of the database file. |
| Time | The last compilation time of the database file. |
| Symbols CRC | Cyclic redundancy checking of the application symbols. Different applications may <br> have matching Symbols CRC values if both applications contain the same symbols. |
| Total Size | Size of the application in bytes. |
| Application CRC | Cyclic redundancy checking of the downloaded TIC code. Recompiling a project <br> changes the value of the Application CRC even when the application not modified. |

Perform these tasks with the TIC Version Checker:

| Task | Procedure |
| :---: | :---: |
| Compare application TIC code with the TIC Version Checker | To compare versions of an application, create a copy of the project running on the processor. The TIC Version Checker compares the project copy with the running application. The TIC Version Checker displays differences caused by modifying the project application. <br> 1. From the TIC Version Checker, in the Communications menu, click Configure Port. <br> 2. In the Communication dialog box, click Setup. <br> 3. In the Serial link parameters dialog box, set the required values for these properties, and then click Ok. <br> - Baudrate <br> - Parity <br> - Format <br> - Flow Control <br> 1. In the Communication dialog box, enter the required values for these properties, and then click OK. <br> - Target Slave Number <br> - Communication port <br> - Timeout <br> - Retries <br> 1. In the File menu, click Load Project, then from the Open dialog box, browse for the required APPLI.X6M file, and then click Open. <br> 2. To compare the project running application, from the File menu, click Compare Project. |
| Print the validator results | Set the default printer to use the A4 paper format. <br> - From the File menu, click Print. |

## See also

## Validators on page 631

## Errors and Warnings on page 637

## Errors and Warnings

When using one of the validators, one of the following detected errors or warnings may display:

| Warning or Error Message | Description |
| :--- | :--- |
| Not all dependency information has been <br> examined | Displays if an attempt is made to exit the program, or print a <br> dependency report, before all the dependency information is viewed. <br> The dependency information is considered viewed when all the top <br> level program nodes in the view tree are expanded. These can be <br> expanded individually, or all at once by clicking 'Expand All.' <br> Dependency information is still considered viewed if the nodes are <br> collapsed again. |
| Abort, name too long! | Displays if a variable or program name read from the application <br> symbols file, or the cross reference, is too long. The maximum <br> identifier length supported by IEC1131 TooLSET version 3.23 is 16. |
| Aborted, unknown block type header | Displays by Validator \#1 if an unknown header type is encounterd <br> while reading through the cross reference file. Verify the correct file <br> is selected, or try re-generating it. |
| Aborting, input file is not a valid cross <br> reference file | Displays by Validator \#1 if a syntax or format error is encountered <br> while reading through the cross reference file. Verify the correct file <br> is selected, or try re-generating it. |
| Aborting, input file is not a valid symbols <br> file | Displays if a Validator encounters a syntax or format error while <br> reading the application symbols file. Try re-compiling the application <br> to re-generate the symbols file. |
| Applications directory: <name> does not <br> exist | Displays when a Validator cannot locate the specified Application <br> Directory. |
| Cannot find application symbols file | Displays if a Validator cannot locate the application symbols file for <br> an application. Verify the application compiled successfully. |
| Appli.tst for application <name> |  |


| Warning or Error Message | Description |
| :--- | :--- |
| Cannot find TIC code file Appli.x6m for <br> application <name> | Displays if a Validator cannot locate the download data base file for <br> the application. Verify the application compiled successfully and the <br> compiler options are set to generate code for a Motorola target <br> (ISA68M). |
| Cannot produce report, printer fault | Displays if a printer fault occurs while a Validator is attempting to <br> output a report to the printer. |
| Download database not in Motorola | Displays if the applications download database file is not the correct <br> format. Ensure the compiler options are set to generate code for a <br> format |
| Motorola target (ISA68M) and re-compile the application. |  |

## See also

Cross Reference Checker on page 632
TIC Dependency Checker on page 633
TIC Difference Checker on page 634
TIC Version Checker on page 636

## I/O Wiring

## To access I/O Wiring

From the Application View, right-click a device, and then click I/O Wiring.
Configure settings in I/O Wiring to connect project variables to channels of I/O boards or complex equipment existing on a target system. To set up I/O wiring, add I/O devices to a rack and then wire variables to the I/O channels of the devices.

An I/O device represents an I/O board or complex equipment on a rack list. Individual I/O boards and single pieces of equipment can have up to 128 I/O channels. An I/O device has channels with the same data type and direction.

I/O channels represent hardware input or output points. Connect an I/O variable to a channel for use in program organization units (POUs). I/O variable values can be locked, unlocked, and forced as needed during program debugging.

The I/O Wiring view has two sections:

- A rack list, displays defined complex equipment and I/O boards in the slots. An order number identifies each slot. Expanding the equipment accesses information and single devices.
- A channel variables list, enables the association of channels with variables. This list displays the name of all variables. When online, the channel variables list also displays the logical value, physical value, and lock status of all variables.

Apply conversion tables to analog input or output variables to create proportional relationships between electrical values (read on input sensor or sent to the output device) and physical values used in application programming.

The following product families of I/O boards and complex equipment are available for use in a project. Simulation does not support some I/O boards and complex equipment.

Tip: - To simulate I/O boards and complex equipment not supported for simulation, toggle the I/O device to the virtual attribute.

- If you change the variable wiring, the variable will be removed from the SOE collection.

| Product Family | Supported for Simulation |
| :---: | :---: |
| Process Historian | Yes |
| Regent | All Regent equipment except: <br> - dx7417f <br> - dx7461a <br> - dx7462a <br> - m7491_i <br> - m7491_0 <br> - s7420_pp <br> - sx7402 <br> - sx746la <br> - sx7462a <br> - ttmrp <br> - tx7402 <br> - tx7402_p |
| SOE | Yes |
| Trusted 8000 Series | Yes |


| Product Family | Supported for Simulation |
| :---: | :---: |
| Trusted CS300 | All Trusted CS300 equipment except: <br> - b8162 <br> - bic <br> - dmx_m <br> - dmx_o <br> - pi616_5 <br> - pi619_9 <br> - pi626 <br> - pi627 <br> - pi627_fs <br> - pi632 <br> - pi641 <br> - pi716 <br> - pi717 <br> - pi726 <br> - pi727 <br> - pi727_fs <br> - pi732 <br> - pi741 |
| Trusted SC300E | All Trusted SC300E equipment except: <br> - tgprot_C <br> - tgprot_s |

## Use I/O Wiring to:

- Add I/O devices to a rack list
- Toggle an I/O device between real and virtual
- Access the simple devices of complex equipment
- Display information for I/O devices
- Delete I/O devices from a rack list
- Wire the channels of an I/O device
- Unwire the channels of an I/O device
- Lock and unlock an I/O variable
- Force the value of an I/O variable
- Apply a conversion table to analog variables
- Edit the points of an existing conversion table
- Delete a conversion table
- Access the Cross Reference Browser


## See also

Add I/O devices to a rack list on page 640
Wire the channels of an I/O device on page 643
1/O wiring keyboard shortcuts on page 647
Toggle the real/virtual attribute of an I/O device on page 641

## Add I/O boards and complex equipment to a rack list

Add I/O devices to a project when developing an application. Add individual simple and complex I/O devices to the rack list. The rack list holds a maximum of 255 simple devices.

Insert I/O devices from the Device Selector. Each device is assigned a device order number ranging from 0 to 254 and has a defined number of channels.

Modify the order number to use any empty slot．Include a comment．Expand individual devices to access their properties．

I／O devices from a rack list contain these types：

| 1／0 Device Type |  | Description |
| :---: | :---: | :---: |
| 䢕咸 0：xboo＿io | $\approx$ | Real complex equipment |
| 䢕成 0：xboo＿io | v | Virtual complex equipment （indicated by the flag） |
|  | $\approx$ | Real I／O board |
| 140n 1：xbo8 | $\approx$ | Virtual I／0 board（indicated by the flag） |

## To add I／O devices to a rack list

1．From the Application View，right－click the device，and then click I／O Wiring．

2．On the I／O Wiring toolbar，click to add complex equipment and I／O boards to the rack list．
3．In the Device Selector，select an I／O device from the list of available devices．

The device order number and number of channels is defined for the I／O device in the library．

## Toggle the real／virtual attribute of an I／O device

## See also

Wire the channels of an I／O device on page 643
Access simple devices of complex equipment on page 642
Display I／O device information on page 642
1／O Wiring on page 638
While running online，I／O devices set to real directly link I／O variables to the corresponding I／O device channels．Input or output operations in the programs correspond to the physical input or output conditions of the actual I／O device channels．I／O devices set to virtual read or update corresponding I／O device channels to simulate I／O processing．

Toggle between the real and virtual attribute for a selected I／O device．Virtual I／O devices display with a flag．

## To toggle the real／virtual attribute for an I／O device

1．From the rack list，select the I／O deviceto change the attribute．
2．From the I／O Wiring toolbar，click

## See also

## I/O Wiring on page 638

## Access simple devices of complex equipment

## Display I/O device information

## To display I/O device information

1. From the rack list, expand the required I/O device by clicking
2. Click Info to view information about the device.

## See also

1/O Wiring on page 638
When deleting an I/O device, all variables are unwired from the device. Delete an I/O device from a project when removing a physical board.

## To delete an I/O device

1. From the rack list, select the I/O device.
2. From the I/O Wiring toolbar, click $\qquad$

## See also

1/O Wiring on page 638

Wire the channels of an I/O device

Wire variables to channels of an I/O device in the channel variables list. In this list, the displayed variable names are their direct representations.I/O channels represent hardware input or output points. An I/O variable is connected to a channel for use in program organization units (POUs). When debugging, lock, unlock, and force the values of I/O variables.

Use direct variable representation (\%IX1.1) to access I/O values when I/O channels have no wiring.

Tip: In the Dictionary, a directly represented variable with the same name represents an unwired channel. Wiring the channel removes its Dictionary instance.

## To wire the channels of an I/O device

1. From the Application View, right-click the device, and then click I/O Wiring.
2. From the rack list, click the I/O device.

Tip: For complex equipment, expand the equipment to access the simple devices, and then click Devices.
3. In the Channel Variables list, double-click the channel to wire.
4. From the Variable Selector, select the variable, and then click OK.

The Name column indicates the wired variable's direct representation. For Boolean channels, the default value operations are direct and no conversion.
5. (optional) To set conversion tables for analog I/O channels, select the wired channel in the list, then choose the conversion table from the Conversion Function drop-down list.

## See also

Access simple devices of complex equipment on page 642
Add I/O devices to a rack list on page 640
Unwire the channels of an I/O device on page 643
Apply a conversion table to analog variables on page 645
I/O Wiring on page 638
Unwire the channels of an I/O device to use the directly represented variable rather than a wired variable. Unwire all channels or individual channels of an I/O device.

## To unwire channels of an I/O device

1. Access the I/O Wiring for the device.
2. Do one of these:

- To unwire all wired channels for the I/O device, from the I/O Wiring toolbar, click

- To unwire individual channels of the I/O device, right-click the channel, and then select Unwire selected channels.
Tip: To unwire multiple channels, hold the Ctrl button while selecting each required channel.


## See also

Wire the channels of an I/O device on page 643
I/O Wiring on page 638

## Lock and unlock an I/O variable

## To lock and unlock an I/O variable

1. Access the I/O Wiring for the device and select the channel to lock or unlock.
2. To lock or unlock the variable, right-click the variable, and then select or clear Toggle lock on selected channels.

## See also

1/O Wiring on page 638

## Force the value of an I/O variable

When debugging, choose to lock, unlock, and force the values of I/O variables.

Tip: To lock or unlock multiple channels, hold Ctrl while selecting each required
Tip: $\quad$ lo lock or unlock multiple channels, hold Ctrl while selecting each required
channel. .

When debugging, force the values of locked I/O variables from the I/O wiring, the global variables grid, or a spy list. The direct representation definition for the I/O wiring determines the variable direction.

## To force the value of an I/0 variable

1. To force the value of an I/O variable from the I/O wiring:
a. Right-click the device in the Application View, then select I/O Wiring.
b. Select the simple I/O device in the rack list. Expand complex I/O devices to access the simple I/O devices.
c. Double-click the the channel variable from the channel variables list.
d. Lock the variable and enter the value in the Write dialog box:

- For an input variable, enter the logical value, then select Write.
- For an output variable, enter the physical value, then select Write.

2. To force the value of an I/O variable from the global variables grid:
a. Right-click Global Variables in the Application View, then select Open.
b. Double-click the variable name in the variables grid.
c. Lock the variable and enter the value to force in the Write dialog box:

- For an input variable, enter the logical value, then select Write.
- For an output variable, enter the physical value, then select Write.

3. To force the value of an I/O variable from a spy list:
a. From the View menu, select Watch, then the spy list.
b. Double-click the leftmost cell for the variable in the spy list.
c. Lock the variable and enter the value in the Write dialog box:

- For an input variable, enter the logical value, then select Write.
- For an output variable, enter the physical value, then select Write.


## See also

Lock and unlock an I/O variable on page 644
A conversion table is a set of points defining an analog conversion. Apply a conversion table to an analog input or output variable to create a proportional relationship between electrical values (read on input sensor or sent to the output device) and physical values used in application programming.

Tip: Trusted ${ }^{\circledR}$ projects can have a maximum of 127 conversion tables.

A conversion table filters the values of any input or output analog variable of a project. Attach a conversion table to a variable from the I/O Wiring.

Important: Create conversion tables from the device level before attaching these to variables.

## To apply a conversion table to analog variables

1. From the Application View, right-click the device, and then click Conversion Tables.

The Conversion Tables Editor appears.
2. In the Conversion Tables section, click Add.

A conversion table is added to the list.
3. In the Details section, specify a name for the conversion table, then define the required points for the conversion by clicking Add.
Tip: Each conversion table can have a maximum of 32 points.
4. Set the Electrical and Physical values for each point. A line graph representing the conversion table appears on the right-hand side of the Conversion Table Editor.

## See also

Wire the channels of an I/O device on page 643
Edit the points of an existing conversion table on page 646
1/O Wiring on page 638

## Edit the points of an existing conversion table

## Delete a conversion table

Edit the points of a conversion table to modify the proportional relationship between electrical values and physical values used in application programming.

## To edit the points of an existing conversion table

1. From the Application View, right-click the device, and then click Conversion Tables.
2. In the Conversion Tables section, select the conversion table to modify.
3. In the Details section, modify the conversion table:

- To edit the values of existing points, enter the new value then press (RH: select) Enter.
- To add a conversion point, click Add, and then enter the Electrical and Physical values for the point.
Tip: $\quad$ Each conversion table can have a maximum of 32 points.
- To remove conversion points, select the points and then click Remove.
A line graph representing the conversion table and its modifications displays on the right-hand side of the Conversion Table Editor.


## See also

Apply a conversion table to analog variables on page 645
I/O Wiring on page 638
Delete conversion tables no longer required for defining analog conversions.

## To delete a conversion table

1. From the Application View, right-click the I/O device, and then click Conversion Tables.
2. In the Conversion Tables section, select the conversion table to remove, then click Remove.

## See also

Edit the points of an existing conversion table on page 646
I/O Wiring on page 638

## I/0 wiring keyboard shortcuts

The following keyboard shortcuts are available for use with I/O wiring. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl +N | Adds a device (not available while debugging). |
| Ctr +F | Unwires all channels of selected devices (not available while debugging). |
| Ctrl +R | Unwires selected channels of a device (not available while debugging). |
| Ctrl +H | Toggles between a real or virtual I/O device (not available while debugging). |
| Ctrl +L | While debugging, toggles between locking and unlocking selected channels. |

## See also

I/O Wiring on page 638
The Functional Block Diagram (FBD) is a graphic language enabling programmers to build complex procedures by taking existing functions from the standard library, function section, or function block section.

In FBD containers, also include Ladder Diagram (LD) elements like coils, contacts, jumps, labels, and returns. However, in contrast to LD elements usage in LD containers where these elements follow strict graphical positioning regulations, LD elements within FBD container are independent of these regulations.

## See also

FBD Diagram Main Format on page 647
Debug FBD Programs on page 649
FBD Elements on page 649
Function Block Diagram Settings on page 84

## FBD Diagram Main Format

Function Block Diagram (FBD) diagrams describe a process between input variables and output variables. A process is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. Outputs of blocks can also be connected to inputs of other blocks.

Function Block


An entire process represented by an FBD program is built using the available variables, operators, functions, and function blocks. Each block has either a fixed or defined number of input and 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 inside its rectangular shape. Each input or output of a block is labeled and 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 literal, any internal or input variable, an output variable, or a block output.

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 can be used to connect two logical points of a 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

The connection is oriented, meaning that the line carries associated data from left to right. The left and right ends of the connection line must be of the same data type.

Vertical bars accept 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. All ends of the connections must be of the same data type.

## See also

Vertical Bars on page 652
FBD Variables on page 650

Regions on page 320
Comments on page 664

## Debug FBD Programs

When debugging Functional Block Diagram (FBD) programs, monitor the output values of elements. These values display using color, numeric, or textual values according to their data type:

- Output values of Boolean type display using color. The output value color continues to the next input. When the output value is unavailable, Boolean elements remain black. The colors are red when True and blue when False.
- Output values of DINT, REAL, TIME, and MESSAGE type display as a numeric or textual value in the element.

When the output value for a numeric or textual value is unavailable, the WAIT text displays in the output label. Values also display in the corresponding dictionary instance.

When the program is debugging, perform these operations:

| Task | Procedure |
| :--- | :--- |
| Change the cycle timing | From the Target Execution toolbar, set the required value in the Cycle Timing (ms) box <br> Cycle Timing (ms): 700 |
| Toggle the logical value of a contact, <br> coil, or block input | • Select the required element, then right-click Toggle Boolean Value (or press Ctrl+T). |

## See also

FBD Variables on page 650
Data Types on page 524
Enable power flow debugging on page 524

## FBD Elements

When programming in Function Block Diagram (FBD), drag elements from the Toolbox into the workspace. These elements are available for FBD program organization units (POUs):

- Blocks
- Variables
- Vertical Bars
- Labels
- Jumps
- Returns
- Rungs
- Left Power Rails
- Right Power Rails
- Coils
- Contacts
- Regions
- Comments


## See also

## FBD Variables on page 650

Coils on page 657
Contacts on page 660
Reset Container and Shape Settings in FBD POUs on page 34

## Blocks in FBD editors

## FBD Variables

## To access the Parameters view

In the Application View, right-click the required function or function block, then click Parameters.

Block elements are operators, functions, or function blocks. Connect block inputs and outputs to variables, contacts or coils, or other block inputs and outputs. Insert and resize block elements in language editors to create program organization units (POUs).

The Function Block Diagram (FBD) editor displays the name of the function, function block, or operator, and the parameter names that represent functions or function blocks. AADvance ${ }^{\circledR}$-Trusted ${ }^{\circledR}$ SIS Workstation software instantiates function blocks.

For functions, the return parameter is the only output. For function blocks, multiple return parameters provide multiple outputs. The return parameter of a function has the same name as the function. The return parameters of a function block has any name. Define the parameters of functions and function blocks in the Parameters view.

## To insert a block element

1. From the Toolbox, drag the block element into the language container. The block selector displays.
2. In the Block Selector, choose the required function block, then click OK. Sort the block list according to the columns by setting these in ascending or descending order.
The selected block displays in the language container.

## See also

FBD Variables on page 650
FBD Diagram Main Format on page 647
FBD Keyboard Shortcuts on page 665
To connect a new symbol to an existing one (another variable, a block input, or a block output) in the workspace, 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.


Replace existing variables in program organization units (POUs) by doubleclicking them to access the Variable Selector or single-clicking them to select from a drop-down combo-box containing the global and local variables. Also, single-click a variable, then type a literal value in the text box provided. When inserting literal values beginning with a letter or an underscore, enclose these in single quotes as follows: 'abc'.


When selecting items like local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

For input and output variables, choose to display comments entered in the dictionary. From the View menu, access the Properties window and define the Comment Position property.


Resize variables that display in the workspace.

## To insert a variable

1. From the Toolbox, drag the variable element into the language container. The Variable Selector is displayed.
2. In the Variable Selector, select the required variable, then click OK.

## See also

## FBD Diagram Main Format on page 647

FBD Keyboard Shortcuts on page 665

## Vertical Bars

## Labels

Vertical bars are graphic components of Function Block Diagram (FBD) programs enables closing multiple parallel links. More than one horizontal links on the left side of a vertical bar are connected to one link on the right side. The Boolean state of the right end is the logical OR between all the left extremities.


## To insert a vertical bar

- From the Toolbox, drag the vertical bar element into the language container.

Labels can be placed anywhere in a Function Block Diagram (FBD) diagram. These are used as a target for jump instructions, to change the execution order of the diagram. Labels are not connected to other elements.

Place labels on the left of the diagram in order to increase diagram readability. Labels are used to control the execution of the diagram. No other object may be connected on the right of a label symbol.

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



## To insert a label

1. From the Toolbox, drag the label element into the language container.
2. In the language container, click the label, then type a label name in the space provided. The label is displayed in the language container.

## See also

## Jumps on page 653

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.

Jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump symbol.

Inserting backward jumps in the diagram may block the programmable logic controller (PLC) loop.

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



Tip: Before inserting jumps, define one or more labels within the program.

## To insert a jump to a label

1. From the Toolbox, drag the jump element into the language container.
2. In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed in the language container with the required label name.

## See also

Labels on page 652

## Returns

If the connection line (to the left of the Return element) 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 element.
The Return element must be connected to a Boolean output connection point of a block. This element represents a Conditional End of the program: if the output of the box connected to the element 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;
While inserting or dragging a Return element, drop points indicate the possible locations where the contact can be placed on a rung.

## To insert a return

- From the Toolbox, drag the Return element into the language container, placing it on the rung where a drop point is displayed.


## See also

## FBD Diagram Main Format on page 647

## Rungs

Rungs are graphic components of Function Block Diagram (FBD) programs and represent a group of circuit elements leading to the activation of a coil. Dragging the rung element into the workspace inserts a left power rail linked to a right power rail. Also, the rung contains a direct contact and a direct coil.
Error symbols ( ${ }^{1}$ ) indicate that the direct contact and direct coil are undefined.


## To insert an FBD rung

- From the Toolbox, drag the rung element into the language container.


## See also

Coils on page 657
Contacts on page 660
Right Power Rails on page 656
Left Power Rails on page 656
Right Power Rails are graphic components of Functional Block Diagram (FBD) programs that represent the right boundary of a rung.

You can link right power rails to left power rails and many FBD and Ladder Diagram (LD) elements, including variables, blocks, vertical bars, coils, and contacts.

## To insert a right power rail

- From the Toolbox, drag the right power rail element into the language container.


## See also

Left Power Rails on page 656
Coils on page 657
Contacts on page 660

## FBD Variables on page 650

Left Power Rails are graphic components of Functional Block Diagram (FBD) programs that represent the left boundary of a rung. Any horizontal link connected to a left power rail has the Boolean state TRUE.

You can link left power rails to right power rails and many FBD and Ladder Diagram (LD) elements, including variables, blocks, jumps, returns, vertical bars, coils, and contacts.

## To insert a left power rail

- From the Toolbox, drag the left power rail element into the language container.


## See also

Right Power Rails on page 656
Coils on page 657
Contacts on page 660
FBD Variables on page 650

## Coils

Coils are graphic components of Ladder Diagram (LD) programs used in Function Block Diagram (FBD) programs representing the assignment of Boolean outputs. A coil represents an action. It must be connected on the left to a Boolean symbol, like a contact or the Boolean output of a block.

Types of coils available from the FBD toolbox are:

- Direct Coil
- Reverse Coil
- Set Coil
- Reset Coil

Change the type of a coil at any time after insertion.
When inserting coils in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables display above the coil elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.


| Task | Procedure |
| :--- | :--- |
| Insert an FBD coil | 1. From the Toolbox, drag the desired coil type into the language container and place it on the rung. <br> The Variable Selector is displayed. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> The coil element and its associated variable name are displayed in the language container. |
| Insert a parallel FBD coil | 1. From the Toolbox, drag the coil element into the language container while placing it parallel to the existing coil. <br> 2. Drag the left and right connections to the respective connection points on the rung. <br> The required coil is displayed on the parallel branch. |
| Change the type of a coil in FBD | - In the language container, select the coil, and then select the required type in the Modifier property of the Properties <br> window. |

## See also

Direct Coil on page 658
Reverse Coil on page 658
Set Coil on page 659
Reset Coil on page 660

## Direct Coil



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 can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := input1;
output2 := input1;
```


## See also

Reverse Coil on page 658
Set Coil on page 659
Reset Coil on page 660
Coils on page 657
Reverse Coil
Reverse 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. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```


## See also

## Direct Coil on page 658

Set Coil on page 659

## Reset Coil on page 660

Coils on page 657

## 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)

```
IF input1 THEN
    output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END_IF;
```


## See also

## Direct Coil on page 658

Reverse Coil on page 658
Reset Coil on page 660
Coils on page 657
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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)

```
output1 := input1 AND input2;
```


## See also

Direct Coil on page 658
Reverse Coil on page 658
Set Coil on page 659
Coils on page 657

## Contacts

Contacts are graphic components of Ladder Diagram (LD) diagrams used in Function Block Diagram (FBD) programs. Depending on the type of contact, it represents the value or function of an input or internal variable.

Contact types available from the FBD toolbox are:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

Change the type of a contact at any time after insertion.
When inserting contacts in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables display above the contact elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

| Select a variable from the <br> drop-down combo-box: |  | Type a literal value in the text |  |
| :--- | :--- | :--- | :--- |
| box: |  |  |  |


| Task | Procedure |
| :--- | :--- |
| Insert an FBD contact | 1. From the Toolbox, drag the desired contact type into the language container and place it on the rung. <br> The Variable Selector is displayed. |
| 2. In the Variable Selector, select the required variable, then click OK. |  |

## See also

Direct Contact on page 661
Reverse Contact on page 662
Pulse Rising Edge Contact on page 663
Pulse Falling Edge Contact on page 663

## Direct Contact

Direct contacts enable 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;
```


## See also

## Reverse Contact on page 662

Pulse Rising Edge Contact on page 663
Pulse Falling Edge Contact on page 663

## Contacts on page 660

Reverse contacts enable 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);
```


## See also

Direct Contact on page 661
Pulse Rising Edge Contact on page 663
Pulse Falling Edge Contact on page 663
Contacts on page 660

Pulse Rising Edge Contact

## Example


(* ST Equivalence: *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)

## See also

## Direct Contact on page 661

Reverse Contact on page 662
Pulse Falling Edge Contact on page 663
Contacts on page 660

## Pulse Falling Edge Contact

Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling 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 falls from TRUE to FALSE. The state 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 *)
```


## See also

Direct Contact on page 661
Reverse Contact on page 662

## Pulse Rising Edge Contact on page 663

## Contacts on page 660

Regions delineate and group together areas of a Function Block Diagram (FBD) program organization unit (POU). A region consists of a header and a delineated zone grouping together elements. The header section enables entering free-format text. After entering text in the header, click elsewhere in the region to exit editing mode. When moving the location of a region in the language container, also move all the content grouped within. Resize regions.


| Task | Procedure |
| :--- | :--- |
| Insert a region | From the Toolbox, drag the region element into the language container. |
| Move a region | 1. In the language container, left-click the top right corner of the region element and hold the mouse button. |
|  | 2. Drag the region element to the required location and release the mouse button. |

## See also

FBD Elements on page 649

## Comments

Comments are free format text inserted anywhere in the Function Block Diagram (FBD) program organization unit (POU), for documentation purposes only. After entering text, click elsewhere in the workspace to exit editing mode. For variables, this syntax displays the comment field of a variable as defined in the Dictionary:
@VarCom(VariableName)


Expand and collapse comment elements that display in the workspace by clicking the maximize and minimize buttons. Also resize comments.


## To insert a comment

1. From the Toolbox, drag the comment element into the language container.
2. In the language container, double-click the comment, then type the required text within the space provided.

## See also

Regions on page 320
FBD Elements on page 649
FBD Diagram Main Format on page 647

FBD Keyboard Shortcuts
These keyboard shortcuts are available for use with the Function Block Diagram (FBD) language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :---: | :---: |
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl +V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl $+X$ | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctri+ + | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |


| Keyboard Shortcut | Description |
| :---: | :---: |
| Enter | When a function block is selected, opens the Block Selector (not available while debugging). <br> When a variable is selected, opens the Variable Selector (not available while debugging). <br> When a comment is selected, starts editing it (not available while debugging). |
| Ctrl+Enter | When a variable is selected, opens the drop-down list of available variables (not available while debugging). <br> When editing a comment, confirms the text (not available while debugging). |
| Ctrl+- | Decreases the magnification |
| Ctrl+= | Increases the magnification |
| Ctrl+0 | 100\% magnification |
| Ctrl+ 1 | Inserts a variable (not available while debugging) |
| CtrI+2 | Inserts a function block (not available while debugging) |
| Ctrl+3 | Inserts a comment (not available while debugging) |
| Shift+Up Arrow | Reduces the height of the selected element (not available while debugging) |
| Shift+Down Arrow | Increases the height of the selected element (not available while debugging) |
| Shift+Left Arrow | Reduces the width of the selected element (not available while debugging) |
| Shift+Right Arrow | Increases the width of the selected element (not available while debugging) |
| Ctrl+Up Arrow | Moves the selection to the next element located higher in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Down Arrow | Moves the selection to the next element located lower in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Left Arrow | Moves the selection to the next element located to the left in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Right Arrow | Moves the selection to the next element located to the right in the diagram without keeping the previous element selected (not available while debugging) |
| Alt+Shift+Up Arrow | When a function block is selected, navigates up the different inputs and outputs (not available while debugging) |
| Alt+Shift+Down Arrow | When a function block is selected, navigates down the different inputs and outputs (not available while debugging) |
| Alt+Shift+Left Arrow | When a function block is selected, navigates left across the different inputs and outputs (not available while debugging) |
| Alt+Shift+Right Arrow | When a function block is selected, navigates right across the different inputs and outputs (not available while debugging) |
| Ctrl + Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Up Arrow | Moves selected elements up the language container. While debugging, scrolls up. |
| Down Arrow | Moves selected elements down the language container. While debugging, scrolls down. |
| Left Arrow | Moves selected elements left across the language container. While debugging, scrolls left. |
| Right Arrow | Moves selected elements right across the language container. While debugging, scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |

## 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 as an electric contact diagram. Thus, the term "ladder" coming from the concept of rungs connected to vertical power rails at both ends where each rung represents an individual circuit.


Adjust editor and view settings for individual or all Ladder Diagrams. When working in a Ladder Diagram, set the properties for the diagram from the Container properties in the Properties window. Set the properties for all Ladder Diagrams using the options available from the Tools menu. Some of the available properties include:

- background and gradient colors for operators, functions, and function blocks
- displaying the grid and the height and width of grid cells, in pixels
- the height and width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. For example, note the contact using the default settings of one grid cell high by four grid cells wide. The following block uses a basic element width for the inputs, another for the block, and another for the outputs. The block uses a basic element height for the EN/ENO level, another for the first input and the output, and another for the second input.

- the font type, size, style, and color applied to the text displayed in elements
- various options such as displaying comments and labels, aligning coils, and setting the colors for variables, labels, comments, power rails, and rung headers


## See also

Debug LD Programs on page 668
LD Elements on page 669
Ladder Diagram Settings on page 87
Debug LD Programs
When power flow debugging Ladder Diagram (LD) programs, monitor the output values of elements. These values display using color, numeric, or textual values according to their data type:

- Output values of Boolean type display using color. The output value color continues to the next input. When the output value is unavailable, Boolean elements remain black. The default colors are red when True and blue when False. Customize the colors used for Boolean items.
- Output values of DINT, REAL, TIME, and MESSAGE type display as a numeric or textual value in the element. When the output is a structure type, the display value is the selected member.


When the output value for a numeric or textual value is unavailable, the WAIT text displays in the output label. Transitional elements like Pulse rising edge (positive) contacts, with an unstable state, remain black. Values also display in the corresponding dictionary instance.

When the device is in the DEBUGGING state, perform one of these operations:

| Task | Procedure |
| :--- | :--- |
| Change the cycle timing | • From the Target Execution toolbar, set the required value in the Cycle Timing (ms) box <br> Cycle Timing (ms): <br> 700 |
| Toggle the logical value of a contact, coil, or block input | $\bullet$ Celect the required element, then right-click Toggle Boolean Value (or press Ctrl+ + T). |
| Important:Set the Generate debug information property to False if a build error <br> occurs when the In-line property is True for LD function blocks. |  |

## See also

## LD Elements on page 669

Data Types on page 524
LD Language on page 666

## LD Elements

When editing a Ladder Diagram (LD) program organization unit (POU), place elements in a language container by dragging them from the LD Toolbox or from a contextual menu accessed by right-clicking. An element is inserted at the current position in the diagram. When inserting subsequent elements, these are placed to the right of the selected element on the rung, then onto the next rung. These elements are available for LD POUs:

- Rungs
- Blocks
- Coils
- Contacts
- Jumps
- Returns
- Branches

While dragging rungs and elements from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For rungs, the possible drop points are displayed between the rung handles on the left. For elements, the possible drop points are displayed on the rung.

Drop points between rung handles while dragging a rung in a diagram


## See also

LD Rungs on page 670
Branches on page 688
Selecting LD Elements on page 346
LD Keyboard Shortcuts on page 690
Reset Visual Settings in LD POUs on page 35
Rungs are graphic components of Ladder Diagram (LD) programs and represent a group of circuit elements leading to the activation of a coil. Rungs have labels to identify them within the diagram. Labels along with jumps enable controlling the execution of a diagram. The label and jump must have the same name. When the connection on the left of the jump element has the

TRUE Boolean state, the diagram execution proceeds at the label element. Comments are free format text inserted above the rung, for documentation purposes only. To the left of a rung, a handle indicates the position within a diagram and enables repositioning by dragging and dropping.

While dragging rungs from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For rungs, the possible drop points are displayed between the rung handles on the left.


| Task | Procedure |
| :--- | :--- |
| Insert a rung | - From the Toolbox, drag the rung element into the language container where a drop point appears. |
| Drag a rung | 1. To reposition a rung within a diagram, select the rung handle while holding down the left mouse button. The rung handle <br> indicates the rung position on the left. <br> 2. Drag the rung to the required position in the diagram where a drop point is displayed. |
| Define the label for a rung | 1. In the language container, click anywhere, then from the contextual menu, choose Add Label. <br> 2. In the upper left-hand corner, click in the text area beside the grey square and type the required label text. |
| Define a comment for a rung | Place comments in the space above the rung. After entering text, click elsewhere in the workspace to 'validate' the comment. <br> Text formatting options including bold, italic, underline, strikethrough, and justify, are available from the Format menu. Using <br> the Format menu, define the foreground color. <br> - In the language container, click the rectangular space above the rung, then type the required text. |

## See also

LD Blocks on page 672
Coils on page 675
Contacts on page 681
Branches on page 688
Selecting LD Elements on page 346

In a language container, connect blocks to Boolean lines. Blocks can be operators, functions, or function blocks. Boolean inputs and outputs are not always contained within blocks. Boolean inputs connecting blocks to rungs are always executed each cycle. Boolean outputs connecting blocks to rungs control the remaining rung power flow. When inserting blocks in a diagram, the EN and ENO parameters are added to some block interfaces. Also force the inclusion of the EN and ENO parameters for blocks with either one Boolean input, one Boolean output, or no Boolean input and output. Activate the Enable EN/ENO option from the Ladder Diagram options.

For functions and function blocks, set the value of return parameters using coils. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.

Insert blocks from the LD Toolbox. Set the type of a block using the Block Selector at any time following insertion. When setting the type of block, variables automatically display and are connected to the inputs and outputs of the block.

Replace input and output variables by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combobox containing the global and local variables. Also, single-click a variable, then type a literal value in the text box provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.


When selecting items like local variables, global variables, and defined words from the drop-down combo-box, type characters in the text box to focus on the possible items.

## EN Input

For operators, functions, and function blocks where the first input is not a Boolean data type, another input called EN is automatically inserted at the first position since the first input is always connected to the rung. The block is
executed only when the EN input is TRUE. The following example shows a comparison operator and its equivalent code expressed in ST:


## ENO Output

For operators, functions, and function blocks where the first output is not a Boolean data type, another output called ENO is automatically inserted at the first position since the first output is always connected to the rung. The ENO output always has the same state as the first input of the block. The following example shows the AVERAGE function block and its equivalent code expressed in ST:


```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```


## EN and ENO Parameters

In some cases, both EN and ENO parameters are required. The following example shows an arithmetic operator and its equivalent code expressed in ST.


While dragging blocks from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung.

Drop points on a rung while dragging elements over the rung:


Perform these tasks for blocks in the LD editor:

| Task | Procedure |
| :--- | :--- |
| Access the Parameters view | - In the Application View, right-click the required function or function block, then click Parameters. |
| Insert a block | 1. From the Toolbox, drag the block element into the language container and place it on the rung where a drop point is <br> displayed. <br> The Block Selector is displayed. <br> 2. In the Block Selector, locate the required block. You can sort the block list according to the columns by setting these in <br> ascending or descending order. <br> - To force the inclusion of the EN/ENO parameters, select Enable EN/ENO. <br> 3. Click OK. <br> The selected block is displayed on the rung. |
| Insert a parallel block | 1. From the Toolbox, drag the branch element onto the existing block in the language container. <br> 2. Place a block element on the branch: <br> a. From the Toolbox, drag the block element into the language container. The block appears on the branch. <br> b. From the Block Selector, locate the block. Sort the block list by clicking the column heading to toggle between <br> ascending and descending order. |
| 3. Click OK. The selected block appears on the branch. |  |


| Task | Procedure |
| :--- | :--- |
| Drag a block | • To reposition a block within the diagram, select the block while holding down the left mouse button, then drag the block to <br> the required location on a rung where a drop point is displayed. |

## See also

LD Rungs on page 670

## Selecting LD Elements on page 346

Coils are graphic components of Ladder Diagram (LD) programs and represent the assignment of Boolean outputs. In an LD program, a coil represents an action. It must be connected on the left to a Boolean symbol, like a contact or the Boolean output of a block.

Types of coils are available from the LD toolbox are:

- Direct Coil
- Reverse Coil
- Pulse Rising Edge Coil
- Pulse Falling Edge Coil
- Set Coil
- Reset Coil

Change the type of a coil at anytime after insertion.
When inserting coils in POUs, assign variables using the Variable Selector. Names of assigned variables display above the coil elements within POUs. Replace existing variables by double-clicking the coil to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.


When selecting items like local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

While dragging coils from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung. For instance, since a rung can only have one coil on the right, the editor does not display possibilities when dragging another coil onto the rung. However, if a
parallel branch is placed around the coil, the editor displays a possible location for another coil on the branch.

Drop points on a run while dragging elements over the rung


| Task | Procedure |
| :--- | :--- |
| Insert a coil | 1. From the Toolbox, drag the desired coil type into the language container and place it on the rung where a drop point <br> appears. <br> The Variable Selector appears. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> The coil element and its associated variable name appear on the rung. |
| Insert a parallel coil | 1. From the Toolbox, drag the branch element onto the existing coil in the language container. <br> 2. From the Toolbox, drag the desired coil type onto the branch. <br> The Variable Selector appears. <br> 3. In the Variable Selector, select the required variable, then click OK. <br> The coil element and its associated variable name appear on the branch. |
| Drag a coil | - To reposition a coil within the diagram, select the coil while holding down the left mouse button, then drag the coil to the <br> required location on a rung where a drop point is displayed. |
| Change the type of a coil | •In the language container, select the coil, then press the space bar. |
| Align all coils in a diagram | 1. Right-click in the language container, then choose Properties from the contextual menu. <br> 2. In the Properties window, set the Coil Alignment property to True. |

## See also

Direct Coil on page 676
Reverse Coil on page 677
Pulse Rising Edge Coil on page 678
Pulse Falling Edge Coil on page 679
Set Coil on page 679
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 can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := input1;
output2 := input1;
```


## See also

Coils on page 675
Reverse Coil on page 677
Set Coil on page 679
Reset Coil on page 680
Pulse Rising Edge Coil on page 678

## Reverse Coil

Reverse 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. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

## Example


(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```


## See also

Coils on page 675
Direct Coil on page 676
Pulse Falling Edge Coil on page 679
Set Coil on page 679

## Reset Coil on page 680

## Pulse Rising Edge Coil

Pulse rising edge coils 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)

```
IF (inputl and NOT(inputlprev)) THEN
        output1 := TRUE;
ELSE
        output1 := FALSE;
END_IF;
(* inputlprev is the value of inputl at the previous
cycle *)
```


## See also

Coils on page 675
Direct Coil on page 676

## Pulse Falling Edge Coil on page 679

## Set Coil on page 679

Reset Coil on page 680

## Pulse Falling Edge Coil

Pulse falling edge coils 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

## Example


(* ST Equivalence: *)
IF (NOT (input1) and input1prev) THEN
output1 := TRUE;
ELSE
output1 := FALSE;
END_IF;
(* inputlprev is the value of input1 at the previous cycle *)

## See also

Coils on page 675
Direct Coil on page 676
Reverse Coil on page 677
Pulse Rising Edge Coil on page 678
Reset Coil on page 680

## 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

Example

(* ST Equivalence: *)

```
IF input1 THEN
        output1 := TRUE;
END_IF;
IF input2 THEN
        output1 := FALSE;
END_IF;
```


## See also

## Coils on page 675

Reset Coil on page 680
Direct Coil on page 676
Reverse Coil on page 677

## 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. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

(* ST Equivalence: *)

```
IF input1 THEN
        output1 := TRUE;
END_IF;
IF input2 THEN
    output1 := FALSE;
END IF;
```


## See also

## Coils on page 675

Set Coil on page 679
Pulse Rising Edge Coil on page 678
Pulse Falling Edge Coil on page 679
Direct Coil on page 676

## Contacts

Contacts are graphic components of Ladder Diagram (LD) diagrams.
Depending on the type of contact, it represents the value or function of an input or internal variable.

Contact types available from the LD stencil are:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

Change the type of a contact at any time after insertion.
When inserting contacts in program organization units (POUs), assign variables using the Variable Selector. Names of assigned variables display above the contact elements within POUs. Replace existing variables by double-clicking the variable names to access the Variable Selector or by singleclicking variable names to select from drop-down combo boxes containing the global and local variables. Also single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:


Type a literal value in the text box:


When selecting items like local variables, controller variables, system variables, and defined words from the drop-down combo box, type characters in the text box to focus on the possible items.

While dragging contacts from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung.

Drop points on a rung while dragging elements over the rung


| Task | Procedure |
| :--- | :--- |
| Insert a contact | 1. From the Toolbox, drag the desired contact type into the language container and place it on the rung where a drop <br> point appears. <br> The Variable Selector appears. <br> 2. In the Variable Selector, select the required variable, then click OK. <br> The contact and its associated variable name appear on the rung. |
| Insert a parallel contact | 1. From the Toolbox, drag the branch element on the existing contact in the language container. <br> 2. From the Toolbox, drag the desired contact type on the branch. <br> The Variable Selector appears. |
| 3. In the Variable Selector, select the required variable, then click OK. |  |
| The contact and its associated variable name appear on the branch. |  |

## See also

Direct Contact on page 682
Reverse Contact on page 683
Pulse Rising Edge Contact on page 684

## Pulse Falling Edge Contact on page 684

## Direct Contact

Direct contacts enable 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;
```


## See also

## Reverse Contact on page 683

Pulse Rising Edge Contact on page 684
Pulse Falling Edge Contact on page 684
Contacts on page 681
Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.


Left Connection Right Connection
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);
```


## See also

## Direct Contact on page 682

Pulse Rising Edge Contact on page 684
Pulse Falling Edge Contact on page 684

## Contacts on page 681

## Pulse Rising Edge Contact

Pulse rising edge (positive) contacts enable 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. The state 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 *)
```


## See also

Direct Contact on page 682
Reverse Contact on page 683
Pulse Falling Edge Contact on page 684
Contacts on page 681

## Pulse Falling Edge Contact

Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling 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 falls from TRUE to FALSE. The state 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 *)

## See also

Direct Contact on page 682
Reverse Contact on page 683

## Pulse Rising Edge Contact on page 684

Contacts on page 681

## Jumps

Conditional and unconditional jump elements enable controlling the execution of diagrams. Connections cannot be placed to the right of a jump element. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label. The label and jump must have the same name.

Inserting backward jumps in the diagram may block the PLC loop.

Example:


While dragging jumps from either the toolbox or within a diagram, the language editor displays the possible drop points. Drop points are displayed as plus signs in a black circle. For elements, the possible drop points are displayed on the rung. For instance, since a rung can only have one jump, the editor does not display possibilities when dragging another jump onto the rung. However, if a parallel branches are placed on a rung, the editor displays possible locations for jumps on the rung and on all branches.


| Task | Procedure |
| :--- | :--- |
| Insert a jump | Before inserting jumps, define one or more labels within the program. Insert jumps from the Toolbox, contextual menus, or <br> using keyboard shortcuts. While inserting or dragging a jump, drop points indicate the possible locations where the jump <br> can be placed on a rung. <br> 1. From the Toolbox, drag the jump element into the language container and place it on the rung where a drop point is <br> displayed. <br> 2. In the language container, click the jump element, then select the required label name from the drop-down combo-box. |


| Task | Procedure |
| :--- | :--- |
| Drag a jump | When dragging jumps, the possible drop points on a rung are displayed while moving over the rung. <br> - To reposition a jump within the diagram, select the jump while holding down the left mouse button, then drag the jump to <br> the required location on a rung where a drop point is displayed. |

## See also

## LD Rungs on page 670

## Returns

Use Return elements as outputs representing a conditional end of a diagram. Connections cannot be placed to the right of a Return element.

When the left connection line has the TRUE Boolean state, the diagram ends without executing the equations located on the next lines of the diagram.

When the LD diagram is a function, its name is associated with an output coil to set the return value (returned to the calling diagram).

While dragging Return elements from either the Toolbox or within a diagram, the language editor displays the possible drop points. Drop points display as plus signs in a black circle. For elements, the possible drop points display on the rung. For instance, since a rung can only have one Return, the editor does not display possibilities when dragging another Return onto the rung. However, if a parallel branch is placed around the Return, the editor displays a possible location for another Return on the branch.


Insert Return elements from the Toolbox, contextual menus, or using keyboard shortcuts. While inserting or dragging a Return element, drop points indicate the possible locations where the contact can be placed on a rung.

| Task | Procedure |
| :--- | :--- |
| Insert a return | 1. From the Toolbox, drag the Return element into the language container, placing it on the rung where a drop point is displayed. <br> 2. (optional) To reposition a return within the diagram, select the Return element while holding down the left mouse button, then drag <br> the Return element to the required location on a rung where a drop point is displayed. |
| Drag a return | - To reposition a Return element within the diagram, select the Return element while holding down the left mouse button, then drag <br> the Return element to the required location on a rung where a drop point is displayed. |

Example

(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```


## See also

## LD Elements on page 669

## Branches

Branches create alternative routing for connections. Add parallel branches to elements on a rung. Parallel branch structures display with connections upon clicking a rung.

To perform operations like cutting, copying, pasting, and deleting branches, select a single branch or all branches from a parallel structure. When selecting a single branch, click the connection on the left or right extremity of the branch. The branch connection is highlighted and the branch is outlined. When selecting all branches, click any of the vertical lines joining the branch connections on the left or right extremity of the branches. The branch arm is highlighted and all branches in the structure are outlined.


Single Branch Selection


All Branches Selection

While dragging branches from either the Toolbox or within a diagram, the language editor displays the possible drop points. Drop points are display as plus signs in a black circle. For elements, possible drop points display on the rung.

Drop points on a rung while dragging elements over the rung


Tip: When dragging branches, the possible drop points on a rung display while
moving over the rung.

## To insert a branch

1. From the Toolbox, drag the branch element into the language container and place in on the rung where a drop point appears.
2. (optional) To reposition a branch within the diagram, select the branch while holding down the left mouse button, then drag the branch to the required location on a rung where a drop point displays.

## See also

Coils on page 675
Contacts on page 681

## Selecting LD Elements

In Ladder Diagram (LD) program organization units (POUs), select individual or multiple LD elements to perform management tasks like cutting, copying, pasting, and deleting. Select multiple elements by holding down Ctrl while clicking the left mouse button or by dragging to enclose.

Only select elements having the same level on a rung; elements from different rungs are unavailable. Upon clicking a rung, parallel branch structures display with connections. This diagram shows the selection of two contacts and a parallel branch structure on a rung. These three elements are at the same level while the individual parallel branches are sub-elements of the branch structure.


For branches, select a single branch or all branches from a structure. When selecting a single branch, click the connection on the left or right extremity of the branch. The connection for the selected branch highlights and the branch is outlined. When selecting all branches, click any of the vertical lines joining
the branch connections on the left or right extremity of the branches. The branch arm highlights and all branches in the structure are outlined.
Single Branch Selection

## To select elements

1. To select a single element, click the element on the rung.
2. To select multiple elements, click each of the required elements while holding down the Ctrl key.
3. To select a single branch from a parallel branch structure, click the connection on the left or right extremity of the branch.
The branch connection is highlighted and the branch is outlined.
4. To select all branches from a parallel branch structure, click any of the vertical lines joining the branch connections on the left or right extremity of the branches.
The branch arm is highlighted and all branches are outlined.

## LD Keyboard Shortcuts

Use these keyboard shortcuts in Ladder Diagram (LD) POUs to perform many editing operations and navigate the elements making up a diagram. When navigating an LD diagram, the arrow keys enable moving up and down and left and right from rung to rung and from element to element. Enable navigating in a diagram by clicking in the language container.

These keyboard shortcuts are available for use with the LD language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+A | Selects all rungs (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while <br> debugging) |
| Ctrl +V | Pastes elements saved on the clipboard to the insertion point (not available <br> while debugging) |
| Ctrl +X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift + Ctrl+Alt+ +G | Enables/disables the grid in rungs (only available when the Display Grid <br> property is set to True) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl +R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically <br> opens the Block Selector and Variable Selector (not available while <br> debugging). |


| Keyboard Shortcut | Description |
| :---: | :---: |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+ | Italicizes selected comment text (not available while debugging) |
| CtrI+U | Underlines selected comment text (not available while debugging) |
| Enter | Calls the Variable/Block selector depending on the selected element (not available while debugging) |
| Space Bar | For coils or contacts, toggles between the available types (not available while debugging) |
| Ctrl+0 | Inserts a rung below a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+Alt+0 | Inserts a rung above a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl +1 | Inserts a branch after a selected element (not available while debugging) |
| Ctrl + Alt +1 | Inserts a branch before a selected element (not available while debugging) |
| Ctrl+2 | Inserts a block after a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+Alt+2 | Inserts a block before a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+3 | Inserts a contact after a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+Alt+3 | Inserts a contact before a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| CtrI+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl +5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl +7 | When an element is selected, surrounds the element with a branch (not available while debugging). |
| Ctrl + Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Ctrr+Up Arrow | Slowly scrolls up. |
| Ctrl+Down Arrow | Slowly scrolls down. |
| Ctrl+Left Arrow | Slowly scrolls left. |
| CtrI+Right Arrow | Slowly scrolls right. |
| Up Arrow | Moves up the elements. |
| Down Arrow | Moves down the elements. |
| Left Arrow | Moves to the left across the elements. |
| Right Arrow | Moves to the right across the elements. |
| Alt+Up Arrow | Selects the previous rung. When no element or rung is selected, selects the last rung. |
| Alt+Down Arrow | Selects the next rung. When no element or rung is selected, selects the first rung. |
| Alt+Left Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Alt+Right Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Shift+Up Arrow | Scrolls up |
| Shift+Down Arrow | Scrolls down |
| Shift+Left Arrow | Scrolls left |
| Shift+Right Arrow | Scrolls right |


| Keyboard Shortcut | Description |
| :--- | :--- |
| Delete | Removes a selected rung or element (not available while debugging) |
| Ctrl +D | Edits the comment of the selected rung (not available while debugging) |
| Ctrl +T | Only available in debug mode for the Boolean data type. Toggles the logical <br> value of a selected contact, coil or block input. |

## ST Language

## See also

## ST Main Syntax on page 692

Expressions and Parentheses on page 694
Debug ST Programs on page 696
ST Basic Elements and Statements on page 697
Structured Text Settings on page 90

## ST Main Syntax

A Structured Text (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 enable the inclusion of non-executed information throughout code. Insert comments anywhere in an ST program. Comments can run multiple lines and must begin with "(*" and end with "*)". Do not use interleave comments, for example, comments within comments.

When typing statements, a drop-down combo-box automatically lists the available items such as identifiers, operators, functions, and function blocks. The listed items are focused by typing letters, digits, and underscore characters.

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

When entering ST syntax, basic coding is black while other items are displayed using customizable colors. The default colors for ST elements are:

- Comments are green
- The Editor background is white
- Identifiers are black
- Numbers are firebrick
- Operators are black
- Program organization units (POUs) are blue-violet
- Punctuation marks are black
- Reserved words are fuchsia
- Strings of text are gray

Inactive separators between active separators, literals, and identifiers increase ST program legibility. ST inactive separators are: space (blank), tabs and end of line. Place end of lines anywhere in a program. These rules apply to using inactive separators:

- Write one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase legibility 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;
```

| Task | Procedure |
| :--- | :--- |
| Customize the default display settings <br> for ST programs | 1. From the Tools menu, click Options. <br> 2. From the Options dialog box, expand IEC Languages, and then click Structured Text (ST). <br> 3. Expand the respective category, customize the required setting, then click OK. <br> The customized settings are now the default values for Structured Text (ST) programs. |
| Customize the display settings for the <br> current ST program | 1. From the View menu, click Properties Window. The Properties window is displayed. <br> 2. Select the Structured Text (ST) Container. <br> 3. From the Properties Window: <br> - Customize the font for the required item by clicking <br> of the font, text size, bold, italic, strikeout, and underline styles. Font dialog box is displayed allowing for customization <br> - Customize the text color for the required items. The possible colors are custom, web, and system colors. <br> The customized settings only affect the current ST program. |

## See also

## Calling Functions on page 695

Calling Function Blocks on page 696

## Expressions and Parentheses

Structured Text (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:

| Single expression |  |
| :--- | :--- |
| (boo_var1 AND boo_var2) | Has BOOL type |
| not (boo_var1) | Has BOOL type |
| $(\sin (3.14)+0.72)$ | Has DINT type |
| $($ t\#1s23 +78$)$ | Is an invalid expression |

Parentheses are used to isolate sub parts of an expression and to explicitly order the priority of operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default priority between ST operators.

| Precedence | Operators | Symbols |
| :--- | :--- | :--- |
| 1 (Highest) | Function evaluation | identifier(arguement list) <br> For example: MAX(X,Y) |
| 2 | Negation | - |
|  | Complement | NOT |
| 3 | Multiplication | $*$ |
|  | Division | $I$ |
| 4 | Addition | + |
| 5 | Subtraction | - |
| 6 | Comparison | $<_{1}>_{1}<==_{,}>=$ |
| 7 | Equality | $=$ |
| 7 | Inequality | $<>$ |
| 9 | Boolean AND | $\& \&$, AND |
| 9 (Lowest) | Boolean Exclusive OR | Boolean OR |

## Examples

Parentheses examples

| $2+3 * 6$ | equals $2+18=20$ | Because multiplication operator has a higher priority |
| :--- | :--- | :--- |
| $(2+3) * 6$ | equals $5 * 6=30$ | Priority is given by parenthesis |

## See also

## ST Main Syntax on page 692

The Structured Text (ST) programming language enables calling functions. Function calls can be used in any expression.

| Function calls |  |
| :---: | :---: |
| Name | Name of the called function written in IEC 61131-3 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 |
| When setting the value of the return parameter in the body of a function, assign the return parameter using the same name as the function: |  |
| Function | := <expression>; |

## Example

Example1: IEC 61131-3 function call

```
(* Main ST program *)
(* gets an integer value and converts it into a limited
time value *)
dint_timeprog := SPlimit ( tprog_cmd );
appl_timer := TMR (dint_timeprog * 100);
(* Called FBD function named 'SPlimit' *)
```

Example2: "C" function call - same syntax as for IEC 61131-3 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);
```


## See also

Calling Function Blocks on page 696
ST Main Syntax on page 692

## Calling Function Blocks

## 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 (trigbl.Q) Then nb_edge := nb_edge + 1; End_if;
```


## See also

Calling Functions on page 695
ST Main Syntax on page 692
When debugging Structured Text (ST) programs, monitor the output values of elements by viewing the dictionary instances. Change the cycle timing to debug.

## To change the cycle timing

- From the Target Execution toolbar, select the required value in the Cycle Timing (ms) list Cycle Timing (ms): $700 \quad$.


## See also

ST Main Syntax on page 692
ST Basic Elements and Statements on page 697

## ST Basic Elements and Statements

## Assignments

The basic elements and statements of the Structured Text (ST) language are:

- Assignments
- CASE Statement
- EXIT Statement
- FOR Statement
- IF-THEN-ELSIF-ELSE-END_IF Statement
- REPEAT Statement
- RETURN Statement
- WHILE Statement


## See also

Assignments on page 355
CASE Statement on page 698
EXIT Statement on page 356
FOR Statement on page 357
ST Main Syntax on page 692
The expression can be a call to a function.

| Assignments |  |
| :--- | :--- |
| 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 <br> type |

## 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) );
```


## To insert an assignment

- In the language container, type :=.

CASE Statement

## 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;
```


## To insert a CASE

- From the Toolbox, drag the CASE element into the language container.


## See also

IF-THEN-ELSIF-ELSE-END IF Statement on page 700

## WHILE Statement on page 361

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

## EXIT Statement

| Name | EXIT |
| :--- | :--- |
| Meaning | exit from a FOR, WHILE or REPEAT iteration statement |
| Syntax | EXIT; |

## 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;
```


## To insert an EXIT

- In the language container, type EXIT.


## See also

Assignments on page 355
FOR Statement on page 357
REPEAT Statement on page 359
WHILE Statement on page 361
The FOR statement executes a limited number of iterations, using an integer index variable.

| FOR Statement |  |
| :--- | :--- |
| Name | FOR ... TO ... BY ... DO ... END_FOR |
| Syntax | FOR <index> := <mini> TO <maxi> BY <step> DO <br> $<$ <tatement> ; <br> <statement> ; <br> END_FOR; |
| Operands | index: internal integer variable increased at each loop <br> mini: initial value for index (before first loop) <br> maxi: maximum allowed value for index <br> step: index increment at each loop |

The [ BY step ] statement is optional. If not specified, the increment step is 1.

| Important: | Because the virtual machine is a synchronous system, input <br> 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;
```


## To insert a FOR statement

- From the Toolbox, drag the FOR element into the language container.


## See also

EXIT Statement on page 356
REPEAT Statement on page 359
WHILE Statement on page 361
Assignments on page 355
IF-THEN-ELSIF-ELSE-END_IF Statement

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 can be used more than once. The ELSE statement, if used, must appear only once at the end of the 'IF, ELSIF...' sequence.

| Name | IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF |
| :---: | :---: |
| Meaning | Executes one of several lists of Structured Text (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;``` |

## 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;
```


## To insert an IF-THEN-ELSIF-ELSE-END_IF

- From the Toolbox, drag the IF THEN ELSE element into the language container.


## See also

## CASE Statement on page 698

WHILE Statement on page 361
The REPEAT statement is an iteration structure for a group of Structured Text (ST) statements. The "continue" condition is evaluated AFTER any iteration

## REPEAT Statement

| Name | REPEAT ... UNTIL ... END_REPEAT |
| :---: | :---: |
| Syntax | REPEAT <br> <statement> ; <br> <statement> ; <br> UNTIL <Boolean_condition> <br> END REPEAT ; |


| Important: | Because the virtual machine is a synchronous system, input <br> variables are not refreshed during REPEAT iterations. The change of <br> state of an input variable cannot be used to describe the ending <br> condition of a REPEAT statement. |
| :--- | :--- |

## Example

```
(* ST program using REPEAT statement *)
(* this program uses specific "C" functions to read
characters *)
(* on a serial port *)
str := ''; (* empty string *)
nbchar := 0;
```

```
IF ComIsReady ( ) THEN
REPEAT
str := str + ComGetChar ( );
nbchar := nbchar + 1;
UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
END_REPEAT;
END_IF;
```


## To insert a REPEAT statement

- From the Toolbox, drag the REPEAT element into the language container.


## See also

Assignments on page 355
EXIT Statement on page 356
WHILE Statement on page 361

## RETURN Statement

Ends the execution of the current program.

| Name | RETURN |
| :--- | :--- |
| Syntax | RETURN ; |
| Operands | (none) |

## 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; (* ends the program *)
end_if;
if RESET then
    CV := 0;
else
    if (CV < PV) then
```

```
        CV := CV + 1;
        end_if;
end_if;
Q := (CV >= PV);
```


## To insert a RETURN

- In the language container, type RETURN.


## See also

Assignments on page 355
CASE Statement on page 698
EXIT Statement on page 356
IF-THEN-ELSIF-ELSE-END IF Statement on page 700
REPEAT Statement on page 359
The WHILE statement is an iteration structure for a group of Structured Text (ST) statements. The "continue" condition is evaluated BEFORE any iteration

| WHILE Statement |  |
| :--- | :--- |
| Name | WHILE ... DO ... END_WHILE |
| Syntax | WHILE <Boolean_expression> DO <br> $<$ statement> ; <br> <statement> ; |
| Important: | Since the virtual machine is a synchronous system, input variables <br> are not refreshed during WHILE iterations. The change of state of an <br> input variable cannot be used to describe the condition of a WHILE <br> statement. |

## Example

```
(* ST program using WHILE statement *)
(* this program uses specific "C" functions to read
characters *)
(* on a serial port *)
str := ''; (* empty string *)
nbchar := 0;
WHILE ((n.bchar < 16) & ComIsReady ( )) DO
str := str + ComGetChar ( );
nbchar := nbchar + 1;
END_WHILE;
```


## To insert a WHILE statement

- From the Toolbox, drag the WHILE element into the language container.


## See also

Assignments on page 355
EXIT Statement on page 356
REPEAT Statement on page 359
These statements and functions are extensions of the Structured Text (ST) language:

| ST Statement and Functions |  |
| :--- | :--- |
| TSTART | Starts a timer |
| TSTOP | Stops a timer |

## See also

TSTART Statement on page 704
TSTOP Statement on page 705
Starts the counting of a timer variable. The timer value is not modified by the TSTART command,for example, the counting starts from the current value of the timer.

| TSTART Statement |  |
| :--- | :--- |
| Name | TSTART |
| Syntax | TSTART ( <timer_variable> ); |
| Operands | Any inactive timer variable |
| Return value | (none) |

## Example

```
(* ST program using the TSTART statement *)
bo100 := TRUE; (* boolean output *)
tm_ctrl := t#0s;
TSTART(tm_ctrl);
```


## To insert a TSTART statement

- In the language container, type TSTART.


## See also

TSTOP Statement on page 705
Stops updating a timer variable. The timer value is not modified by the TSTOP command.

| TSTOP Statement |  |
| :--- | :--- |
| Name | TSTOP |
| Syntax | TSTOP ( <timer_variable> ); |
| Operands | Any active timer variable |
| Return value | (none) |

## Example

(* ST program using the TSTOP statement *)
TSTOP(tm_ctrl);
alarm := not(bi100):

## To insert a TSTOP statement

- In the language container, type TSTOP.


## See also

TSTART Statement on page 704
Boolean operators are available for the Structured Text (ST) language:

## ST Boolean Operators

| FEDGE | Evaluates the falling edge of a Boolean expression |
| :--- | :--- |
| REDGE | Evaluates the rising edge of a complete Boolean expression |

Additional Boolean operators are available:

| AND | Boolean AND between 2 to 32 Boolean variables |
| :--- | :--- |
| $\mathbf{O R}$ | Boolean OR of 2 to 32 Boolean variables |
| $\mathbf{X O R}$ | Boolean exclusive OR between two terms |

## See also

FEDGE on page 706
REDGE on page 706
AND on page 748
OR on page 759
XOR on page 764

Typing FEDGE evaluates the falling edge of a Boolean expression.
Tip: The IEC 61131-3 standard does not include the FEDGE operator.

FEDGE Operator

| Name | FEDGE |
| :--- | :--- |
| Syntax | <edge> := FEDGE ( <boo_expression>, <memo_variable> ); |
| Operands | First operand is any Boolean variable or complex expression <br> Second operand is an internal Boolean variable used to store the last state of the <br> expression |
| Return value | TRUE when the expression changes from TRUE to FALSE <br> FALSE for all other cases |

An execution cycle detects one falling edge of an expression with the FEDGE operator. Call the FEDGE operator from a function block instead of the F_TRIG function block.

Tip: $\quad$ Trigger edges for an expression with a variable other than the memory Boolean variable that stores the last state of the expression.

When the expression is a Boolean variable, declare a unique internal variable with same name and the EDGE_ prefix for use in the FEDGE expressions for the variable. This internal variable prevents other FEDGE operations from overwriting the memory variable.

## Example

```
(* ST program using FEDGE operator *)
(* this program counts the falling edges of a Boolean
input *)
(* Bi120 is an input Boolean variable *)
(* Edge_Bil20 is the memory of the Bil20 variable state
*)
If FEDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```


## See also

REDGE on page 706
F TRIG on page 813
Typing REDGE evaluates the rising edge of a complete Boolean expression.
Tip: The IEC 61131-3 standard does not include the REDGE operator.

## REDGE Operator

| Name | REDGE |
| :--- | :--- |
| Syntax | <edge> := REDGE ( <boo_expression>,<memo_variable> ); |

## REDGE Operator

| Operands | First operand is any Boolean variable or complex expression <br> Second operand is an internal Boolean variable used to store the last state of the <br> expression |
| :--- | :--- |
| Return value | TRUE when the expression changes from FALSE to TRUE <br> FALSE for all other cases |

An execution cycle detects one rising edge of an expression with the REDGE operator. Call the REDGE operator from a function block instead of the R_TRIG function block.

## Tip: $\quad$ Trigger edges for an expression with a variable other than the memory Boolean variable that stores the last state of the expression.

When the expression is a Boolean variable, declare a unique internal variable with same name and the EDGE_ prefix for use in the REDGE expressions for the variable. This internal variable prevents other REDGE operations from overwriting the memory variable.

## Example

```
(* ST program using REDGE operator *)
(* this program counts the rising edges of a Boolean
input *)
(* Bi120 is an input Boolean variable *)
(* Edge_Bil20 is the memory of the Bi120 variable state
*)
If REDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```


## See also

FEDGE on page 706
R TRIG on page 816

## ST Keyboard Shortcuts

Keyboard shortcuts are available for use with the Structured Text (ST) language. Some shortcuts do not apply or may differ while debugging.

| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+A | Selects the entire document (not available while debugging) |
| Ctrl+C | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+Insert | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+V | Pastes text copied to the clipboard to the insertion point (not available while <br> debugging) |
| Shift+Insert | Pastes text copied to the clipboard to the insertion point (not available while <br> debugging) |
| Ctrl+X | Cuts the selected text to the clipboard (not available while debugging) |
| Shift+Delete | Cuts the selected text to the clipboard (not available while debugging) |
| Ctrl+L | Cuts the current line to the clipboard (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Shift+Z | Redoes the previous command (not available while debugging) |


| Keyboard Shortcut | Description |
| :---: | :---: |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Insert | Toggles between the overwrite/insert typing mode |
| Shift+Enter | Inserts a line break. While debugging, placing the insertion point on a variable opens the Write Logical Value dialog box. |
| Ctrl+Enter | Inserts a line above the current line. While debugging, placing the insertion point on a variable opens the Write Logical Value dialog box. |
| Ctrl+Shift+Enter | Inserts a line below the current line. While debugging, placing the insertion point on a variable opens the Write Logical Value dialog box. |
| Ctrl+Shift+T | Transposes the current and previous word (not available while debugging) |
| Ctrl+Shift+Alt+T | Transposes the current and next line (not available while debugging) |
| Ctrl+Space | Displays a list of available items such as variables, operators, functions, and function blocks. Filter displayed items by typing letters, digits, and underscore characters. (not available while debugging) |
| Ctrl+Shift+Space | Displays a list of available items such as variables, operators, functions, and function blocks. Filter displayed items by typing letters, digits, and underscore characters. (not available while debugging) |
| Ctrl+Shift+U | Changes the selected text into uppercase (not available while debugging) |
| Ctrl+U | Changes the selected text into lowercase (not available while debugging) |
| Up Arrow | Positions the insertion point in the previous line. |
| Down Arrow | Positions the insertion point in the next line. |
| Left Arrow | Positions the insertion point before the previous character. |
| Right Arrow | Positions the insertion point after the next character |
| Ctrl+Left Arrow | Positions the insertion point on the previous statement or word |
| Ctrl+Right Arrow | Positions the insertion point on the next statement or word |
| Home | Positions the insertion point at the beginning of the line |
| End | Positions the insertion point at the end of the line |
| Ctrl+Home | Positions the insertion point at the beginning of the document |
| Ctrl+End | Positions the insertion point at the end of the document |
| Page Up | Positions the insertion point at the top of the of the visible code |
| Page Down | Positions the insertion point at the bottom of the page of the visible code |
| Ctrl+Page Up | Positions the insertion point at the beginning of the code |
| Ctrl+Page Down | Positions the insertion point at the end of the code |
| Ctrl+Up Arrow | Scrolls up |
| Ctrl+Down Arrow | Scrolls down |
| Shift+Up Arrow | Selects characters from the insertion point to the previous line |
| Shift+Down Arrow | Selects characters from the insertion point to the next line |
| Shift+Left Arrow | Selects characters to the left |
| Shift+Right Arrow | Selects characters to the right |
| Ctrl+Shift+Left Arrow | Selects to the previous statement or word |
| Ctrl+Shift+Right Arrow | Selects to the next statement or word |
| Shift+Home | Selects from the insertion point to the start of the line |
| Shift+End | Selects from the insertion point to the end of the line |
| Ctrl+Shift+Home | Selects from the insertion point to the beginning of the document |
| Ctrl+Shift+End | Selects from the insertion point to the end of the document |
| Ctrl+Shift+Page Up | Selects from the insertion point to the top of the visible code |
| Ctrl+Shift+Page Down | Selects from the insertion point to the end of the visible code |
| Ctrl+Shift+W | Selects the next word |
| Shift+Alt+Up Arrow | Selects the current and previous lines |
| Shift+Alt+Down Arrow | Selects the current and next lines |
| Shift+Alt+Left Arrow | Selects the previous character on the current line |
| Shift+Alt+Right Arrow | Selects the next character on the current line |
| Escape | Deselects the selected text |


| Keyboard Shortcut | Description |
| :--- | :--- |
| Ctrl+l | Opens the Variable Selector. (Design mode only) |
| Ctrl+Shift+l | Opens the Variable Selector. (Design mode only) |
| Ctrl+R | Opens the Block Selector. During debugging, selects the variable with the <br> insertion point. |
| Ctrl+Alt+R | Opens the Block Selector. During debugging, selects the variable with the <br> insertion point. |
| Ctrl+Shift+Alt+R | Opens the Block Selector. During debugging, selects the variable with the <br> insertion point. |
| Delete | Removes the next character (not available while debugging) |
| Ctrl+Shift+L | Removes the current line (not available while debugging) |
| Ctrl+Delete | Removes the next word in the current line (not available while debugging) |
| Ctrl+Backspace | Removes the previous word in the current line (not available while <br> debugging) |
| Backspace | Removes the previous character (not available while debugging) |
| Shift+Backspace | Removes the previous character (not available while debugging) |

## See also

## ST Main Syntax on page 692

## ST Basic Elements and Statements on page 697

Create graphical interfaces or ISaVIEW screens to monitor or run control processes on local computers or remote locations that use Internet or network connections. Add ISaVIEW screens to projects at the device and program level.

Create and develop ISaVIEW screens while editing a project or running online (simulation or debugging). To develop ISaVIEW screens, insert graphic objects and define animation behaviors for execution at run-time.

Create ISaVIEW screens from blank documents or from a template. Insert objects available from the Toolbox. Modify object properties to define animation effects. Also group objects together, then define animation effects for the group.

While running online, switch between design mode and animation mode. Design mode to edit objects contained in screens. Animation mode launches the execution of animation effects defined for objects contained in screens.

From the ISaVIEW toolbar, perform these operations:

| Operation | Description |
| :--- | :--- |
| a | Design Mode, edit objects contained in screens while connected (simulation or online) |
| Animation Mode, launch the execution of animation effects defined for objects contained in |  |
| screens while connected (simulation or online) |  |

## See also

Create an ISaVIEW screen from a blank document on page 710
Create an ISaVIEW screen from a template on page 710
Export an ISaVIEW screen as a template on page 711
Insert Objects on page 711
Define Animation Effects for Objects on page 725

## Create an ISaVIEW screen from a blank document

Create an ISaVIEW screen from a template

Create an ISaVIEW screen from a blank document, then insert objects from the Toolbox and define animation effects. Export screens as templates for reuse.

## To create an ISaVIEW screen from a blank document

1. From the Application View, right-click the controller or program, point to Add, and then click New ISaVIEW.
2. Double-click the added ISaVIEW screen.

The ISaVIEW screen appears.
3. Insert objects and define animation effects.

## See also

ISaVIEW Screens on page 709
Export an ISaVIEW screen as a template on page 711
Insert Objects on page 711
Edit Objects on page 721
Define Animation Effects for Objects on page 725
Create an ISaVIEW screen from an existing ISaVIEW template. ISaVIEW templates have the ${ }^{*}$.hmi extension.

## To create an ISaVIEW screen from a template

1. From the Application View, right-click the controller or program, point to Add, and then click ISaVIEW from Template
2. In the Select the ISaVIEW Template dialog box, browse to locate the required template, and then click Open.
3. In the Import ISaVIEW Template dialog box, specify a screen name and associate the required variables where required, and then click OK.
4. Double-click the imported ISaVIEW screen.

The ISaVIEW screen appears.
5. Edit and insert objects, and then define animation effects.

## See also

ISaVIEW Screens on page 709

## Export an ISaVIEW screen as a template on page 711

Insert Objects on page 711

## Edit Objects on page 721

Define Animation Effects for Objects on page 725

## Export an ISaVIEW screen as a template

Develop an ISaVIEW screen then export the screen for use as a template with the *.hmi extension. The default names of templates are the same as the screen names. When adding an ISaVIEW screen from a template, choose from the available templates. ISaVIEW templates are stored here:
\%USERPROFILE\%\AppData\Local\Rockwell Automation\SIS Workstation <version>\ACP\Templates\Isaview

## To export an ISaVIEW screen as template

1. Develop an ISaVIEW screen.
2. From the Application View, right-click the ISaVIEW item, and then click Export as Template.
3. In the Export ISaVIEW Template dialog box, type the template name, and then click OK.

## See also

ISaVIEW Screens on page 709
Create an ISaVIEW screen from a blank document on page 710
Insert Objects on page 711
Define Animation Effects for Objects on page 725
Insert these objects into an ISaVIEW screen from the Toolbox.

- Arc
- Arrow
- Ellipse
- Rectangle
- Rounded Rectangle
- Triangle
- Image
- Web Container
- Button
- Edit Box
- Gauge
- Slider
- Line
- Bar Meter
- Polygon

Overlap or superimpose objects and groups of objects. Use contextual menu options to group objects and move objects to the front or back.

Tip: Web containers always remain on top of other objects.

By default, only the basic properties display in the Properties window for ISaVIEW objects.

## To display the extended properties for an ISaVIEW object

1. From the View menu, click Properties Window.
2. In the Properties window, click .

## See also

Define Animation Effects for Objects on page 725

## Edit Objects on page 721

Preview ISaVIEW Screens on page 732
An arc is any unbroken part of the circumference of a circle. An arc represents, for example, a container that displays a changing quantity of liquid as it flows to or from another object. An arc object consists of a starting angle and an angle length:


Define the properties for the arc object from the Properties window. For the arc object, define properties for Action, Color, Displacement, Size, Text, and Visibility. The arc object specific properties are:

| Property | Description |
| :--- | :--- |
| Angle Length | Length of the arc in degrees |
| Angle Start | Size of the angle prior to the start of the arc |

## To insert an arc

- From the Toolbox, drag the arc object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
The arrow object is a directional shape with a rectangular shaft and triangular head. Define the properties for the arrow object from the Properties window.

For the arrow object, define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The arrow also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Define fill color for objects in the Color properties. |  |

## To insert an arrow

- From the Toolbox, drag the arrow object into the workspace.


## See also

## Define Animation Effects for Objects on page 725

Insert Objects on page 711
An ellipse represents items like a container that displays a changing quantity of liquid as it flows to or from another object. Define the properties for the ellipse object from the Properties window. For the ellipse object, define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The ellipse also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Define fill color for objects in the Color properties. |  |

## To insert an ellipse

- From the Toolbox, drag the ellipse object into the workspace.


## See also

## Define Animation Effects for Objects on page 725

Insert Objects on page 711
Rectangle
A rectangle represents, for example, pipes that indicate a flow from one object to another with a change of color or a container that displays a changing quantity of liquid as it flows to or from another object. Define properties for the rectangle object from the Properties window. For the rectangle object, define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The rectangle also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |

Define fill color for objects in the Color properties.

## To insert a rectangle

- From the Toolbox, drag the rectangle object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711

## Rounded Rectangle

## Triangle

A rounded rectangle is a rectangular shape with rounded corners. Define the properties for the rounded rectangle object from the Properties window. For the rounded rectangle object, define properties for Action, Color, Displacement, Size, Text, and Visibility. The rounded rectangle also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Corner Radius | Radius of the corners for a rounded rectangle. Possible values are literal values. |
| Define fill color for objects in the Color properties. |  |

## To insert a rounded rectangle

- From the Toolbox, drag the rounded rectangle object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
A triangle object is a triangular shape. Define properties for the triangle object from the Properties window. For the triangle object, define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The triangle also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Define fill color for objects in the Color properties. |  |

## To insert a triangle

- From the Toolbox, drag the triangle object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711

The image object holds file formats such as GIF, JPEG, BMP, PNG, and TIFF. Image objects cannot be rotated.

Define the properties for the image object from the Properties window. For the image object, define properties for Action, Displacement, Size, Text, and Visibility. The image object also has the Image Path property.

| Property | Description |
| :--- | :--- |
| Image Path | Path to the image to display |

## To insert an image

- From the Toolbox, drag the image object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
The web container object has a rectangular shape. Define properties for the web container object from the Properties window. For the web container object, define properties for Action, Displacement, Size, Text, and Visibility. The web container object also has frame and object-specific properties:

| Property | Description |
| :--- | :--- |
| Link Page | Target URI to display in the object |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |

The web container always remains on top of other objects in a screen.

## To insert a web container

- From the Toolbox, drag the web container object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711

## Button

Edit Box

## Gauge

The button object displays text and has a rectangular shape. Define the properties for the button object from the Properties window. For the button object, define properties for Action, Color, Displacement, Size, Text, and Visibility.

## To insert a button

- From the Toolbox, drag the button object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
The edit box object enables to display and enter text and has a rectangular shape. Define the properties for the edit box object from the Properties window. For the edit box object, define properties for Action, Color, Displacement, Size, Text, and Visibility. Also choose to display a border to outline the edit box:

| Property | Description |
| :--- | :--- |
| Border | The object has a border. Possible values are True or False. |

## To insert an edit box

- From the Toolbox, drag the edit box object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
The gauge object is a circular dial with a needle and range of values that represent a traditional meter or dial. The gauge's needle moves around the dial indicating the changing value.

Define the properties for the gauge object from the Properties window. For the gauge object, define properties for Action, Color, Displacement, Size, Text, and Visibility. The gauge object also has frame and object-specific properties:

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system <br> colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Background Shape | Shape of the measuring object. Possible shapes are rectangle, ellipse, hexagon, <br> and octagon. |
| Indicator Color | Color for the interior of the indicator. Possible colors are custom, web, and <br> system colors. |

Chapter 3
Working with Trusted Applications

| Property | Description |
| :---: | :---: |
| Indicator Constant Length | The indicator maintains the same length when traveling along the scale of the measuring object. Possible values are True or False. |
| Indicator Frame Color | Color for the outline of the indicator. Possible colors are custom, web, and system colors. |
| Indicator Frame Width | Width of the outline for the indicator. Possible values are literal values. |
| Indicator Thickness | Width of the indicator. Possible values are literal values. |
| Indicator Value | Initial value of the measuring object. Possible values are literal values. |
| Indicator Value Variable | Variable that controls the indicator of the measuring object. Possible variable data types are DINT or REAL. Click ... to select a variable. |
| Margin Bottom Right | Margin from the gauge dial to the bottom and right sides of the object perimeter. Possible values are literal values. |
| Margin Top Left | Margin from the gauge dial to the top and left sides of the object perimeter. Possible values are literal values. |
| Maximum | Maximum value of the scale on the measuring object. Possible values are literal values. |
| Minimum | Minimum value of the scale on the measuring object. Possible values are literal values. |
| Scale Frame Color | Color of the scale on the measuring object. Possible colors are custom, web, and system colors. |
| Scale Frame Width | Width of the scale on the measuring object. Possible values are literal values. |
| Scale Label Distance | Distance between the scale on the measuring object and the displayed range values, in pixels. Possible values are literal values. |
| Scale Label Frequency | Frequency of labeling of major divisions on the scale of the measuring object. For example, a value of two (2) results in labeling every second major division. Possible values are literal values. |
| Scale Label Style | Location of the displayed labels in reference to the circular scale. Possible values are Left, Right, AlternateStartLeft, and AlternateStartRight. Setting labels on the left places these on the outside of the scale while labels on the right places these on the inside of the scale. Alternate starts places the lowest range label respectively then every other label on alternating sides of the scale. |
| Scale Label Text Bold | Apply the bold style to the label text. Possible values are True or False. |
| Scale Label Text Color | Color of the label text. Possible colors are custom, web, and system colors. |
| Scale Label Text Size | Size of the label text. Possible values are literal values. |
| Scale Start Angle | Angle at which the circular scale starts in reference to the $x$-axis. For example, a start angle of $0^{\circ}$ places the beginning of the scale on the positive $x$-axis. Possible values are 0 to 360 . |
| Scale Sweep Angle | Span of the circular scale. For example, a sweep angle of $180^{\circ}$ indicates a semicircular scale. Possible values are 0 to 360 . |
| Scale Tick Major Frequency | Frequency of major ticks in reference to minor ticks on the scale. For example, on a scale that ranges from 1 to 100 with a Tick Unit value of 5 , a major tick frequency setting of 5 sets a major division at every 5 th minor division. The major division is each increment of 25 . The major division is each increment of 25. Possible values are literal values. |
| Scale Tick Major Width | Width of the major tick marks to divide the scale. Possible values are literal values. |
| Scale Tick Width | Width of the minor tick marks to divide the scale. Possible values are literal values. |
| Tick Color | Color of the ticks that divide the scale. Possible colors are custom, web, and system colors. |
| Tick Unit | Value associated to individual tick divisions on the measuring scale. Possible values are literal values. |

## To insert a gauge

- From the Toolbox, drag the gauge object into the workspace.


## See also

Define Animation Effects for Objects on page 725

## Edit Objects on page 721

## Insert Objects on page 711

The slider object reads the position of the indicator within its perimeter then sends a value associated to the position to mapped variables. At run time, the bar meter is not visible. Increase or decrease the number of horizontal and vertical divisions within the slider to define the accuracy of position readings.

Define the properties for the slider object from the Properties window. For the slider object, define properties for Action, Color, Displacement, Size, Text, and Visibility. Other slider properties include:

| Property | Description |
| :---: | :---: |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Indicator Color | Color for the interior of the indicator. Possible colors are custom, web, and system colors. |
| Indicator Dimensions | The length and width of the indicator, in pixels. Possible values are literal values. |
| Indicator Frame Color | Color for the outline of the indicator. Possible colors are custom, web, and system colors. |
| Indicator Style | Shape of the indicator. Possible shapes are bar and triangles. |
| Indicator Value | Initial value of the measuring object. Possible values are literal values. |
| Indicator Value Variable | Variable that controls the indicator of the measuring object. Possible variable data types are DINT or REAL. Click ... to select a variable. |
| Maximum | Maximum value of the scale on the measuring object. Possible values are literal values. |
| Minimum | Minimum value of the scale on the measuring object. Possible values are literal values. |
| Orientation | The orientation of the measuring object is horizontal or vertical |
| Scale Frame Color | Color of the scale on the measuring object. Possible colors are custom, web, and system colors. |
| Scale Frame Width | Width of the scale on the measuring object. Possible values are literal values. |
| Scale Label Distance | Distance between the scale on the measuring object and the displayed range values, in pixels. Possible values are literal values. |
| Scale Label Frequency | Frequency of labeling of major divisions on the scale of the measuring object. For example, a value of two (2) results in labeling every second major division. Possible values are literal values. |
| Scale Label Text Bold | Apply the bold style to the label text. Possible values are True or False. |
| Scale Label Text Color | Color of the label text. Possible colors are custom, web, and system colors. |
| Scale Label Text Size | Size of the label text. Possible values are literal values. |
| Scale Tick Major Frequency | Frequency of major ticks in reference to minor ticks on the scale. For example, on a scale that ranges from 1 to 100 with a Tick Unit value of 5 , a major tick frequency setting of 5 sets a major division at every 5 th minor division. For example, the major division is each increment of 25. Possible values are literal values. |
| Scale Tick Major Width | Width of the major tick marks to divide the scale. Possible values are literal values. |
| Scale Tick Width | Width of the minor tick marks to divide the scale. Possible values are literal values. |
| Tick Color | Color of the ticks that divide the scale. Possible colors are custom, web, and system colors. |


| Property | Description |
| :--- | :--- |
| Tick Unit | Value associated to individual tick divisions on the measuring scale. Possible <br> values are literal values. |

## To insert a slider

- From the Toolbox, drag the slider object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Edit Objects on page 721
Insert Objects on page 711
The line object is a unbroken linear shape. Define the properties for the line object from the Properties window. For the line object, define properties for Action, Displacement, Rotation, Size, and Visibility. The line also has color and width properties.

| Property | Description |
| :--- | :--- |
| Line Color | Color for the line object. Possible colors are custom, web, and system colors. |
| Line Width | Width of the line object. Possible values are literal values. |

## To insert a line

- From the Toolbox, drag the line object into the workspace.


## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711

## Bar Meter

The bar meter object reads the position of the indicator within its perimeter then sends a value associated to the position to mapped variables. At run time, the bar meter is not visible. Increase or decrease the number of divisions within the bar meter to define the accuracy of position readings.

Define the properties for the bar meter object from the Properties window. For the bar meter object, define properties for Action, Color, Displacement, Size, Text, and Visibility. Also define properties for the frame color and width. Specific bar meter object properties are:

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, <br> and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |
| Indicator Color | Color for the interior of the indicator. Possible colors are custom, <br> web, and system colors. |


| Indicator Frame Color | Color for the outline of the indicator. Possible colors are custom, web, <br> and system colors. |
| :--- | :--- |
| Indicator Value | Initial value of the measuring object. Possible values are literal <br> values. |
| Indicator Value Variable | Variable that controls the indicator of the measuring object. Possible <br> variable data types are DINT or REAL. Click ... to select a variable. |
| Maximum | Maximum value of the scale on the measuring object. Possible values <br> are literal values. |
| Minimum | Minimum value of the scale on the measuring object. Possible values <br> are literal values. |
| Orientation | Orientation of the measuring object is horizontal or vertical |
| Scale Frame Color | Color of the scale on the measuring object. Possible colors are <br> custom, web, and system colors. |
| Scale Frame Width | Width of the scale on the measuring object. Possible values are literal <br> values. |
| Scale Label Distance | Distance between the scale on the measuring object and the <br> displayed range values. Possible values are literal values. |
| Scale Label Frequency | Frequency of labeling of major divisions on the scale of the <br> measuring object, in pixels. For example, a value of two (2) results in a <br> label for every second major division. Possible values are literal <br> values. |
| Scale Label Text Bold | Apply the bold style is applied to the label text. Possible values are <br> True or False. |
| Scale Label Text Color | Color of the label text. Possible colors are custom, web, and system <br> colors. |
| Scale Label Text Size | Size of the label text. Possible values are literal values. |
| Scale Tick Major Frequency | Frequency of major ticks in reference to minor ticks on the scale. <br> Example: on a scale that ranges from 1 to 100 with a Tick Unit value of <br> $5, ~ s e t t i n g ~ 5 ~ a s ~ t h e ~ m a j o r ~ t i c k ~ f r e q u e n c y ~ s e t s ~ a ~ m a j o r ~ d i v i s i o n ~ a t ~ e v e r y ~$ |
| 5th minor division. The major division is each increment of 25. |  |
| Possible values are literal values. |  |

## To insert a bar meter

- From the Toolbox, drag the bar meter object into the workspace.


## See also

Define Animation Effects for Objects on page 725

## Edit Objects on page 721

## Insert Objects on page 711

A polygon object consists of three or more connected straight lines to form a closed figure. Each line is a segment. Define the properties for the polygon object from the Properties window. For the polygon object, define properties
for Action, Color, Displacement, Size, Text, and Visibility. The polygon also has frame color and width properties.

| Property | Description |
| :--- | :--- |
| Frame Color | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| Frame Width | Width of the frame for the object. Possible values are literal values. |

Define fill color for objects in the Color properties.
For polygon objects, establish the end of each segment that forms the shape, then establish the end of the shape upon completion of all segments. Establish the end of a segment.

Use keyboard commands with polygon objects: press Ctrl+Z to undo up to the first segment and press Escape to delete a polygon object in progress.

## To insert a polygon

1. From the Toolbox, drag the polygon object into the workspace.
2. Click and drag to define each segment that forms the shape, then press Enter to complete the shape.
3. To cancel the shape, press Escape.

## See also

Define Animation Effects for Objects on page 725
Insert Objects on page 711
Perform many edition tasks on objects:

- Select Objects
- Edit the Properties of Objects
- Cut, Copy, and Paste Objects
- Delete Objects
- Move Objects
- Resize Objects
- Group Objects
- Align Objects
- Move Objects to the Front and Back

Edit objects in design mode.

## See also

Select Objects on page 722
Edit the Properties of Objects on page 722
Delete Objects on page 723
Align Objects on page 724
Move Objects to the Front and Back on page 725

Select Objects
Select one or more objects to edit functions. Drag selected objects with the mouse to move these in the workspace.

| Task | Procedure |
| :--- | :--- |
| Select a single object | - In the workspace, click the object. |
| Select multiple objects | Drag to enclose multiple objects or select individual objects. An invisible rectangle encloses the area when dragging. <br> - Position the cursor to the left and above the objects, then drag to enclose them. <br> - Hold down the Ctrl key and click the objects one after the other. Click a selected object to deselect the object while all others <br> remain selected. <br> - Hold down the Shift key and click the objects one after the other. |
| Deselect objects | - Click on an empty space in the workspace. <br>  |

## See also

Edit the Properties of Objects on page 722
Cut, Copy, and Paste Objects on page 722
Move Objects on page 723
Resize Objects on page 723
Align Objects on page 724

## Edit the Properties of Objects

Change the properties of objects in the Properties window.

## To edit the properties of objects

1. Select the object.
2. In the Properties window, enter the information for the individual properties.

## See also

Insert Objects on page 711
Resize Objects on page 723
Select Objects on page 722
Cut, copy, and paste objects in screens.
Cut, Copy, and Paste Objects

| Task | Procedure |
| :---: | :---: |
| Cut an object | 1. Select the object. <br> 2. Do one of these: <br> - Right-click, then click Cut. <br> - From the Edit menu, click Cut. |
| Copy an object | 1. Select the object. <br> 2. Do one of these: <br> - Right-click, then click Copy. <br> - From the Edit menu, click Copy. <br> - Press the Ctrl key and drag the object. |


| Task | Procedure |
| :--- | :--- |
| Paste an object | • In the workspace, click where to insert the object, then do one of the following: <br>  <br>  <br>  <br>  <br>  <br>  <br>  <br> • Right-click, then click Paste. |

## See also

Delete Objects on page 723
Move Objects on page 723
Move Objects to the Front and Back on page 725
Select Objects on page 722

Delete Objects

Move Objects

## See also

Cut, Copy, and Paste Objects on page 722
Group Objects on page 724
Move Objects to the Front and Back on page 725
Select Objects on page 722
Move objects within the screen.

## To move objects

- Select one or more objects then drag to their new position.


## See also

Move Objects to the Front and Back on page 725
Align Objects on page 724
Group Objects on page 724
Select Objects on page 722
Resize objects in screens.

## To delete objects

1. Select the object.
2. Do one of these:

- Right-click, then click Delete.
- From the Edit menu, click Delete.
.


## To resize objects

1. Select the object to resize.
2. Click a handle (small square on the outer edge of the selected object) then move it in the appropriate direction.

## See also

Select Objects on page 722
Edit the Properties of Objects on page 722

## Group Objects on page 724

## Group Objects

Group individual objects in a screen to form a unique object. Also group individual groups of objects. When objects are grouped, individual objects in the group cannot be resized, moved, deleted, or copied. Change the properties of individual objects belonging to a group and those properties of grouped objects.

Apply action, size, and visibility animation effects to grouped objects other than those effects attached to the grouped items.
Once objects are selected, choose to group and ungroup using commands from the contextual menu or icons from the toolbar.

| Task | Procedure |
| :--- | :--- |
| Group objects | 1. Select the required objects. |
|  | 2. Do one of these: |
|  | • Right-click, then click Group Items. |
|  | • From the ISaVIEW toolbar, click $\bullet$ |
| Ungroup objects | 1. Slect the grouped object. |
|  | 2. Do one of these: |
|  | • Right-click, then click Ungroup Items. |
|  | • From the ISaVIEW toollar, click $\quad$. |

## See also

Move Objects on page 723
Resize Objects on page 723
Align Objects on page 724
Move Objects to the Front and Back on page 725
Select Objects on page 722

## Align Objects

Align objects relative to their left side, right side, top edge, or bottom edge. Elements align relative to the first element selected.

Once objects are selected, choose to align them with commands from the Layout menu available from the contextual menu or use the arrow keys.

## To align and position objects

1. Select the objects to align starting with the element to use as reference for the alignment.
2. Right-click the elements, point to Layout, then click the required alignment command:

- Align Left
- Align Center
- Align Right
- Align Top
- Align Middle
- Align Bottom

The objects are aligned in the selected direction in reference to the first selected item.

## See also

Move Objects on page 723
Move Objects to the Front and Back on page 725
Group Objects on page 724
Select Objects on page 722

## Move Objects to the Front and Back

Move objects to the front or to the back of each other. Move these using the commands from the contextual menu after selecting objects.

| Task | Procedure |
| :--- | :--- |
| Bring an object to the front | 1. Select the object. <br> 2. Right-click the element, then click Bring to Front. |
| Send an object to the back | 1. Select the object. <br> 2. Right-click the element, then click Send to Back. |

## See also

Move Objects on page 723
Group Objects on page 724
Align Objects on page 724
Select Objects on page 722
Define Animation Effects for Objects

Define animation of these effects for objects or groups of objects defined in ISaVIEW screens:

- Action
- Color
- Displacement
- Rotation
- Size
- Text
- Visibility

Set property values from the Properties window to define animation effects. Also switch to the animation preview mode to graphically modify the rotation, displacement, and size properties.

All global and local variables are available to set the properties in the Properties window. The Collection Editor is available to define the color property Fill Color Phase and the text property Text Color Phase. Use the Collection Editor to create and edit the members of a collection and to define the colors (Phase Colors) and numerical values (Phase Maximum and Phase Minimum) for each member. When online, the object or object text displays the color that corresponds to its current value as defined in the collection. Example: for the color black, assign a value of 10 to Phase Maximum and a value of 0 to Phase Minimum displays the object or object text as black when its value is between 0 and 10 .

## See also

Insert Objects on page 711
Edit Objects on page 721
Preview ISaVIEW Screens on page 732
Any ISaVIEW object or group acts as a push button. Use the styles and variables of the action properties to define a push button-like behavior for the object. Define the action properties using the Properties window. Properties available to define the action of an object are:

| Action Property | Description |
| :--- | :--- |
| Action Event | Operation to perform upon occurrence of Action Type event. Possible <br> values are None, GoToHTML, GoToPage, IncrementValue, <br> AutolncrementValue, and ReverseValue. |
| Action Link | Destination address or path for GoToHTML or GoToPage Action Event <br> operations. Possible values are ftp://, http://www, and II. |
| Action Type | Mouse event triggering the Action Event operation. Possible values are <br> None, MouseClick, MouseDoubleClick, and MouseAll. |
| Action Variable | Variable that controls the Action Event for IncrementValue, <br> AutolncrementValue, and ReverseValue operations. The Action Event <br> Type determines the expected variable type: <br> $\bullet$ IncrementValue: any integer and any real <br> - AutoIncrementValue: any integer and any real <br> - ReverseValue: Boolean, any integer and any real <br> Click ... to select a variable. |
| Increment Time | Interval between increments, in seconds, of the Action Variable variable <br> where the Action Event is AutolncrementValue |
| Increment Value | Rate of increase of the Action Variable variable for each Action Type <br> mouse event where the Action Event is either IncrementValue or <br> AutolncrementValue |


| ActionEvent Operations | Description |
| :--- | :--- |
| None | Disables Action Event |
| GoToHTML | Jumps to the HTML page defined in Action Link |


| ActionEvent Operations | Description |
| :--- | :--- |
| GoToPage | Jumps to the ISaVIEW page defined in Action Link |
| IncrementValue | Increments once the value of the Action Variable variable by the value of <br> Increment Value |
| AutoIncrementValue | Increments continuously the Action Variable variable by the Increment <br> Value value using the Increment Time time lapse |
| Reverse Value | Reverses the value of the Action Variable variable |


| Mouse Event | Description |
| :--- | :--- |
| None | Disables Action Type |
| MouseClick | Sets a single mouse click to execute Action Event |
| MouseDoubleClick | Sets a double mouse click to execute Action Event |
| MouseAll | Sets any type of mouse click to execute Action Event |

Define action properties for an object from the Properties window while in design mode.

## To define the action properties of an object

1. Click to set the ISaVIEW screen to design mode.
2. In the ISaVIEW screen, select the required object or group of objects.
3. In the Properties window, define the required action properties.

## See also

Insert Objects on page 711
ISaVIEW Screens on page 709
Define Animation Effects for Objects on page 725
Rotation on page 729
Size on page 730
Define color properties for these objects: arcs, arrows, ellipses, rectangles, rounded rectangles, triangles, buttons, edit boxes, gauges, sliders, and polygons. Define the color properties using the Properties window. The properties available for defining the color of an object are:

| Property | Description |
| :--- | :--- |
| Color Variable | Variable defining the phase value during animation mode. Possible variable data <br> type is DINT. Click ... to select a variable. |
| Initial Color | (Read only) Initial color of the object, while in design mode |
| Fill Color | Actual color of the object. Equal to InitialColor while in design mode. Possible <br> Colors are custom, web, and system colors. |
| Fill Color Phase | List of colors to apply during phases while in animation mode. Click ... to access <br> the phase collection editor. |
| Fill Foreground Color | Contrast color used for Fill Style. Possible colors are custom, web, and system <br> colors. Available for all objects except edit boxes. |

Fill Style
Style applied to the coloring of an object such as a gradient, texture, or hatch line. Available for all objects except edit boxes. Fill Foreground Color provides the contrast color used in the style. Possible styles are available from a drop-down combo-box.

Define color properties for an object from the Properties window while in design mode.

## To define the color properties of an object

1. Click to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. In the Properties window, define the required color properties.

## See also

Collection Editor on page 728
Text on page 731
Visibility on page 732
ISaVIEW Screens on page 709
Define Animation Effects for Objects on page 725

## Collection Editor

## Displacement

## To access a collection editor

From the Properties window, in the Fill Color Phase property for an ISaVIEW object, click ...

Use the Collection Editor to create and edit individual members of collections. The available edition properties depend on the collection. The Collection Editor has a members list and a properties grid. Perform these tasks in the Collection Editor:

- Add members to the list. Select a member, then click Add. The first click adds an initial member.
- Remove members from the list. Select a member, then click Remove.
- Reorder members in the list. Select the member, then click the up or down arrows.
- Edit the properties of a member. Select the member, then edit its properties in the grid.


## See also

Color on page 727
Define displacement properties for all ISaVIEW objects. Before displacement occurs, the coordinates of the upper left corner of the object define the start position. Use the displacement properties to define the linear movement of the object when in animation mode. Define displacement properties from the

Properties window. Also, define the Animation Position property within the workspace. These properties define the displacement of an object:

| Property | Description |
| :--- | :--- |
| Animation Position | Destination coordinates after displacement during animation mode in <br> reference to the top left corner of the object bounding box, in pixels <br> (design mode displays InitialPosition coordinates). |
| Displacement Variable | Variable that controls the object displacement. Possible variable data <br> types are DINT or STACKINT. Click ... to select a variable. |
| Initial Position | (Read only) Coordinates of the object prior to displacement. |
| Location | Actual coordinates of the object. Equal to Initial Position while in design <br> mode. |
| Maximum Displacement | Maximum amount of displacement during animation mode. The default <br> value is 1000. |
| Minimum Displacement | Minimum amount of displacement during animation mode. The default <br> value is 0. |

In animation mode, the final position of the object is defined as:

```
Initial Position + (Animation Position - Initial Position)
* [(Displacement Variable - Minimum Displacement) /
(Maximum Displacement - Minimum Displacement)]
```

Define displacement properties for an object from the Properties window while in design mode.

## To define the displacement properties of an object

1. Click $\mathbb{Z}$ to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. In the Properties window, define the required displacement properties.

## See also

Rotation on page 729
Size on page 730
Action on page 726
ISaVIEW Screens on page 709
Define Animation Effects for Objects on page 725

## Rotation

Define the rotation properties for these ISaVIEW objects: arrows, ellipses, rectangles, triangles, and lines. Use the rotation properties to define the rotation of the object when in animation mode. Define rotation properties in the Properties window. Also define the Center of Rotation property within the workspace. These properties define the rotation of an object:

| Property | Description |
| :--- | :--- |
| Center of Rotation | Coordinates of the center of rotation for the object in reference to the top left <br> corner of the object bounding box. |
| Rotation Variable | Variable controlling the object rotation. Possible variable data types are DINT or <br> STACKINT. Click ... to select a variable. |


| Property | Description |
| :--- | :--- |
| Maximum Rotation | Maximum range of rotation of the object, in degrees. Possible values are <br> positive or negative; The default value is 360 degrees. |
| Minimum Rotation | Minimum range of rotation of the object, in degrees. Possible values are positive <br> or negative; The default value is 0 degrees. |
| Static Angle | Initial angle in reference to the right side of the base of the object. Possible <br> values are 0 to 360. |
| The final rotation of an object is defined as: |  |
| \{[ (Rotation Variable - Minimum Rotation) * 360]/ (Maximum |  |
| Rotation - Minimum Rotation) \} \%360 |  |

## To define the rotation properties of an object

1. Click to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. In the Properties window, define the required rotation properties.

## See also

Size on page 730
Text on page 731
Visibility on page 732
ISaVIEW Screens on page 709
Define Animation Effects for Objects on page 725

## Size

Modify the size of all ISaVIEW objects. Define size properties in the Properties window. Also, define the AnimationSize property within the workspace. Properties available for defining the size of an object are:

| Property | Description |
| :--- | :--- |
| Animation Size | Maximum enlargement of the object in percentage (\%). This value must be at least <br> $100 \%$. |
| Size Variable | Variable controlling the resizing of the object. Possible variable data types are DINT or <br> STACKINT. Click $\ldots$ to select a variable. |
| Initial Size | (Read only) The width and height of the object prior to resize occurs in animation <br> mode |
| Maximum Size | Value used by Size Variable to define the maximum range of enlargement for the <br> object. Possible values are positive or negative and must be greater than Minimum <br> Size; the default value is 100. |
| Minimum Size | Value used by Size Variable to define the minimum range of enlargement for the <br> object. Possible values are positive or negative and must be less than Maximum Size; <br> the default value is 0. |
| Size | Actual size of the object whether in design or animation mode |

## To define the size properties of an object

1. Click to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. From the Properties window, define the required size properties.

## See also

Text on page 731
Visibility on page 732
Action on page 726
ISaVIEW Screens on page 709
Define Animation Effects for Objects on page 725
Define text properties for these objects: arcs, arrows, ellipses, rectangles, rounded rectangles, triangles, buttons, edit boxes, web containers, gauges, sliders, bar meters, and polygons. However, for web containers, only the Text text property is available; all other text properties are unavailable for this object. Define text properties in the Properties window.

Properties that define the appearance of the text associated with objects:

| Property | Description |
| :--- | :--- |
| Initial Text | (Read only) The text prior to animation mode. Equal to Text while in design mode. <br> Available for all objects except web containers. |
| Initial Text Color | (Read only) The text color prior to animation mode. Equal to Text Color while in <br> design mode. Available for all objects except web containers. |
| Text | Actual text displayed on the object whether in design or animation mode |
| Text Color | Actual text color whether in design or animation mode. Possible colors are <br> custom, web, and system colors. Available for all objects except web containers. |
| Text Color Phase | List of colors to apply to displayed text during phases while in animation mode. <br> Click ... to access the phase collection editor. Available for all objects except <br> web containers. |
| Text Color Variable | Variable to control the text color. The possible variable data type is DINT. Click <br> .... to select a variable. Available for all objects except web containers. |
| Text Size | Size of the text displayed on the object. Possible values are literal values. <br> Available for all objects except web containers. |
| Text Variable | Variable to control the text displayed on the object. All variable data types are <br> possible. Click $\ldots$ to select a variable. Available for all objects except web <br> containers. |

## To define the text properties of an object

1. Click to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. From the Properties window, define the required text properties.

## See also

Visibility on page 732
Size on page 730
Rotation on page 729
Displacement on page 728
ISaVIEW Screens on page 709

## Visibility

## To define the visibility property of an object

1. Click to set the ISaVIEW screen to design mode.
2. Select the required object or group of objects.
3. From the Properties window, define the visibility property.

## See also

Text on page 731
Size on page 730
Rotation on page 729
Displacement on page 728
When visualizing ISaVIEW screens, choose to display different graphic views of objects and their properties:

## Preview ISaVIEW Screens

visibility property in the Properties window. This property defines the visibility of an object:

## Visibility Variable

 BOOL. Click ... to select a variable.| Icon | Description | Procedure |
| :--- | :--- | :--- |
| a | No Preview, displays the objects defined in a screen where selecting an <br> object exposes bounding box and dimension lines for the object. Modify, <br> add, delete, move, group, or ungroup objects. | • From the ISaVIEW toolbar, click |

Previewing screens is available while online.

## See also

Preview Selections on page 732
Preview Animation Effects (Editable) on page 733

## Preview Selections

Preview selections to modify objects and visualize the rotation, displacement, and size animation effects defined for selected individual and grouped objects. While previewing selections, objects and their properties can be modified; animation effects cannot be modified. However, since the size animation effect is defined as a percentage, the boundaries outlining this effect change as an object is resized.

Visualize the animation effects defined for selected individual and grouped objects from the colored indicators as follows:


The displacement for the Animation Position property where the broken red line indicates the end position and path of travel for the object.

The rotation for the Center of Rotation property where the blue circle indicates the center of rotation for the object.

The size for the Animation Size property where the green broken outline indicates the final size of the object.

Previewing screens is available while online.

## See also

Preview Animation Effects (Editable) on page 733
Preview ISaVIEW Screens on page 732

## Preview Animation Effects (Editable)

Graphically modify the rotation, displacement, and size properties for individual and grouped objects while in animation preview mode. Animation preview mode prohibits to add, delete, move, group, or ungroup objects.

Graphically modify animation effects properties by repositioning the displayed indicators as follows:


- The displacement indicator for the Animation Position property where the red dot and broken line indicate the end position and path of travel for the object.
- The rotation indicator for the Center of Rotation property where the blue dot indicates the center of rotation for the object.
- The size indicator for the Animation Size property green dot and broken lines indicate the final size of the object.

Previewing screens is available while online.
Access the animation preview mode from the ISaVIEW toolbar.

## See also

Preview Selections on page 732
Preview ISaVIEW Screens on page 732

## Language Reference

The language reference includes information about the usage and limitations of various project elements and other aspects:

- Programs
- Functions
- Function Blocks
- Execution Rules
- Reserved Keywords
- Variables
- Directly Represented Variables
- Defined Words
- Data Types


## See also

Programs on page 735

## Execution Rules on page 737

Reserved Keywords on page 738

## Directly Represented Variables on page 739

## Data Types on page 524

Programs are logical program organization units (POUs) describing operations between variables in a process. Programs describe the cyclic operations that execute at each target system cycle.

Cyclic programs systematically execute at the beginning of each run-time cycle and are not time-dependent.

Programs at the beginning of a cycle generally describe preliminary operations on input devices to build high level filtered variables. Programs at the end of the cycle generally describe security operations on the variables. Programs at the end of the cycle also send the values to output devices.

Describe programs using the available graphic or literal languages. Specify the programming language while creating the program. Changing the Programming language is not supported.

POUs defined as programs execute on the Trusted ${ }^{\circledR}$ controller, The POU adheres to the order shown in the Programs section of the Application View. The hierarchical tree links programs. The system activates the programs at the top of the hierarchy. Parents activate child POUs lower in the hierarchy.

Programs have these naming conventions and limitations:

- Unique names limited to eight characters beginning with a letter or single underscore followed by letters, digits, and single underscores.
- Names cannot include reserved words, defined words, or data types.
- A maximum of 255 programs in a project.


## See also

Execution Rules on page 737
Programs on page 525
Functions on page 735

## Function Blocks on page 736

Functions are program organization units (POUs) that have one or more input parameters and one output parameter. Programs, functions, or function blocks call functions. A function has no instance and does not store local data. Local data is generally lost from one call to the other.

The parent program drives the execution of the function. The execution of the parent program is suspended until the function ends:


Any POU of any section calls one or more functions. A function has local variables.

Recursivity is not supported during function calls. A build error occurs when a function of the Functions section calls itself or one of its called functions. Functions do not store the local values of local variables. Since functions do not instantiate, functions cannot call function blocks.

Define the interface of a function with a type and a unique name for each input parameters or output parameter. Functions have up to 31 input parameters and one output parameter.

Functions within a project and the global library have these naming conventions and limitations:

- Unique names limited to eight characters and must begin with a letter or single underscore followed by letters, digits, and single underscores.
- A maximum of 32 parameters -- 31 inputs and one output with names limited to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscores.
- Names cannot be reserved words, defined words, or data types.


## See also

Execution Rules on page 737
Programs on page 735

## Function Blocks on page 736

## Function Blocks

Function blocks are program organization units (POUs) that have multiple input and output parameters. Instantiating copies the local variables of a function block for each instance. A call to a function block in a program calls the instance of the block from the same location as the called code. The function block instance uses the data allocated to the instance. The Trusted ${ }^{\circledR}$ controller stores the variable values of an instance from one cycle to the other.

Call function blocks from any POU in a project. Call functions from function blocks.

Important: Function blocks written with an IEC 61131-3 language other than Ladder Diagram (LD) are unable to call other function blocks. The instantiation mechanism only manages the local variables of the block itself. When the In-line property is set to True for LD blocks, call LD function blocks from other LD function blocks, known as nested blocks. Standard function blocks not supported for use inside an IEC 61131-3 library function block include: AVERAGE, BLINK, CMP, CTD, CTU, CTUD, DERIVATE, F_TRIG, HYSTER, INTEGRAL, LIM_ALRM, R_TRIG, RS, SEMA, SIG_GEN, SR, STACKINT, TOF, TON, and TP.

Define the interface of a function block with a type and a unique name for each of its input parameters or output parameters. Function blocks can have more than one output parameter. The value of an output parameter for a function block differs per programming language.

Function blocks within a project and the global library have these naming conventions and limitations:

- Unique names limited to eight (8) characters beginning with a letter or single underscore followed by letters, digits, and single underscores.
- A maximum of 32 parameters with names limited to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscores.
- Names cannot be reserved words, defined words, or data types.


## See also

Execution Rules on page 737
Programs on page 735
Functions on page 735

## Execution Rules

The execution of a control application is a synchronous system where a clock triggers all operations for a device. The cycle timing performs these steps:

1. Scans input variables
2. Processes program organization units (POUs)
3. Processes Modbus ${ }^{\circledR}$ messages
4. Updates output devices
5. Saves retained values
6. Sleeps until the next cycle

With a specified cycle time, a device waits until this time elapses before starting the execution of a new cycle. The POU's execution time varies depending on the size of the application. When a cycle exceeds the specified time, the loop continues to execute the cycle but sets an overrun flag. The application no longer runs in real time. Without a specified cycle time, a device performs all programs then restarts a new cycle without waiting.

## See also

Programs on page 735
Functions on page 735

Function Blocks on page 736
Trusted Devices on page 523

## Reserved Keywords

Reserved keywords are unavailable for use as names of POUs or variables.

| Keywords |  |
| :---: | :---: |
| 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, BINDING, 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_RESOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHLLE, ENO, EO, EXIT, EXP, EXPT, |
| F | FALSE, FIND, FOR, FUNCTION, |
| G | GE, GFREEZE, GKILL, GLOBALVARIABLE, GRST, GSTART, GSTATUS, GT, |
| H | HEADER, |
| 1 | IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME, 10, |
| 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, |
| 0 | OF, ON, OPERATE, 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, RESOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR, |
| S | S, SEL, SET, SHL, SHR, SIN, SINT, SORT, 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_TOVERFLOW, 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, TSTART, TSTOP, TYPE, |
| $U$ | UDINT, UINT, ULINT, UNTIL, 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 |

## Variables

The scope of variables can be local to a program organization unit (POU) or global to a device. Use local variables within one POU only. Use global variables within any POU of the device.

Variable names may have up to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscore characters. Names cannot be reserved words, defined words, or data types. Names must be unique.

All variables have an attribute and direction. Variables have one of these attributes:

| Attribute | Description |
| :--- | :--- |
| Read/Write | Variable with an initial value that reads or writes |
| Read | Read-only variable with an initial value |

Variables have one direction.

| Direction | Description |
| :--- | :--- |
| Var | Internal variable. The POU updates this variable. |
| Varlnput | For I/O wiring, a variable connected to an input device. For functions or function <br> blocks, a local input variable. The system refreshes these variables. |
| VarOutput | For I/O wiring, a variable connected to an output device. For functions or function <br> blocks, a local output variable. |
| VarGlobal | A global variable |

Assign initial values to global variables or local program variables. Default initial values are 0 or FALSE. An initial value is the value of a variable when a target starts its first cycle.

| Important: | Do not use or assign initial values for local variables located in the <br> Global Library. The compiler ignores these values. |
| :--- | :--- |

## See also

## Data Types on page 524

Variables Grid on page 615
Directly Represented Variables on page 739
Reserved Keywords on page 738
The system enables the use of directly represented variables in the source of programs to represent an unwired channel. Unwired channels are those not linked to a declared I/O variable. The identifier of a directly represented variable always begins with the "\%" character.

The naming conventions of a directly represented variable for a channel of a single board. "s" is the slot number of the board. " c " is the number of the Channel:

## Single board naming conventions

| Convention | Description |
| :--- | :--- |
| \%IXS.C | unwired channel of a Boolean input board |
| \%IDs.C | unwired channel of an integer input board |
| \%ISs.c | unwired channel of a message input board |
| \%QXS.C | unwired channel of a Boolean output board |
| \%ODS.C | unwired channel of an integer output board |
| \%OSs.C | unwired channel of a message output board |

The naming conventions of a directly represented variable for a channel of a complex equipment. "s" is the slot number of the equipment. "b" is the index of the single board within the complex equipment. " c " is the number of the channel:

| Complex equipment naming conventions |  |
| :--- | :--- |
| Convention | Description |
| \%IXs.b.c | unwired channel of a Boolean input board |

Complex equipment naming conventions

| \%IDs.b.c | unwired channel of an integer input board |
| :--- | :--- |
| \%ISs.b.c | unwired channel of a message input board |
| \%QXs.b.c | unwired channel of a Boolean output board |
| \%ODs.b.c | unwired channel of an integer output board |
| \%OSs.b.c | unwired channel of a message output board |

## Example

```
%QX1.6 6th channel of the board #1 (Boolean output)
%ID2.1.7 7th channel of the board #1 in the equipment
#2 (integer input)
```


## Defined Words

The Trusted ${ }^{\circledR}$ application supports identifier names called defined words. When building the application, variables and expressions replace the defined words. Possible scopes for defined words include:

- global scope, available for use in any program organization unit (POU) of a project
- local scope, available for use in only one POU of a project

Trusted programs contain up to 255 defined words. Trusted projects contain a maximum of 3554 defined words.

Defined word names may contain up to 32 characters beginning with a letter or single underscore followed by letters, digits, and single underscores. Names cannot be reserved words, defined words, or data types. Names within a project must be unique. The definition of a defined word cannot contain a defined word.

## Examples of Defined Words



| Item | Name | Description |
| :--- | :--- | :--- |
| $\mathbf{1}$ | Name | Name of the defined word. |
| $\mathbf{2}$ | Equivalent | Equivalent value of the defined word. For POUs, defined words replace literal <br> expressions, boolean expressions, reserved keywords, or complex ST expressions. <br> Defined words that use equivalent complex ST expressions are not supported as <br> inputs for blocks in Ladder Diagram (LD) or Function Blocks Diagram (FBD) POUs. |
| $\mathbf{3}$ | Comment | Free-format text that describes the defined word. Maximum of 60 characters. |

When defining such an equivalence, its identifier replaces the attached expression. This Structured Text programming example uses defined words:

If OK Then
angle := PI / 2.0;
isdone := YES;
End_if;

## See also

Execution Rules on page 737
Reserved Keywords on page 738
Variables on page 738

Data Types

Boolean Data Type

Define any literal, expression, or variable used in a program organization unit (POU) by a data type. Apply data type coherence in graphic operations and literal statements. Program objects using these elementary IEC 61131-3 types:

- BOOL: logic (true or false) value
- DINT - Integer: 32-bit integer value
- REAL - Real: 32-bit real (floating) value
- TIME: time values up to 23 h 59 m 59 s 999 ms . These value types cannot store dates
- MESSAGE: character string having a defined size that represents the maximum number of characters the string can contain. In the dictionary instance, set StringSize to the maximum character length.


## See also

Boolean Data Type on page 741
Double Integer Data Type on page 742
Real Data Type on page 743
Time Data Type on page 743
Message Data Type on page 744
Boolean variables (BOOL) take one of the Boolean values: TRUE or FALSE. Boolean variables are typically used in Boolean expressions.

For Boolean literal expressions, targets evaluate all parts of such expressions. Whereas, the IEC 61131-3 standard states that Boolean expressions may be evaluated only to the extent necessary to determine the resultant value. In the following example according to the IEC 61131-3 standard, if $B$ is zero then the first expression ( $B<>0$ ) is false and the second expression ( $A / B>0$ ) is not performed.

```
if ((B <> 0) and (A/B > 0)) then
GREATER := true;
else
GREATER := false;
end_if;
```

Boolean literal expressions are:

- TRUE is equivalent to the integer value 1


## FALSE is equivalent to the integer value $\mathbf{0}$ See also

Double Integer Data Type on page 742
Real Data Type on page 743
Time Data Type on page 743

## Message Data Type on page 744

## Double Integer Data Type

Double integer variables are 32-bit signed integer values ranging from 2147483647 to +2147483647 . Double Integer variables can have one of these attributes:

- Internal: memory variable updated by the program
- Constant: read-only memory variable with an initial value
- Input: variable connected to an input device (refreshed by the system)
- Output: variable connected to an output device
Important: A double integer expression cannot contain integer and real variables or literal expressions.

A bit of an integer variable can be accessed using this syntax:
MyVar.i
If MyVar is an Integer.
MyVar.i is a Boolean. "i" must be a literal value from 0 to 31.
Integer literal values represent signed long integer (32-bit) values ranging from -2147483647 to +2147483647 . Integer literals may be expressed with one of these bases. Integer literals must begin with a prefix identifying the base 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 |

Tip: Do not use the underscore character ('_') to separate groups of digits.

## See also

Boolean Data Type on page 741
Real Data Type on page 743
Time Data Type on page 743
Message Data Type on page 744

## Real Data Type

## Example

| 3.14159 | $-1.0 \mathrm{E}+12$ |
| :--- | :--- |
| +1.0 | $1.0 \mathrm{E}-15$ |
| -789.56 | $+1.0 \mathrm{E}-37$ |

The expression "123" does not represent a Real literal value. Its correct real representation is "123.0".

## See also

## Boolean Data Type on page 741

Double Integer Data Type on page 742
Time Data Type on page 743
Message Data Type on page 744
Time variables are typically used in Time expressions. A Time value represents positive values from 0 to 23 h 59 m 59 s 999 ms . Time variables are stored in 32 bit words. The internal representation is a positive number of milliseconds. Time variables can be used with timer function blocks such as TOF and TON. Timer variables can have one of these attributes:

- Internal: memory variable managed by the program, refreshed by the system
- Constant: read-only memory variable with an initial value

Time variables cannot have the INPUT or OUTPUT attributes.
When a timer is active, its value is automatically increased according to the target system real-time clock. Use these Structured Text (ST) language statements to control a timer:

- TSTART, starts automatic refresh of a timer
- TSTOP, stops automatic refresh of a timer

Time literal values represent time values from 0 to 23 h 59 m 59 s 999 ms . The lowest allowed unit is a millisecond. Standard time units used in literal values are:

| Time unit | Description |
| :--- | :--- |
| Hours | The "h" letter must follow the number of hours |
| Minutes | The "m" letter must follow the number of minutes |
| Seconds | The "s" letter must follow the number of seconds |
| Milliseconds | The "ms" letter must follow the number of milliseconds |

The time literal value 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 controller time value displays the time value from T\#Os to T\#49d17h2m47s294ms when monitoring variables.

The simulator time value displays the time value from T\#Os to T\#23h59m59s999ms when monitoring variables.

The max initial value that can be assigned to the time data type is T\#23h59m59s999ms. To use a larger value, from the POU, assign a literal value (e.g. T\#596h31m22s) to the time variable.

The max value that can be forced for a time variable is T\#23h59m59s999ms.

## See also

Boolean Data Type on page 741
Double Integer Data Type on page 742
Real Data Type on page 743
Message Data Type on page 744

## Message Data Type

Message 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 240 characters excluding the terminating null character ( 0 ), a byte for the current length of the string, and a byte for the maximum length of the string.

Message 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.

Messages have a size representing the maximum number of characters that the string can contain.

Message literal values represent character strings. Characters must be preceded and followed by single quote (') characters. Example:
'THIS IS A MESSAGE'
Tip: When placing single quote (') characters within a message literal, precede these characters with the dollar (\$) character. In this message literal, the dollar character precedes the single quote character.
'THIS IS \$' A MESSAGE'
A message literal value must be expressed on one line of the program source code. Its length cannot exceed 240 characters, including spaces.

Empty message literal values are represented two single quote (') characters, 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 message literal value to represent a non-printable character:

| Sequence | Meaning | ASCII(hex) | Example |
| :---: | :---: | :---: | :---: |
| \$ \$ | '\$' character | 16\#24 | 'I paid \$\$5 for this' |
| \$' | apostrophe | 16\#27 | 'Enter \$'YS' for YES' |
| \$L | line feed | 16\#0a | 'next \$L line' |
| \$R | carriage return | 16\#Od | 'Ilo \$R He' |
| \$N | new line | 16\#OdOa | '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 $=$ \$ $41 \$ 42 \$ 43 \$ 44{ }^{\text {a }}$ |

$\left(^{*}\right)$ "hh" is the hexadecimal value of the ASCII code for the expressed character.

## See also

Boolean Data Type on page 741
Double Integer Data Type on page 742
Real Data Type on page 743
Time Data Type on page 743

## Operators

These are Operators of the IEC 61131-3 languages:

| Operations |  |  |  |
| :--- | :--- | :---: | :---: |
| Arithmetic Operations | Description |  |  |
| Addition | Adds two or more variables |  |  |
| Division | Divides two variables |  |  |
| Multiplication | Multiplies two or more variables |  |  |
| Subtraction | Subtracts a variable from another |  |  |
| 1GAIN | Assigns one variable into another |  |  |
| NEG | Integer negation |  |  |
| Boolean Operations | Boolean AND |  |  |
| AND | Boolean OR |  |  |
| OR | Boolean exclusive OR |  |  |
| XOR | Tests if one value is less than another |  |  |
| Comparators | Tests if one value is less than or equal to another |  |  |
| Less Than | Tests if one value is greater than another |  |  |
| Less Than or Equal | Tests if one value is greater than or equal to another |  |  |
| Greater Than | Tests if one value is equal to another |  |  |
| Greater Than or Equal | Tests if one value is not equal to another |  |  |
| Equal |  |  |  |
| Not Equal | Converts to Boolean |  |  |
| Data Conversion | Converts to real |  |  |
| BOO | Converts to real |  |  |
| ANA | Converts to message |  |  |
| REAL | Converts to time |  |  |
| MSG | Concatenates multiple messages into one |  |  |
| TMR | Varies depending on the implementation of the treated I/O |  |  |
| Internal Operations | Accesses the system parameters |  |  |
| OPERATE |  |  |  |
| SYSTEM | String Manipulation |  |  |

## See also

Functions on page 765
Function Blocks on page 806
Assignment of one variable into another.
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.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN | DINT - BOOL - <br> MESSAGE - REAL - <br> TIME |  |
| 0 | DINT - BOOL - <br> MESSAGE - REAL - <br> TIME | IN and Q must have the same format |

## Example

(*FBD example with assignment Operators *)

(* ST equivalence: *)

```
ao23 := ai10;
bo100 := NOT (bi1 AND bi2);
```


## See also

## AND on page 748

Adds 2 to 32 integer, real, or time variables.


## Arguments

| (inputs) | DINT - REAL - TIME | All inputs must have the same format. From 2 to 32 inputs. |
| :--- | :--- | :--- |
| output | DINT - REAL - TIME | addition of the input variables |

## Example

(* FBD example with Addition Operators *)

(* ST equivalence: *)

```
ao10 := ai101 + ai102;
ao5 := ai51 + ai52 + ai53;
```


## See also

Division on page 751
Subtraction on page 760
Multiplication on page 756
Converts a non-integer variable to an integer variable.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN | BOOL - MESSAGE - REAL - TIME | A non-integer value |
| $Q$ | DINT | O if $\operatorname{IN}$ is FALSE $/ 1$ if $\operatorname{Nis}$ is TRE <br> Number of milliseconds for a timer <br> Integer part for real <br> Decimal number represented by a string |

## Example

(* FBD example with "ANA" operators *)

(* ST equivalence: *)

```
bres := ANA (true); (* bres is 1 *)
tres := ANA (t#1s46ms); (* tres is 1046 *)
mres := ANA ('0198'); (* mres is 198 *)
```


## See also

BOO on page 749
MSG on page 756
REAL on page 760
TMR on page 764

In the text editor, use the ' $\&$ ' character or type AND.


Arguments

| (inputs) | BOOL | From 2 to 32 inputs. |
| :--- | :--- | :--- |
| output | BOOL | Boolean AND of the input variables |

## Example

(* FBD example with "AND" Operators *)

(* ST equivalence 1: *)

```
bo10 := bi101 AND NOT (bi102);
bo5 := bi51 AND bi52 AND bi53;
```

(* ST equivalence 2: *)

```
bo10 := bi101 & NOT (bi102);
bo5 := bi51 & bi52 & bi53;
```


## See also

## 1 GAIN on page 746

OR on page 759
XOR on page 764
Converts a non-boolean variable to a boolean variable.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN | DINT- MESSAGE - REAL - <br> TIME | A non-boolean value |


| 0. | BOOL | TRUE for non-zero numerical value <br> FALSE for zero numerical value <br> TRUE for 'TRUE' message <br> FALSE for 'FALSE' message |
| :--- | :--- | :--- |

## Example

(* FBD example with "BOO" operators *)

(* ST equivalence: *)

```
ares := BOO (10); (* ares is TRUE *)
tres := BOO (t#Os); (* tres is FALSE *)
mres := BOO ('false'); (* mres is FALSE *)
```


## See also

ANA on page 748
MSG on page 756
REAL on page 760
TMR on page 764
Concatenates 2 to 32 messages into one message.


## Arguments

| (inputs) | MESSAGE | From 2 to 32 inputs. The output message capacity limits the length of <br> the concatenated message. |
| :--- | :--- | :--- |
| output | MESSAGE | Concatenation of the input messages. |

## Example

(*FBD example with "CAT" Operator *)

(* ST equivalence: *)
MyName : = ('Mr' + ' ') + 'Jones';
(* means: MyName := 'Mr Jones' *)

## See also

## MSG on page 756

Division
Divides two integer or real variables (the first divided by the second).


## Arguments

| IN1 | DINT - REAL | can be integer or real format (operand) |
| :--- | :--- | :--- |
| IN2 | DINT - REAL | non-zero integer or real value (divisor) <br> (IN1 and IN2 must have the same format) |
| $\mathbf{Q}$ | DINT - REAL | integer or real division of IN1 by IN2 |

## Example

(* FBD example with Division Operators *)

(*ST Equivalence: *)

```
ao10 := ai101 / ai102;
ao5 := (ai51 / 1) / ai53;
```


## See also

Addition on page 747
Equal on page 752
Multiplication on page 756
Subtraction on page 760

## Equal

Tests if one value is EQUAL TO another one (on integer, real, bool, and message variables).


| Arguments |  |  |
| :--- | :--- | :--- |
| IN1 | DINT - BOOL - MESSAGE - REAL | Both inputs must have the same format. |
| IN2 | DINT - BOOL - MESSAGE - REAL |  |
| 0 | BOOL | TRUE if IN1 = IN2 |

## Example

(* FBD example with "Is Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```


## See also

Addition on page 747
Division on page 751
Multiplication on page 756
Not Equal on page 758
Subtraction on page 760

## Greater Than

Tests if one value is GREATER THAN another one (on integer, real, bool, time, and message variables).


## Arguments

| IN1 | DINT - BOOL - MESSAGE - REAL - <br> TIME | Both inputs must have the same type |
| :--- | :--- | :--- |
| IN2 | DINT - BOOL - MESSAGE - REAL - <br> TIME |  |
| 0 | BOOL | TRUE if $\operatorname{IN1}>\operatorname{IN2}$ |

## Example

(* FBD example with "Greater than" Operators *)

(* ST Equivalence: *)

```
aresult := (10 > 25); (* aresult is FALSE *)
mresult := ('ab' > 'a'); (* mresult is TRUE *)
```


## See also

Greater Than or Equal on page 753
Less Than on page 754
Less Than or Equal on page 755
Tests if one value is GREATER THAN or EQUAL TO another one (on integer, real, bool, and message variables).


## Arguments

| IN1 | DINT - BOOL - <br> MESSAGE - REAL | Both inputs must have the same type. |
| :--- | :--- | :--- |
| IN2 | DINT - BOOL - <br> MESSAGE - REAL |  |


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| $Q$ | BOOL | TRUE if $\operatorname{NN} 1>=\mid \mathrm{N} 2$ |  |

## Example

(* FBD example with "Greater or Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 >= 25); (* aresult is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```


## See also

Greater Than on page 752
Less Than on page 754
Less Than or Equal on page 755
Less Than
Tests if one value is LESS THAN another one (on integer, real, bool, time, and message variables).


| Arguments |  |  |
| :--- | :--- | :--- |
| IN1 | DINT - BOOL - <br> MESSAGE - REAL - <br> TIME | Both inputs must have the same type |
| IN2 | DINT - BOOL - <br> MESSAGE - REAL - <br> TIME |  |
| 0 | BOOL | TRUE if $\operatorname{IN} 1<$ IN2 |

## Example

(* FBD example with "Less than" Operators *)

(*ST Equivalence: *)

```
aresult := (10 < 25); (* aresult is TRUE *)
mresult := ('z' < 'B'); (* mresult is FALSE *)
```


## See also

Greater Than on page 752
Greater Than or Equal on page 753
Less Than or Equal on page 755

## Less Than or Equal

Tests if one value is LESS THAN or EQUAL TO another one (on integer, real, bool, and message variables).


Arguments

| IN1 | DINT - BOOL - <br> MESSAGE - REAL | Both inputs must have the same type. |
| :--- | :--- | :--- |
| IN2 | DINT - BOOL - <br> MESSAGE - REAL |  |
| 0 | BOOL | TRUE if $\operatorname{IN1}<=\operatorname{IN} 2$ |

## Example

(* FBD example with "Less or equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```


## See also

Greater Than on page 752
Greater Than or Equal on page 753
Less Than on page 754
Converts an integer, real, boolean, or time variable to a string variable.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN | DINT - BOOL - REAL - TIME | A non-string value |
| 0 | MESSAGE | "false' or 'true' i IN is a boolean value <br> decimal representation if IN is an integer or real |

## Example

(* FBD example with "Convert to Message" blocks *)

(* ST Equivalence: *)

```
bres := MSG (TRUE); (* bres is 'TRUE' *)
ares := MSG (125); (* ares is '125' *)
```


## See also

ANA on page 748
BOO on page 749
CAT on page 750
REAL on page 760
TMR on page 764

## Multiplication

Multiplies 2 to 32 integer or real variables.


Arguments

| (inputs) | DINT - REAL | All inputs must have the same format. From 2 to 32 inputs. |
| :--- | :--- | :--- |
| output | DINT - REAL | Multiplication of the input terms. |

## Example

(* FBD example with Multiplication Operators *)

(* ST equivalence *)

```
ao10 := ai101 * ai102;
ao5 := ai51 * ai52 * ai53;
```


## See also

Addition on page 747
Division on page 751
Equal on page 752
Not Equal on page 758
Subtraction on page 760
Assignment of the negation of a variable.


Arguments

| IN | DINT - REAL | Input and output must have the same format |
| :--- | :--- | :--- |
| 0 | DINT - REAL |  |

## Example

(* FBD example with Negation Operators *)

(* ST equivalence: *)

```
ao23 := - (ai10);
ro100 := - (ri1 + ri2);
```


## See also

Addition on page 747
Tests if one value is NOT EQUAL TO another one (on integer, real, boolean, and message variables).


Arguments

| IN1 | DINT - BOOL - <br> MESSAGE - REAL | both inputs must have the same type |
| :--- | :--- | :--- |
| IN2 | DINT - BOOL - <br> MESSAGE - REAL |  |
| 0 | BOOL | TRUE if first $<>$ second |

## Example

(* FBD example with "Is Not Equal to" Operators *)

(* ST Equivalence: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```


## See also

Addition on page 747
Division on page 751
Equal on page 752
Multiplication on page 756
Subtraction on page 760

## OPERATE

## Accesses an IO channel.

Tip: $\quad$ The meaning of OPERATE arguments differs from one I/O interface implementation to another.


| Arguments |  |  |
| :--- | :--- | :--- |
| IO | DINT - BOOL - MESSAGE - REAL - <br> TIME | Input or output variable |
| FUNCT | DINT | Action to be performed |
| ARG | DINT | Argument for I/O action |
| 0 | DINT | Return check |

Boolean OR of 2 to 32 Boolean variables.


Arguments

| (inputs) | BOOL | From 2 to 32 inputs. |
| :--- | :--- | :--- |
| output | BOOL | Boolean OR of the input variables. |

## Example

(* FBD example with "OR" Operators *)

(* ST equivalence: *)

```
bo10 := bi101 OR NOT (bi102);
bo5 := (bi51 OR bi52) OR bi53;
```


## See also

AND on page 748
XOR on page 764
Converts a non-real variable to a real variable.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| IN | DINT - BOOL - <br> MESSAGE - TIME | A non-real value (no message) |  |
| 0 | REAL | 0.0 if IN is FALSE / 1.0 if IN is TRUE <br> Number of milliseconds for a timer <br> Equivalent number for integer |  |

## Example

(* FBD example with "REAL" operators *)

(* ST Equivalence: *)

| bres $:=$ REAL (true); | $(*$ bres is 1.0 *) |
| :--- | :--- |
| tres $:=$ REAL (t\#1s 46 ms$) ;$ | $(*$ tres is 1046.0 *) |
| ares $:=$ REAL $(198) ;$ | $(*$ ares is $198.0 *)$ |

## See also

ANA on page 748
BOO on page 749
MSG on page 756
TMR on page 764

## Subtraction

Subtracts two integer or real variables (first - second).


## Arguments

| IN1 | DINT - REAL | can be integer or real format |
| :--- | :--- | :--- |
| IN2 | DINT - REAL | (IN1 and IN2 must have the same format) |
| $Q$ | DINT - REAL | subtraction (first - second) |

## Example

(* FBD example with Subtraction Operators *)

(* ST equivalence: *)

```
ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;
```


## See also

Addition on page 747
Division on page 751
Equal on page 752
Multiplication on page 756
Accesses the system parameters to enable performing these tasks:

- Reading various cycle timing information and changing cycle timing
- Resetting timing counters
- Checking for and reading run-time errors
- Backing up, saving, and restoring variables


| Arguments |  |  |
| :--- | :--- | :--- |
| MODE | DINT | Identifies the system parameter and the access mode |
| ARG | DINT | New value for a "write" access |
| PARAM | DINT | Value of the accessed parameter |

These are the available commands (pre-defined keywords) and expected arguments for the SYSTEM operator:

| Command | Meaning | Argument | Value | Return Value |
| :--- | :--- | :--- | :--- | :--- |
| SYS_TALLOWED | reads allowed cycle timing | 0 | 1 | allowed cycle timing |
| SYS_TCURRENT | reads current cycle timing | 0 | 2 | current cycle timing |
| SYS_TMAXIMUM | reads maximum cycle timing | 0 | 3 | maximum detected timing |
| SYS_TOVERFLOW | reads cycle timings overflows | 0 | 4 | number of timing overflows |
| SYS_TWRITE | changes cycle timing | new allowed <br> cycle timing | 5 | written time |
| SYS_TRESET | resets timing counters | 0 | 6 | 0 |
| SYS_ERR_TEST | checks for run time errors | 0 | 16 | 0 |
| SYS_ERR_READ | reads oldest run time error | 0 | 17 | oldest error code detected |
| SYS_INITBOO | backs up init Boolean | memory address | 32 | next free address |
| SYS_SAVBOO | saves Booleans | 0 | 33 | zero if OK |
| SYS_RESTBOO | restores Booleans | 0 | 34 | zero if 0K |
| SYS_INITANA | backs up init analog | memory address | 36 | next free address |
| SYS_SAVANA | saves analogs | 0 | 37 | zero if 0K |
| SYS_RESTANA | restores analogs | 0 | 38 | zero if OK |
| SYS_INITTMR | backs up init timer | memory address | 40 | next free address |
| SYS_SAVTMR | saves timers | 0 | 41 | zero if 0K |
| SYS_RESTTMR | restores timers | 0 | 42 | zero if 0K |
| SYS_INITALL | backs up init all types | memory address | 44 | next free address |
| SYS_SAVALL | saves all types | 0 | 45 | zero if OK |
| SYS_RESTALL | restores all types | 0 | 46 | zero if OK |

Define the memory backup location using this syntax to back up variables for a specific type or for all types:
<new_address> := SYSTEM(SYS_INITxxx,<address>);
where:
<address> is the memory backup address location (16\# value for Hexadecimal format). The location must be an even address or the operation fails.

SYS_INITXxx can be one of these:
SYS_INITBOO to define memory backup location for all Boolean variables. SYS_INITANA to define memory backup location for all analog variables. SYS_INITTMR to define memory backup location for all timer variables. SYS_INITALL to define memory backup location for all Boolean, analog, and timer variables.
<new_address> gets the next free address, for example, <address> + size of backed up variables (in bytes) according to SYS_INITxxx. This enables verifying the size of the required memory backup. If the operation fails, <new_address> gets a zero value.

After having defined the backup memory location, perform backups of the variables at any time during the application. The backup is performed once only at the end of the current cycle. If the hardware delivers a Boolean input or a C function to inform of a power failure and allows at least one cycle delay
before closing down, the backup may only be performed after detecting the power failure.
<error> :=SYSTEM(SYS_SAVxxx,0);
where:
SYS_SAVxxx can be one of these:
SYS_SAVBOO to ask for all Boolean variables backup.
SYS_SAVANA to ask for all analog variables backup.
SYS_SAVTMR to ask for all timer variables backup.
SYS_SAVALL to ask for all Boolean, analog and timer variables backup.
<error> gets an error status other than zero when the operation fails (SYS_INITxxx is not called).

Restore variables at any time during the application. Perform the restoration once only at the end of the current cycle. Set an analog variable to a constant value for use as a signature to ensure the validity of the backed up data.
<error> := SYSTEM(SYS_RESTxxx,0);
where:
SYS_RESTxxx can be one of these:
SYS_RESTBOO to restore all Boolean variables.
SYS_RESTANA to restore all analog variables.
SYS_RESTTMR to restore all timer variables.
SYS_RESTALL to restore all Boolean, analog and timer variables.
<error> gets an error status other than zero when the operation fails (SYS_INITxxx is not performed).

## Example

(* FBD example with "SYSTEM" operators *)

(* ST Equivalence: *)

```
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
    nb_err := nb_err + 1;
    rc := SYSTEM (SYS_TRESET, 0);
End_If;
```


## Converts an integer or real variable to a time one.



## Arguments

| IN | DINT - REAL | A non-TIME value <br> IN (or integer part of $\operatorname{IN}$ if it is real) <br> is the number of milliseconds |
| :--- | :--- | :--- |
| $Q$ | TIME | Time value represented by IN |

## Example

(* FBD example with "Convert to Timer" Operators *)

(* ST Equivalence: *)

```
ares := TMR (1256); (* ares := t#1s256ms *)
rres := TMR (1256.3); (*rres := t#1s256ms *)
```


## See also

ANA on page 748
BOO on page 749
MSG on page 756
REAL on page 760
Boolean exclusive OR between two terms.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN1 | B0OL |  |
| IN2 | BOOL |  |

## Arguments

| $Q$ | BOOL | Boolean exclusive OR of both input items |
| :--- | :--- | :--- |

## Example

(* FBD example with "XOR" operators *)

(* ST equivalence: *)

```
bo10 := bi101 XOR NOT (bi102);
bo5 := (bi51 XOR bi52) XOR bi53;
```


## See also

## AND on page 748

OR on page 759

## Functions

The system supports these functions:

| Function |  |  |
| :--- | :--- | :---: |
| Arithmetic Operations | Description |  |
| 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 |  |
| Array Manipulation |  |  |
| ARCREATE | Creates an array of integers |  |
| ARREAD | Reads an element in an array of integers |  |
| ARWRITE | Stores (writes) a value in an array of integers |  |
| 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 |  |


| Function | Description |  |
| :--- | :--- | :---: |
| SHL, SHR | Shift Left, Shift Right an integer value |  |
| Boolean Operations | Odd parity |  |
| ODD | Minimum, Maximum, Limit |  |
| Data Manipulation | Multiplexer (4 or 8 entries) |  |
| MIN, MAX, LIMIT | Binary selector |  |
| MUX4, MUX8 | Closes a binary file |  |
| SEL | Tests if end of a file has been reached |  |
| File Management (for Trusted ${ }^{\text {® }}$ Toolset configurations only) |  |  |
| F_CLOSE | Opens a binary file in read mode |  |
| F_EOF | Opens a binary file in write mode |  |
| F_ROPEN | Reads integer and real variables from a binary file |  |
| F_WOPEN | Writes integer and real variables to a binary file |  |
| FA_READ | Reads MESSAGE variables from a binary file |  |
| FA_WRITE | Writes MESSAGE variables to a binary file |  |
| FM_READ |  |  |
| FM_WRITE | Character -> ASCII code |  |
| String Manipulation | ASCII code -> Character |  |
| ASCII | Get string length |  |
| CHAR | Delete sub-string, Insert string |  |
| MLEN | Find sub-string, Replace sub-string |  |
| DELETE, INSERT | Extract left, middle or right of a string |  |
| FIND, REPLACE | Gives date or time of the day |  |
| LEFT, MID, RIGHT |  |  |

## See also

Operators on page 745
Function Blocks on page 806
Gives the absolute (positive) value of a real value.


## Arguments

| IN | IN | REAL | Any signed real value |
| :--- | :--- | :--- | :--- |
| ABS | 0 | REAL | Absolute value (always positive) |

## Example

(* FBD Program using "ABS" Function *)

(* ST Equivalence: *)
over := (ABS (delta) > range);

## See also

REAL on page 760
Greater Than on page 752
LOG on page 787
POW on page 796
Yields the Arc Cosine of a REAL value. Input and output values are in radians.


| Arguments |  |  |  |
| :--- | :--- | :--- | :--- |
| IN | IN | REAL | Must be in set $[-1.0 . .+1.0]$ |
| ACOS | 0 | REAL | Arc-cosine of the input value (in set $[0.0 ~ . . ~ P I]) ~$ <br> $=0.0$ for invalid input |

## Example

(* FBD Program using "COS" and "ACOS" Functions *)

(* ST Equivalence: *)

```
cosine := CoS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```


## See also

ASIN on page 771
ATAN on page 772
COS on page 773
SIN on page 803
TAN on page 804

Integer AND bit-to-bit mask.


## Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| MSK | MSK | DINT | Must have integer format |
| AND_MASK | Q | DINT | Bit-to-bit logical AND between IN and MSK |

## 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
*)
```


## See also

NOT MASK on page 794
OR MASK on page 795

## XOR MASK on page 806

Creates an array of integers.


| Arguments |  |  |
| :--- | :--- | :--- |
| ID | DINT | Identifier of the array (must be in set [0..15]) |
| SIZE | DINT | Number of elements in the array |


| OK | DINT | execution status: <br> $1=$ if array has been successfully created <br> $2=$ invalid array identifier or array already created <br> 3 |
| :--- | :--- | :--- |
|  |  | invalid size <br> $4=$ not enough memory |

## Example

(* FBD Program creating an array of integers*)

(* ST Equivalence: *)

```
array error := (ARCREATE (ident, 10) <> 1));
```


## See also

ARREAD on page 769
ARWRITE on page 770
Reads an element in an array of integers.


| Arguments |  |  |
| :--- | :--- | :--- |
| ID | DINT | Identifier of the array (must be in set [0..15]) |
| POS | DINT | Position of the element in the array must be in set [0 .. size-1] $]$ |
| $O$ | DINT | value of the element read <br> 0 if the arguments are not valid |

## Example

(* FBD program using an array management function*)

(* ST Equivalence: *)

```
If (array_error) Then Return;
End_if;
read_value := ARREAD (ident, index);
(* array_error comes from the ARCREATE call *)
```


## See also

ARCREATE on page 768
ARWRITE on page 770

## ARWRITE

Stores (writes) a value in an array of integers.


| Arguments |  |  |
| :--- | :--- | :--- |
| ID | DINT | Identifier of the array (must be in set [0.15]) |
| POS | DINT | Position of the element in the array; must be in set [0.. size-1] |
| IN | DINT | New value for the element |
| OK | DINT | Execution status: <br> $1=$ $=$ writing has succeeded <br> $2=$ invalid array identifier <br> $3=$ invalid index |

## Example

(* FBD program using an array management function*)

(* ST Equivalence: *)

```
If (array_error) Then Return;
End_if;
read_value := ARWRITE (ident, index, value);
(* array_error comes from the ARCREATE call *)
```


## See also

ARCREATE on page 768
ARREAD on page 769
Gives the ASCII code of one character in a message string.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| Pos | Pos | DINT | Position of the selected character in set [1.. Ien] (len is the <br> length of the IN message) |
| ASCII | Code | DINT | Code of the selected character (in set [0 .. 255]) <br> returns 0 is Pos is out of the string |

## Example

(* FBD Program using "ASCII" Function *)

(* ST Equivalence: *)

```
FirstChr := ASCII (message_input, 1);
(* FirstChr is the ASCII code of the first character of
the string *)
```


## See also

CHAR on page 773
MID on page 788
Calculates the Arc sine of a real value.


## Arguments

| IN | IN | REAL | Must be in set $[-1.0 . .+1.0]$ |
| :--- | :--- | :--- | :--- |
| ASIN | 0 | REAL | Arc-sine of the input value (in set $[-P / / 2 . .+$ PI/2]) <br> $=0.0$ for invalid input |

## Example

(* FBD Program using "SIN" and "ASIN" Functions *)

(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```


## See also

ACOS on page 767
ATAN on page 772
COS on page 773
SIN on page 803
TAN on page 804
Calculates the arc tangent of a real value.


## Arguments

| $\mathbb{N}$ | IN | REAL | Any real value |
| :--- | :--- | :--- | :--- |
| ATAN | 0 | REAL | Arc-tangent of the input value (in set $[-P / / 2 . .+P / / 2])$ <br> $=0.0$ for invalid input |

## Example

(* FBD Program using "TAN" and "ATAN" Function *)

(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```


## See also

ACOS on page 767
ASIN on page 771
COS on page 773
SIN on page 803
TAN on page 804

## 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' *)
```


## See also

ASCII on page 771
MID on page 788
Calculates the cosine of a real value.


Arguments

| $\mathbb{N}$ | $\mathbb{N}$ | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| $\operatorname{COS}$ | 0 | REAL | Cosine of the input value (in set $[-1.0 . .+1.0])$ |

## Example

(* FBD Program using "COS" and "ACOS" Functions *)

(* ST Equivalence: *)
cosine := $\operatorname{COS}$ (angle);
result := ACOS (cosine); (* result is equal to angle *)

## See also

ACOS on page 767
ASIN on page 771
ATAN on page 772
SIN on page 803
TAN on page 804
Gives date or time of the day as a message string.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| SEL | DINT | output selection <br> $0=$ get current date <br> $1=$ get current time <br> $2=$ get day of week |  |
| 0 | MESSAGE | time/date expressed on a character string <br> "YYYY/MM/DD' if SEL = 0 |  |
| "HH:MM:SS' if SEL = 1 |  |  |  |
| day name if SEL = 2 (ex: 'Monday') |  |  |  |

## Example


(* ST Equivalence: *)

```
Display := Day_Time (0) + ' ; ' + Day_Time (1);
(* Display text format is: 'YYYY/MM/DD ; HH:MM:SS' *)
```


## See also

## CAT on page 750

Deletes a part of a message string.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first deleted character (first character of the <br> string has position 1) |
| DELETE | 0 | MESSAGE | modified string <br> empty string if Pos $<1$ <br> initial string if Pos $>$ IN string length <br> initial string if NbC $<=0$ |

## Example


(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is
```

'ABCDEFG $\bar{H}$ ' *)
sub_string := DELETE (complete_string, 4, 3); (*
sub_string is 'ABGH'*)

## See also

CAT on page 750
ASCII on page 771
MID on page 788

## EXPT

Gives the real result of the operation: (base exponent) 'base' being the first argument and 'exponent' the second one.


Arguments

| IN | IN | REAL | Any signed real value |
| :--- | :--- | :--- | :--- |
| EXP | EXP | DINT | Integer exponent |
| EXPT | 0 | REAL | (IN ExP) |

## Example

(* FBD Program using "EXPT" Function *)

(* ST Equivalence: *)

```
tb size := ANA (EXPT (2.0, range) );
```


## See also

ANA on page 748
REAL on page 760

Closes a binary file open with functions F_ROPEN or F_WOPEN.
Important: Avoid using this function. This function is intended to be obsolete.


## Arguments

| ID | DINT | File number returned by F_ROPEN or F_WOPEN |
| :--- | :--- | :--- |
| OK | BOOL | return status <br> TRUE if file close is OK <br> FALSE if an error occurred |

## Example

(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_ROPEN('data.bin');
ok := F_CLOSE(file_id);
```


## See also

F ROPEN on page 778
F WOPEN on page 778

## F_EOF

Tests if end of file has been reached.

| Important: Avoid using this function. This function is intended to be obsolete. |
| :--- |
| Arguments |
| ID |
| OK |

## Example

(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
VAL := FA_READ(file_id);
END_WHILE;
message_input:= 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);
```


## See also

F ROPEN on page 778
F WOPEN on page 778
FA READ on page 779
F CLOSE on page 776
FM READ on page 782

## F_ROPEN

Opens a binary file in read mode. For use with FA_READ, FM_READ, and F_CLOSE.

| Important: Avoid using this function. This function is intended to be obsolete. |
| :--- |
| Arguments |
| PATH |
| MESSAGE |
| ID |

Example
(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_ROPEN('c:\data \data.bin');
error := (file_id=0);
```


## See also

F WOPEN on page 778
FA READ on page 779
FM READ on page 782
F CLOSE on page 776

## F_WOPEN

Opens a binary file in write mode. For use with FA_WRITE, FM_WRITE, and F_CLOSE.

Important: Avoid using this function. This function is intended to be obsolete.

Arguments

| PATH | MESSAGE | May include the access path to the file using the \or / symbol to specify a <br> directory. To ease application portability, I or $\backslash$ is equivalent. |
| :--- | :--- | :--- |
| ID | DINT | File number <br> 0 if an error occurs. If the file already exists, it is overwritten |

## Example

(* FBD program using file management blocks *)

(*ST Equivalence: *)

```
file_id := F_WOPEN('c:\data \data.bin');
error := (file_id=0);
```


## See also

F ROPEN on page 778
FA WRITE on page 780
FM WRITE on page 783
F CLOSE on page 776
Reads integer variables from a binary file. For use with F_ROPEN and F_CLOSE. This procedure makes a sequential access to the file, from the previous position. The first call after F_ROPEN reads the first four bytes of the file, each call pushes the reading pointer. To check if the end of file is reached, use F_EOF.

Important: Avoid using this function. This function is intended to be obsolete.


| Arguments |  |  |  |
| :--- | :--- | :--- | :---: |
| ID | DINT | File number: returned by F_ROPEN |  |
| 0 | DINT | Integer value read from file |  |

## Example

(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_ROPEN('voltramp.bin');
vstart := FA_READ(file_id);
vend := FA_READ(file_id);
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);
```


## See also

F ROPEN on page 778
F CLOSE on page 776
F EOF on page 777
FA WRITE on page 780
Writes integer variables to a binary file. This procedure makes a sequential access to the file, from the previous position. The first call after F_WOPEN writes the first four bytes of the file, each call pushes the writing pointer.

```
Important: Avoid using this function. This function is intended to be obsolete.
```



## Arguments

| ID | DINT | File number: returned by F_WOPEN |
| :--- | :--- | :--- |
| IN | DINT | Integer value to be written in the file |
| OK | BOOL | Execution status: TRUE if ok |

## Example

(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_WOPEN('voltramp.bin');
nb_written := 0;
nb_written := nb_written + dint(FA_WRITE(file_id,vstart));
nb_written := nb_written + dint(FA_WRITE(file_id,vend));
nb_written := nb_written + dint(FA_WRITE(file_id,vinc));
nb_written := nb_written +
din̄t(FA_WRITE(file_id,dint(delta_tim)));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
ERROR := ERR_FILE;
END_IF;
```


## See also

F WOPEN on page 778
F CLOSE on page 776
FA READ on page 779

Finds a sub-string in a message string. Gives the position in the string of the sub-string.


Arguments

| IN | IN | MESSAGE | Any message string |
| :--- | :--- | :--- | :--- |
| Pat | Pat | MESSAGE | Any non-empty string (Pattern) |
| FIND | Pos | DINT | $=0$ if sub string Pat not found <br> $=$ position of the first character of the first occurrence of <br> the sub-string Pat <br> (first position is 1) <br> this function is case sensitive |

## Example


(*ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string
is 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found is 3
*)
```


## See also

## DELETE on page 775

CAT on page 750
ASCII on page 771
Reads message variables from a binary file. For use with F_ROPEN and F_CLOSE. This procedure makes a sequential access to the file, from the previous position. The first call after F_ROPEN reads the first string of the file, each call pushes the reading pointer. A string is a terminated by null (0), end of line (' $\backslash \mathrm{n}$ ') or return (' $\backslash r$ '); To check if the end of file is reached, use F_EOF.

[^14]

## Arguments

| ID | DINT | file number: returned by F_ROPEN |
| :--- | :--- | :--- |
| Q | MESSAGE | message value read from file |

Example
(* FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
    error := err_file;
    unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);
```


## See also

F ROPEN on page 778
F CLOSE on page 776
F EOF on page 777
FM WRITE on page 783
Writes message variables to a binary file. For use with F_WOPEN and F_CLOSE. A message is written in the file as a null terminated string. This procedure makes a sequential access to the file, from the previous position. The first call after F_WOPEN writes the first string to the file, each call pushes the writing pointer.

Important: Avoid using this function. This function is intended to be obsolete.


## Arguments

| ID | DINT | File number: returned by F_WOPEN |
| :--- | :--- | :--- |
| IN | MESSAGE | Message value to be written in the file |
| OK | BOOL | Execution status: TRUE if successful |

## Example

(*FBD program using file management blocks *)

(* ST Equivalence: *)

```
file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);
```


## See also

## F WOPEN on page 778

F CLOSE on page 776
FM READ on page 782
Inserts a sub-string in a message string at a given position.


## Arguments

| IN | IN | MESSAGE | Initial string |
| :--- | :--- | :--- | :--- |
| Str | Str | MESSAGE | String to be inserted |
| Pos | Pos | DINT | Position of the insertion <br> the insertion is done before the position <br> (first valid position is 1) |

## Arguments

| INSERT | 0 | MESSAGE | Modified string <br> empty string if Pos $<=0$ <br> concatenation of both strings if Pos is greater than the <br> length of the IN string |
| :--- | :--- | :--- | :--- |

## Example

(* FBD Program using "INSERT" Function *)

(* ST Equivalence: *)
MyName := INSERT ('Mr JONES', 'Frank ', 4); (* MyName is 'Mr Frank JONES' *)

## See also

ASCII on page 771
DELETE on page 775
FIND on page 781
Extracts the left part of a message string. The number of characters to be extracted is given.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be extracted. This number <br> cannot be greater than the length of the IN string. |
| LEFT | 0 | MESSAGE | Left part of the $\operatorname{IN}$ string (its length $=\mathrm{NbC}$ ) <br> empty string if $\mathrm{NbC}<=0$ <br> complete $\operatorname{IN}$ string if $\mathrm{NbC}>=$ IN string length |

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'
*)
```


## See also

## CAT on page 750

MID on page 788
RIGHT on page 798
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.


| Arguments |  |  |  |
| :---: | :---: | :---: | :---: |
| MIN | MIN | DINT | Minimum allowed value |
| IN | IN | DINT | Any signed integer value |
| MAX | MAX | DINT | Maximum allowed value |
| LIMIT | 0 | DINT | Input value bounded to allowed range |

## 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
*)
```


## See also

MAX on page 787
MIN on page 789
Calculates the logarithm (base 10) of a real value.


## Arguments

| IN | IN | REAL | Must be greater than zero |
| :--- | :--- | :--- | :--- |
| LOG | 0 | REAL | Logarithm (base 10) of the input value |

## Example

(* FBD Program using "LOG" Function *)

(* ST Equivalence: *)

```
xpos := ABS (xval);
xlog := LOG (xpos);
```


## See also

ABS on page 766

Gives the maximum of two integer values.


## Arguments

| IN1 | IN1 | DINT | Any signed integer value |
| :--- | :--- | :--- | :--- |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MAX | 0 | DINT | Maximum of both input 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
*)
```


## See also

MIN on page 789
LIMIT on page 786
Extracts a part of a message string. The number of characters to be extracted and the position of the first character are given.


| Arguments |  |  |  |
| :--- | :--- | :--- | :--- |
| IN | IN | MESSAGE | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted cannot be greater <br> than the length of the IN string |
| Pos | Pos | DINT | Position of the sub-string <br> the sub-string first character will be the one pointed to by <br> Pos <br> (first valid position is 1) |

Arguments

| MID | Q | MESSAGE | Middle part of the string (its length $=$ NbC) <br> empty string if parameters are not valid |
| :--- | :--- | :--- | :--- |

## Example

(* FBD Program using "MID" Function *)

(* ST Equivalence: *)
sub_string := MID ('abcdefg', 2, 4);
(* sub string is 'de' *)

## See also

## DELETE on page 775

FIND on page 781
LEFT on page 785
RIGHT on page 798
Gives the minimum of two integer values.


| Arguments |
| :--- |
| IN1 |
|  |
| IN1 |
| IN2 |

## 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
*)
```


## See also

MAX on page 787
LIMIT on page 786

## MLEN

Calculates the length of a message string.


## Arguments

| IN | IN | MESSAGE | Any message string |
| :--- | :--- | :--- | :--- |
| MLEN | NbC | DINT | Number of characters in the IN 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 three characters on the
left of the string and places the result in the prefix
string variable.
Nothing is done if the string length is less than three
characters. *)
```


## See also

FIND on page 781

## LEFT on page 785

MID on page 788
RIGHT on page 798
Calculates the modulo of an integer value.


## Arguments

| IN | IN | DINT | Any signed integer value |
| :--- | :--- | :--- | :--- |
| Base | Base | DINT | Must be greater than zero |
|  | 0 | DINT | Modulo calculation (input MOD base) <br> returns -1 if Base $<=0$ |

## Example

(* FBD Program using "MOD" Function *)

(* ST Equivalence: *)

```
division_result := (value / divider); (* integer
division **)
rest_of_division := MOD (value, divider); (* rest of
the division *)
```


## See also

Division on page 751
Greater Than on page 752
Less Than on page 754
Subtraction on page 760

Multiplexer with four entries; selects a value between four integer values.


Arguments

| SEL | SEL | DINT | Selector integer value (must be in set [0..3]) |
| :--- | :--- | :--- | :--- |
| IN1...IN4 | IN1...IN4 | DINT | Any integer values |
| MUX4 | 0 | DINT | $=$ value1 if SEL $=0$ <br> $=$ value2 if SEL $=1$ <br> $=$ <br> value3 if SEL $=2$ <br> $=$ <br> value4 if SEL $=3$ <br> $=0$ for all other values of the selector |

## 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 *)
```


## See also

MUX8 on page 792
MUX8

Multiplexer with eight entries; selects a value between eight integer values.


## Arguments

| SEL | SEL | DINT | Selector integer value (must be in set [0..7]) |
| :--- | :--- | :--- | :--- |
| IN1...IN8 | IN1...IN8 | DINT | Any integer values |
| MUX8 | 0 | DINT | $=$ value1 if selector $=0$ <br> $=$ value2 if selector $=1$ <br> $\ldots$ <br> $=$ |
|  |  |  | value8 if selector $=7$ <br> $=0$ for all other values of the selector |

## 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 *)
```


## See also

## MUX4 on page 791

## NOT_MASK

## Example

(* FBD example with NOT_MASK Operators *)

(*ST equivalence: *)

```
result := NOT_MASK (16#1234);
```

(* result is 16\#FFFF_EDCB *)

## See also

AND MASK on page 767
OR MASK on page 795
XOR MASK on page 806
Tests the parity of an integer: result is odd or even.


## Arguments

| $\mathbb{N}$ | IN | DINT | Any signed integer value |
| :--- | :--- | :--- | :--- |
| Odd | 0 | DINT | TRUE if input value is odd <br> FALSE if input value is even |

## Example

(* FBD Program using "ODD" Function *)

(* ST Equivalence: *)

```
If Not (ODD (16#1234)) Then Return;
End_if;
value := 16#1234 + 1;
(* makes value always even *)
```


## See also

MOD on page 791
Integer OR bit-to-bit mask.


Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| MSK | MSK | DINT | Must have integer format |
| OR_MASK | Q | DINT | Bit-to-bit logical OR between IN and MSK |

## 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 *)
```


## See also

AND MASK on page 767
NOT MASK on page 794
XOR MASK on page 806
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.


| Arguments |  |  |  |
| :--- | :--- | :--- | :--- |
| IN | IN | REAL | Real number to be raised |
| EXP | EXP | REAL | Power (exponent) |
| POW | 0 | REAL | $($ IN EXP $)$ <br> 1.0 if $\operatorname{IN}$ is not 0.0 and EXP is 0.0 <br> 0.0 if $\operatorname{IN}$ is 0.0 and EXP is negative <br> 0.0 if both IN and EXP are 0.0 <br> 0.0 if $\operatorname{Nis}$ is negative and EXP does not correspond to an <br> integer |

## Example

(* FBD Program using "POW" Function *)

(* ST Equivalence: *)

```
result := POW (xval, power);
```


## See also

ABS on page 766
LOG on page 787
Gives a random integer value in a given range.


## Arguments

| base | base | DINT | Defines the allowed set of number |
| :--- | :--- | :--- | :--- |
| RAND | 0 | DINT | Random value in set $[0$. .base- -1$]$ |

## Example

(* FBD Program using "RAND" function *)

(* ST Equivalence: *)

```
selected := MUX4 ( RAND (4), 1, 2, 3, 4 );
(*
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 2 if 1 is issued from RAND,
or 3 if 2 is issued from RAND,
or 4 if 3 is issued from RAND,
*)
```


## See also

## MUX4 on page 791

MUX8 on page 792
Replaces a part of a message string by a new set of characters.


## Arguments

| IN | IN | MESSAGE | Any string |
| :--- | :--- | :--- | :--- |
| Str | Str | MESSAGE | String to be inserted (to replace NbC chars) |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first modified character <br> (first valid position is 1) |

Arguments

| REPLACE | 0 | MESSAGE | Modified string: <br> $\bullet$ <br>  <br> $\bullet$ <br> NbC characters are deleted at position Pos <br> returns emptring Str is inserted at this position if Pos $<=0$ <br> returns strings concatenation (IN + Str) if Pos is greater than the <br> length of the IN string <br> returns initial string $\operatorname{IN}$ if $\mathrm{NbC}<=0$ |
| :--- | :--- | :--- | :--- |

## Example

(* FBD program using "REPLACE" function *)

(* ST Equivalence: *)

```
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
```

(* MyName is 'Mr Frank JONES' *)

## See also

DELETE on page 775
LEFT on page 785
MID on page 788
RIGHT on page 798
Extracts the right part of a message string. The number of characters to be extracted is given.


## Arguments

| IN | IN | MESSAGE | Any non-empty string |
| :--- | :--- | :--- | :--- |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be <br> greater than the length of the IN string. |
| RIGHT | 0 | MESSAGE | Right part of the string (length $=$ NbC) <br> empty string if $\mathrm{NbC}<=0$ <br> complete string if $\mathrm{NbC}>=$ string length |

## 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'
*)
```


## See also

## LEFT on page 785

MID on page 788
CAT on page 750
Rotates the DINT type input by NbR bits to the left in a circular form and fills the bits on the right with the bits that are rotated.


Arguments

| IN | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| ROL | 0 | DINT | Left rotated value <br> no effect if $\operatorname{NbR}<=0$ |

## Example

(* FBD Program using "ROL" Function *)

(* ST Equivalence: *)

```
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result = 2#1001_1010_0110_1010*)
```


## See also

## ROR (Rotation Right) on page 800

SHL (Shift Left Arithmetic or Shift Left Signed) on page 801
SHR (Shift Right Arithmetic or Shift Right Signed) on page 802

## ROR (Rotation Right)

Rotates the DINT type input by NbR bits to the right in a circular form and fills the bits on the left with the bits that are rotated.


## Arguments

| $\mathbb{N}$ | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1.31]) |
| ROR | 0 | DINT | Right rotated value <br> no effect if NbR $<=0$ |

## Example

(* FBD Program using "ROR" Function *)

(* ST Equivalence: *)

```
result := ROR (register, 2);
```

```
(* register = 2#0011_0011_0010_1011_0011_0010_1001_1001
*)
(* result = 2#0100_1100_1100_1010_1100_1100_1010_0110
*)
```


## See also

ROL (Rotation Left) on page 799
SHL (Shift Left Arithmetic or Shift Left Signed) on page 801
SHR (Shift Right Arithmetic or Shift Right Signed) on page 802
Binary selector; selects a value between two integer values.


## Arguments

| SEL | SEL | BOOL | Indicates the chosen value |
| :--- | :--- | :--- | :--- |
| IN1, IN2 | IN1, IN2 | DINT | Any integer values |
| SEL | 0 | DINT | $=$ IN1 if SEL is FALSE <br> $=$ IN2 if SEL is TRUE |

## Example

(* FBD Program using "SEL" Function *)

(* ST Equivalence: *)

```
result := SEL (AutoMode, ManuCmd, InpCmd);
(* process command selection *)
```


## See also

ANA on page 748

## SHL (Shift Left Arithmetic or Shift Left Signed)

Shifts the 32 bits of an integer to the left and places a 0 in the least significant bit.


## Arguments

| IN | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| SHL | 0 | DINT | Left shifted value <br> no effect if NbS $<=0$ <br> 0 replaces the least significant bit |

## Example

(* FBD Program using "SHL" Function *)

(* ST Equivalence: *)

```
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```


## See also

$\underline{\text { ROL (Rotation Left) on page } 799}$
ROR (Rotation Right) on page 800
SHR (Shift Right Arithmetic or Shift Right Signed) on page 802
SHR (Shift Right Arithmetic or Shift Right Signed)

Shifts the 32 bits of an integer to the right and replicates the leftmost bit (significant bit) to fill the vacant bits.


Arguments

| IN | IN | DINT | Any integer value |
| :--- | :--- | :--- | :--- |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1.31]) |
| SHR | 0 | DINT | Right shifted value <br> no effect if NbS $<=0$ <br> the leftmost bit is replicated if NbS > $=1$ |

## Example

(* FBD Program using "SHR" Function *)

(* ST Equivalence: *)

```
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result = 2#1110_0110_1001_1010 *)
```


## See also

ROL (Rotation Left) on page 799
ROR (Rotation Right) on page 800
SHL (Shift Left Arithmetic or Shift Left Signed) on page 801
Calculates the Sine of a real value.


## Arguments

| IN | IN | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| SIN | 0 | REAL | Sine of the input value (in set $[-1.0 \ldots+1.0])$ |

## Example

(* FBD Program using "SIN" and "ASIN" Functions *)

(* ST Equivalence: *)

```
sine := SIN (angle);
```


## See also

ACOS on page 767
ASIN on page 771
ATAN on page 772
COS on page 773
TAN on page 804
Calculates the square root of a real value.


## Arguments

| IN | IN | REAL | Must be greater than or equal to zero |
| :--- | :--- | :--- | :--- |
| SQRT | Q | REAL | Square root of the input value |

## Example

(* FBD Program using "SQRT" Function *)

(* ST Equivalence: *)

```
xpos := ABS (xval);
xroot := SQRT (xpos);
```


## See also

ABS on page 766
Calculates the Tangent of a real value.


## Arguments

| IN | IN | REAL | Cannot be equal to PI/2 modulo PI |
| :--- | :--- | :--- | :--- |
| TAN | 0 | REAL | Tangent of the input value <br> $=1 \mathrm{E}+38$ for invalid input |

## Example

(* FBD Program using "TAN" and "ATAN" Functions *)

(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```


## See also

ACOS on page 767
ASIN on page 771
ATAN on page 772
COS on page 773
SIN on page 803
Truncates a real value to have just the integer part.


Arguments

| IN | IN | REAL | Any REAL value |
| :--- | :--- | :--- | :--- |
| TRUNC | 0 | REAL | If $\operatorname{IN}>0$, biggest integer less or equal to the input <br> If $I N<0$, least integer greater or equal to the input |

## Example

(* FBD Program using "TRUNC" Function *)

(* ST Equivalence: *)

```
result := TRUNC (+2.67) + TRUNC (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)
```


## See also

Addition on page 747
Integer exclusive OR bit-to-bit mask


Arguments

| IN | IN | DINT | Must have integer format |
| :--- | :--- | :--- | :--- |
| MSK | MSK | DINT | Must have integer format |
| XOR_MASK | 0 | DINT | Bit-to-bit logical Exclusive OR between IN and MSK |

## Example

(* FBD example with XOR_MASK functions *)

(* ST Equivalence: *)

```
crc32 := XOR_MASK (precrc, nextc);
result := XOR_MASK (16#012, 16#011); (* equals 16#003
*)
```


## See also

AND MASK on page 767
NOT MASK on page 794
OR MASK on page 795
These function blocks are supported:

| Function Blocks | Description |
| :--- | :--- |
| Alarms Management |  |
| LIM_ALRM | High/low limit alarm with hysteresis |
| Boolean Operations |  |
| SR | Set dominant bistable |
| RS | Reset dominant bistable |
| R_TRIG | Rising edge detection |
| F_TRIG | Falling edge detection |
| Comparators |  |

Chapter 3 Working with Trusted Applications

| Function Blocks | Description |
| :--- | :--- |
| CMP | Full comparison function block |
| Counters | Up counter |
| CTU | Down counter |
| CTD | Up-down counter |
| CTUD | Running average over N samples |
| Data Manipulation | Manipulates a software semaphore |
| AVERAGE |  |
| Internal Operations | Differentiation according to time |
| SEMA | Boolean hysteresis on difference of reals |
| Process Control | Integration over time |
| DERIVATE | Stack of integer |
| HYSTER | Blinking Boolean signal |
| INTEGRAL | Signal generator |
| STACKINT |  |
| Signal Generation | On-delay timing |
| BLINK | Off-delay timing |
| SIG_GEN | Pulse timing |
| Time Operations |  |

Call new function blocks from any language.

## See also

## Operators on page 745

## Functions on page 765

Stores a value at each cycle and calculates the average value of all stored values. Stores only the latest N values.

The maximum number of samples $N$ is 128 . When $N$ exceeds 128 , the number of samples is truncated to 128 . If the "RUN" command is FALSE (reset mode), the output value is equal to the input value. Upon reaching the maximum $N$ of stored values, the first stored value is overwritten with the latest value.
Tip: When setting or changing the value for $N$, set RUN to FALSE, then set it back
to TRUE.


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 |

## Example

(* FBD program using the AVERAGE block: *)

(* ST Equivalence: AVERAGE1 instance of AVERAGE block *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;
```


## See also

## AND on page 748

Generates a blinking signal.


## Arguments

| RUN | BOOL | Mode: TRUE=blinking / FALSE=reset the output to false |
| :--- | :--- | :--- |
| CYCLE | TIME | Blinking period. Possible values range from Oms to <br> 23h59m59s999ms. |
| $\square$ | BOOL | Output blinking signal |



## See also

SIG GEN on page 818
Compares two values: tells if they are equal, or if the first is less or greater than the second one.


Arguments

| VAL1 | DINT | Any signed integer value |
| :--- | :--- | :--- |
| VAL2 | DINT | Any signed integer value |
| LT | BOOL | TRUE if vall is less than val2 |
| EO | BOOL | TRUE if vall is equal to val2 |
| GT | BOOL | TRUE if vall is greater than val2 |

## Example

(* FBD program using the 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);

## See also

ANA on page 748
AND on page 748
Greater Than or Equal on page 753
Counts (integers) from a given value down to 0 in increments of 1.

| Important: | The CTD block does not detect the rising edges or falling edges of <br> the counting input (CD). The block must be associated with an |
| :--- | :--- |
|  | "R_TRIG" or "F_TRIG" block to create a pulse counter. |



## Arguments

| CD | BOOL | Counting input <br> (down-counting when CD is TRUE) |
| :--- | :--- | :--- |
| LOAD | BOOL | Load command (dominant) <br> (CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed initial value |
| Q | BOOL | Underflow: TRUE when CV $<=0$ |
| CV | DINT | Counter result |

## Example

(* FBD program using the 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;
```


## See also

CTU on page 810
CTUD on page 811
F TRIG on page 813
R TRIG on page 816
Counts (integers) from 0 up to a given value in increments of 1.

| Important: | The CTU block does not detect the rising edges or falling edges of <br> the counting input (CU). The block must be associated with an |
| :--- | :--- |
|  | "R_TRIG" or "F_TRIG" block to create a pulse counter. |



## 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 |

## Example

(* FBD program using the CTU block *)

(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block and CTU1 is an instance of CTU block*)

```
F_TRIG1 (command);
CTU1(F_TRIG1.Q,NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```


## See also

CTD on page 809
CTUD on page 811
F TRIG on page 813
R TRIG on page 816
Counts (integers) from 0 up to a given value in increments of 1 or from a given value down to 0 in increments of 1 .


Arguments

| CU | BOOL | Up-counting (when CU is TRUE) |
| :--- | :--- | :--- |
| CD | BOOL | Down-counting (when CD is TRUE) |
| RESET | BOOL | Reset command (dominant) <br> (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 |

## Example

(* FBD program using the 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;
```


## See also

CTD on page 809
CTU on page 810
F TRIG on page 813
R TRIG on page 816

## DERIVATE

Differentiation of a real value.
If the CYCLE parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period uses the real duration of the cycle time.


| Arguments |  |  |
| :--- | :--- | :--- |
| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| XIN | REAL | Input: any real value |
| CYCLE | TIME | Sampling period. Possible values range from Oms to <br> 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

## Example

(* FBD program using the DERIVATE block: *)

(* ST Equivalence: DERIVATE1 instance of DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, T#100ms);
derivated_value := DERIVATE1.XOUT;
```


## See also

HYSTER on page 814
INTEGRAL on page 814
STACKINT on page 820
Detects a falling edge of a Boolean variable


## Arguments

| CLK | BOOL | Any Boolean variable |
| :--- | :--- | :--- |
| 0 | BOOL | TRUE when CLK changes from TRUE to FALSE <br> FALSE if all other cases |

## Example

(* FBD program using the F_TRIG block *)

(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block *)

```
F_TRIG1(cmd);
nb_edge := ANA(F_TRIG1.Q) + nb_edge;
```


## See also

R TRIG on page 816
RS on page 817

## SR on page 819

## HYSTER

Hysteresis on a real value for a high limit.


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) |
| 0 | BOOL | TRUE if XIN1 has overpassed XIN2+EPS and is not yet below <br> XIN2-EPS |

## Example timing diagram



## See also

DERIVATE on page 812
INTEGRAL on page 814
STACKINT on page 820
INTEGRAL
Integrates a real value.


## Arguments

| RUN | BOOL | Determines the mode: <br> $\bullet$ TRUE $=$ integrate <br> $\bullet$ FALSE $=$ hold |
| :--- | :--- | :--- |
| R1 | BOOL | Resets the override |
| XIN | REAL | Uses any real value |
| XO | REAL | Defines the initial value |

## Arguments

| CYCLE | TIME | The sampling period where the value ranges from Oms to 23h59m59s999ms. <br> If the CYCLE parameter value is less than the real duration of the cycle time in <br> the virtual machine, the sampling period uses the real duration of the cycle <br> time. |
| :--- | :--- | :--- |
| O | BOOL | (output) Determines if R1 reset the override: <br> TRUE $=$ R1 is FALSE <br> FALSE $=$ R1 is TRUE |
| XOUT | REAL | (output) Results in the integrated output value |

## Function Block Diagram


(* ST Equivalence: INTEGRAL1 instance of INTEGRAL block *)
INTEGRAL1 (manual_mode, NOT (manual_mode), sensor_value, init_value, T\#100ms);
controlled_value $:=$ INTEGRAL1.XOUT;

## See also

DERIVATE on page 812

## HYSTER on page 814

STACKINT on page 820
Hysteresis on a real value for high and low limits.
A hysteresis is applied on high and low limits. The hysteresis delta used for either the high or low limit is equal to the EPS parameter.


Arguments

| $H$ | REAL | High limit value |
| :--- | :--- | :--- |
| X | REAL | Input: any real value |
| L | REAL | Low limit value |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| OH | BOOL | "high" alarm: TRUE if $X$ above high limit $H$ |
| O | BOOL | Alarm output: TRUE if $X$ out of limits |
| QL | BOOL | "low" alarm: TRUE if $X$ below low limit $L$ |

Example timing diagram


R_TRIG
Detects a rising edge of a Boolean variable.


Arguments

| CLK | BOOL | Any Boolean variable |
| :--- | :--- | :--- |
| 0 | BOOL | TRUE when CLK rises from FALSE to TRUE <br> FALSE in all other cases |

## Example

(* FBD program using the R_TRIG block *)

(* ST Equivalence: We suppose R_TRIG1 is an instance of the R_TRIG block *) R_TRIG1 (cmd); nb_edge := ANA(R_TRIG1.Q) + nb_edge;

## See also

## F TRIG on page 813

RS on page 817
SR on page 819
Reset dominant bistable.


| Arguments |  |  |
| :--- | :--- | :--- |
| SET | BOOL | If TRUE, sets Q1 to TRUE |
| RESET1 | BOOL | If TRUE, resets Q1 to FALSE (dominant) |
| Q1 | BOOL | Boolean memory state |


| Set | Reset1 | $\mathbf{0 1}$ | Result 01 |
| :--- | :--- | :--- | :--- |
| 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


(* ST Equivalence: We suppose RS1 is an instance of RS block *)
RSI (start_cmd, (stop_cmd OR alarm));
command $:=$ RS1.Q1;

## See also

## F TRIG on page 813

R TRIG on page 816
SR on page 819

SEMA

## Example

(* "x" is a Boolean variable initialized to FALSE *)

```
busy := x;
    If claim Then
        x := True;
    Else
        If release Then
            busy := False;
            x := False;
        End_if;
    End_if;
```

Generates various signal: blink on a Boolean, an integer counter-up, and real sine wave.

When counting reaches maximum value, counting restarts from 0 (zero). So END keeps the TRUE value only during 1 PERIOD.


Arguments

| RUN | BOOL | Mode: TRUE=running / FALSE=reset to false |
| :--- | :--- | :--- |
| PERIOD | TIME | Duration of one sample. Possible values range from Oms to <br> 23h59m59s9g9ms. |
| 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) |

## Timing diagram



## See also

## BLINK on page 808

Set dominant bistable.


Arguments

| SET1 | BOOL | If TRUE, sets Q1 to TRUE (dominant) |
| :--- | :--- | :--- |
| RESET | BOOL | If TRUE, resets Q1 to FALSE |
| 01 | BOOL | Boolean memory state |


| Set | Reset1 | $\mathbf{0 1}$ | Result 01 |
| :--- | :--- | :--- | :--- |
| 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


(* ST Equivalence: We suppose SR1 is an instance of SR block *)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```


## See also

F TRIG on page 813
R TRIG on page 816
RS on page 817
Manages 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.

The OFLO value is valid only after a reset ( $R 1$ has been set to TRUE at least once and back to FALSE).


Arguments

| PUSH | BOOL | Push command (on rising edge only) <br> add the IN value on the top of the stack |
| :--- | :--- | :--- |
| POP | BOOL | Pop command (on rising edge only) <br> 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 |

## Example

(* FBD program using the STACKINT block: error management *)

(* ST Equivalence: We suppose STACKINT1 is an instance of STACKINT block *)

```
STACKINT1(err detect, acknowledge, manual mode,
err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
```


## See also

## DERIVATE on page 812

## HYSTER on page 814

INTEGRAL on page 814
Increases an internal timer up to a given value.


| Arguments |  |  |
| :--- | :--- | :--- |
| IN | BOOL | When detecting a falling edge, starts increasing the internal timer. <br> When detecting a rising edge, stops and resets the internal timer. |
| PT | TIME | This input defines the maximum programmed time. |
| O | BOOL | If $O$ is TRUE = total time is not elapsed. |
| ET | TIME | This output determines the current elapsed time. |

Timing diagram


## See also

TON on page 822
TP on page 823
Increases an internal timer up to a given value.


## Arguments

| IN | BOOL | When detecting a rising edge, starts increasing the internal <br> timer. <br> When detecting a falling edge, stops and resets the internal <br> timer. |
| :--- | :--- | :--- |
| PT | TIME | Defines the maximum programmed time |
| O | BOOL | If $Q$ is TRUE = programmed time is elapsed. |
| ET | TIME | (output) Determines the current elapsed time. The possible <br> values range from Oms to 23h59m59s999ms. |

## Timing diagram



## See also

TOF on page 821
TP on page 823
Increases an internal timer up to a given value.


## Arguments

| IN | BOOL | When detecting a rising edge, starts increasing the internal <br> timer (if not already increasing). <br> If FALSE and only if the timer elapses, resets the internal timer. <br> Any change on IN during counting has no effect. |
| :--- | :--- | :--- |
| PT | TIME | Defines the maximum programmed time |
| ET | BOOL | If Q is TRUE = timer is counting. |
| TIME | (output) Determines the current elapsed time. The possible <br> values range from Oms to 23h59m59s999ms. |  |

## Timing diagram



## See also

TOF on page 821
TON on page 822
These functions and function blocks are available with the library. Access these functions and function blocks from the Block Selector or Block Library:

| Function or Function <br> Block | Description |
| :--- | :--- |
| AVG | This function calculates the average between 1 and 10 values. |
| BCD | Binary-Coded Decimal (BCD) translation of a 4-bit value. |
| CLAMP | This function block clamps the input process variable to the defined range. |
| DEV_AL | This function block is a deviation alarm indicating the deviation of the process <br> variable away from the set point. |


| Function or Function Block | Description |
| :---: | :---: |
| DIPT | This function block manages the Dual DI Termination Panel autotest sequences. Parameter definitions and additional information are available from the DIPT function block located in the Global Library. |
| ERRSTAT | Several process control functions and function blocks may generate error conditions causing ERRSTAT counters to increment. |
| FL2REGH | This function extracts the top 16 bit values of a 32 bit floating-point number. |
| FL2REGL | This function extracts the bottom 16-bit values of a 32-bit floating-point number. |
| HVS | This function selects the highest value between two process variables. |
| IPID | This function block is a process regulator that uses the feedback concept to regulate the process variable according to the difference between its current value and the value of the set point. |
| ITSTM | This function block manages the Dual DI Termination Panel autotest sequences. Parameter definitions and additional information are available from the ITSTM function block located in the Global Library. |
| LFLT | This function block detects line faults for CS300 Al modules. Parameter definitions and additional information are available from the LFLT function block located in the Global Library. |
| LVS | This function selects the lowest value between two process variables. |
| MANTRK | This function block enables manual set-point tracking. |
| MASSFLOW | The function provides the mass flow corrected for the current operating conditions. |
| MVS | This function selects the median value between three process variables. |
| OTSTM | This function block manages the Dual DO Termination Panel autotest sequences. Parameter definitions and additional information are available from the OTSTM function block located in the Global Library. |
| PACK16 | This function block packs 16 boolean variables into a 16 -bit register. Parameter definitions and additional information are available from the PACK16 function block located in the Global Library. |
| PC_DRUM | This function block provides the pressure compensated drum level. |
| PID_II | This function block is a process regulator that uses the feedback concept to regulate the process variable according to the difference between its current value and the value of the set point. |
| R_SEL | This function selects between two REAL analog values. |
| RATE | This function block detects when the absolute rate of change per second of the input process variable, over the specified time interval, exceeds the maximum rate of change. |
| REG2FL | This function combines the top and bottom 16-bit values of two integers to produce a 32-bit floating point number. |
| RMET | This function block manages the Dual DO Termination Panel autotest sequences. Parameter definitions and additional information are available from the RMET function block located in the Global Library. |
| SCALE | This function block linearly scales the input value according to the low input (IPmin), high input (IPmax), low output (OUTmin) and high output (OUTmax) values. |
| SLEW | This function block tracks the process variable with the output. |
| SORTX | Within a specified range, this function calculates the square root of a REAL value as a percentage. |
| T_AVG | This function block calculates the average value of the process variable over the specified time period. |
| UNPACK16 | This function block unpacks a 16 -bit register into 16 boolean variables. Parameter definitions and additional information are available from the UNPACK16 function block located in the Global Library. |
| VTEST | This function block provides a simplified interface to the valve test functionality of the T8449 Trusted ${ }^{\oplus}$ TMR 24 Vdc Valve Monitor module. This module uses the VTEST function block to initiate a valve test and records the results. <br> For additional information, see <br> http:// literature.rockwellautomation.com/idc/groups/literature/documents/rm/icstt-rm248_-en-p.pdf. |

## See also

Operators on page 745
Functions on page 765
Function Blocks on page 806
Calculates the average between 1 and 10 values.


Parameter values for the AVG function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| N | DINT | Specifies the number of values to average. <br> For example, when $\mathrm{N}=3$, returns the average of PV1, PV2, and PV3. |
| PV1...PV10 | REAL | These inputs use REAL values. Infinity or Not a Number (NaN) values are <br> not supported by the inputs. |
| OUT | REAL | Average value of the specified number of inputs. |

When the value of N is out of range, OUT returns the average of all ten input values and the ERRSTAT Param error count increments.

When an overflow errors occurs during execution of the function, OUT sets to PV1 and the ERRSTAT Overflow error counter increments.

When an underflow error occurs, OUT sets to zero and the ERRSTAT Underflow error counter increments.

Function Block Diagram Example


## Structured Text Example

```
AverageV := AVG(4, PV1, PV2, PV3, PV4, PV5, PV6, PV7,
PV8, PV9, PV10);
```


## See also

Error Statistics (ERRSTAT) on page 829

## BCD Translation (BCD)

Binary-Coded Decimal (BCD) translation of a 4-bit value.
OUT $=(B C D(I P 0, I P 1, I P 2, I P 3) * M)+N$


Parameter values for the BCD function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| IPO | BOOL | Input parameter with least significant bit |
| IP1 | BOOL | Input parameter |
| IP2 | BOOL | Input parameter |
| IP3 | BOOL | Input parameter with most significant bit |
| M | DINT | Value multiplier. <br> This value must be a power of 10. |
| N | DINT | Integer value added to the BCD conversion value, allowing the cascade of <br> several BCD functions. |
| OUT | DINT | Resulting decimal digit. |

When N is -1 , the function assumes that an up-stream conversion error occurs and returns -1 , causing all chained BCD functions to generate values of -1 . Conversion errors result from any of these conditions:

- IPO, IP1, IP2, or IP3 values representing an invalid BCD number
- M values which are not powers of 10
- Overflow conditions, where the ERRSTAT Overflow error counter increments

The value of OUT for each combination of IPO, IP1, IP2, and IP3 when $\mathrm{N}=0$ and $\mathrm{M}=1$ :

| IP3 | IP2 | IP1 | IPO | OUT |
| :--- | :--- | :--- | :--- | :--- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | -1 (conversion error) |
| 1 | 0 | 1 | 1 | $-1($ conversion error) |
| 1 | 1 | 0 | 0 | -1 (conversion error) |
| 1 | 1 | 0 | 1 | -1 (conversion error) |
| 1 | 1 | 1 | 0 | -1 (conversion error) |
| 1 | 1 | 1 | 1 | -1 (conversion error) |

Function Block Diagram Example


## Structured Text Example

BCD_value := BCD(Val1, Val2, Val3, Val4, 10, 2);

## See also

OUT = Min(Max(PV, IPmin), IPmax)


Parameter values for the CLAMP function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| IPmin | REAL | Minimum input range parameter. Infinity or Not a Number (NaN) values are <br> not supported by this input. |
| PV | REAL | Process variable having a REAL value |
| IPmax | REAL | Maximum input range parameter. Infinity or Not a Number (NaN) values <br> are not supported by this input. |
| OUT | REAL | When PV is < IPmin or > IPmax, sets the IPmin or IPmax value <br> respectively. <br> When PV is within range, sets to PV. |
| LIMIT | BOOL | TRUE when clamped, that is, LIMIT = (PV < IPmin) or (PV > IPmax) <br> FALSE when PV is within range of IPmin and IPmax. |

When IPmax < IPmin, the function block assumes that the values are reversed and swaps them. The ERRSTAT Reversed error counter increments each time the function swaps the parameters.

When PV is Not a Number (NaN) value, OUT sets to IPmin, LIMIT to FALSE and the ERRSTAT Param error counter increments.

## Function Block Diagram Example



## Structured Text Example

```
CLAMP(MinValue, Input, Maxvalue);
OutValue := CLAMP.OUT;
Result := CLAMP.LIMIT;
```


## See also

Error Statistics (ERRSTAT) on page 829
Deviation alarm indicating the deviation of the process variable (PV) away from the set point (SP).

DEVhi $=(($ PV $-S P)>=$ DEVus $)$

DEVlo $=((S P-P V)>=$ DEVds $)$


Parameter values for the DEV_AL function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Input process variable. <br> This value must be a REAL value. Infinity or Not a Number (NaN) values are <br> not supported by this input. |
| SP | REAL | Set point value. <br> This value must be a REAL value. Infinity or Not a Number (NaN) values are <br> not supported by this input. |
| DEVus | REAL | Up-scale deviation parameter. <br> This value must be a REAL value. Infinity or Not a Number (NaN) values are <br> not supported by this input. |
| DEVds | REAL | Down-scale deviation parameter. <br> This value must be a REAL value. Infinity or Not a Number (NaN) values are <br> not supported by this input. |
| DEVhi | BOOL | TRUE when PV > SP by at least DEVus. |
| DEVIo | BOOL | TRUE when PV < SP by at least DEVds. |

When an overflow condition occurs during execution, DEVhi sets to PV $>0$, DEVlo to PV <0, and the ERRSTAT Overflow error counter increments.

## Function Block Diagram Example



## Structured Text Example

```
DEV_AL(Process1, SetPoint, upV, DownV);
High := DEV_AL.DEVhi
Low := DEV_AL.DEVlo
```


## See also

Error Statistics (ERRSTAT) on page 829
Several process control functions and function blocks may generate error conditions causing these ERRSTAT counters to increment: reversed
parameters, incorrect parameter values, overflow conditions, underflow conditions, and divide by zero.


Parameter values for the ERRSTAT function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| RESET | BOOL | When TRUE, resets the error counters to zero. |
| Reversed | DINT | Reversed parameters error counter. The count increments on each <br> execution of the function when the parameters are reversed. <br> The initial value is zero. |
| Param | DINT | Incorrect parameter values error counter. The count increments on each <br> execution of the function when the parameter value is incorrect. <br> The initial value is zero. |
| Overflow | Overflow condition error counter. The count increments on each <br> execution of the function when the overflow condition occurs. <br> The initial value is zero. |  |
| Underflow | DINT | Underflow condition error counter. The count increments on each <br> execution of the function when the underflow condition occurs. <br> The initial value is zero. |
| ZeroDivide | DINT | Divide by zero error counter. The count increments on each execution of <br> the function when the divide by zero error occurs. <br> The initial value is zero. |
| Tip: | Counter values do not wrap to zero. Counter overflow conditions return a |  |
| value of -1. |  |  |

## Function Block Diagram Example



## Structured Text Example

```
ERRSTAT(Reset_count);
Reversed_count := ERRSTAT.Reversed;
IncorrectParam_count := ERRSTAT.Param;
Overflow_count := ERRSTAT.Overflow;
Underflow_count := ERRSTAT.Underflow;
ZeroDivide_COUNT := ERRSTAT.ZeroDivide;
```


## See also

AVG on page 825
SCALE on page 846
SLEW on page 847
Square Root Extraction (SQRTX) on page 848

## FL2REGH

Extracts the top 16 bit values of a 32 bit floating-point number. FL2REGH enables sending float numbers over Modbus as a pair of 16 -bit word values. The Modbus protocol does not support floats.


Parameter values for the FL2REGH function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| Val | REAL | 32-bit floating-point REAL value. |
| 0 | DINT | Integer value representing the top 16-bits of the 32-bit floating point <br> number. |
| Tip:FL2REGH does not convert floats to integers. Numbers remain in floating <br> point format. To convert the input value to an integer, use the ANA operator. |  |  |

## Function Block Diagram Example



## Structured Text Example

```
Output := Fl2RegH(floating_value) ;
```


## See also

ANA on page 748
FL2REGL on page 831
REG2FL on page 845
Extracts the bottom 16-bit values of a 32-bit floating-point number. FL2REGL enables sending float numbers over Modbus as a pair of 16 -bit word values. The Modbus protocol does not support floats.


Parameter values for the FL2REGL function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| $V a l$ | REAL | 32-bit floating-point REAL value. |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| 0 | DINT | Integer value representing the bottom 16-bits of the 32-bit floating point <br> number. |
| Tip: | FL2REGL does not convert floats to integers. Numbers remain in floating <br> point format. To convert the input value to an integer, use the ANA operator. |  |

## Function Block Diagram Example



## Structured Text Example

```
OutputL := Fl2RegL(floating real) ;
```


## See also

ANA on page 748
FL2REGH on page 831
REG2FL on page 845
Selects the highest value between two process variables.
OUT = Max(PV1, PV2)


Parameter values for the HVS function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV1 | REAL | This input must use any REAL value |
| PV2 | REAL | This input must use any REAL value |
| OUT | REAL | Highest REAL value between PV1 and PV2 |
| SEL | BOOL | FALSE when PV1 is the highest value. <br> TRUE when PV2 is the highest value. |

When PV1 or PV2 is infinity or Not a Number (NaN) value, OUT sets to PV1, SEL to FALSE and the ERRSTAT Param error counter increments.

## Function Block Diagram Example



## Structured Text Example

```
HVS(Process1, Process2);
OutValue := HVS.OUT;
Result := HVS.SEL;
```


## See also

Error Statistics (ERRSTAT) on page 829
Low Value Select (LVS) on page 836
Median Value Select (MVS) on page 839
Process regulator that uses the feedback concept to regulate the process variable according to the difference between its current value and the value of the set point. PID achieves calculation by a method of sampling.


Parameter values for IPID function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Process variable. This value must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| SP | REAL | Set point. This value must be a REAL value. Infinity or Not a Number (NaN) <br> values are not supported by this input. |
| Auto | Determines whether IPID operates in AUTO or MANUAL mode. <br> - TRUE for AUTO mode <br> - FALSE for MANUAL mode |  |
| XO | REAL | Adjustment value. This value must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. <br> In MANUAL mode, Xout = X0 |
| Kp | Proportionality constant. This value must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |  |
| Ti | Integral time constant. <br> Possible values range from Oms to 23hr59m59s999ms. |  |
| Td | Derivative time constant. <br> Possible values range from Oms to 23hr59m59s999ms. |  |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| Ts | TIME | Minimum sample period. <br> If Ts is less than the application scan time, uses the application scan time <br> as the effective sample interval. <br> The maximum effective sample period is Ts plus the application scan <br> time. <br> Possible values range from Oms to 23hr59m59s999ms. |
| Tdf | TIME | Derivative time constant. <br> Possible values range from Oms to 23hr59m59s999ms. |
| Xmax | REAL | Maximum value of Xout. This value must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| Xmin | REAL | Minimum value of Xout. This value must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| INCstop | BOOL | Increment stop. |
| DECstop | BOOL | Decrement stop. |
| Derr | BOOL | Derivative action switch. <br> TRUE = error <br> FALSE $=$ PV |
| Fwd | BOOL | Gain action. <br> TRUE $=$ forward acting |
| Xout | REAL | IPID function block output. <br> Action required on the system to keep the regulation. Xout recalculates <br> when taking each sample. |
| LIMIThi | BOOL | TRUE if Xout clamps at the maximum value of Xmax. |
| LIMITIo | BOOL | TRUE if Xout clamps at the minimum value of Xmin. |
| DQP | REAL | REAL |
| DOI | Delta output due to proportional action |  |
| DQD | Delta output due to integral action |  |
| DXout | REAL | Delta output due to derivative action |
| Total delta output | Loss of precision and underflow may occur. |  |

When Xmax < Xmin, the function assumes that the values are reversed and swaps the values. The Reversed error counter increments each time the function has to swap the parameters.

When overflow occurs during execution, the stored error and delta terms set to zero.

In AUTO mode:

- Xout sets to zero
- LIMIThi sets to FALSE
- LIMITlo sets to FALSE

In MANUAL mode:

- Xout sets to XO clamped to the range Xmin to Xmax
- LIMIThi sets to XO > Xmax
- LIMITlo sets to XO < Xmin


## AUTO Mode Operation

When a new sample is made and Auto = TRUE, calculates the output Xout:

```
Xout = Xout + Kp * ( dE[t] + I[t]/Ti + fddD[t] * Td /
```

T)
where

```
E[t] = SP - PV (error)
dE[t] = E[t] - E(t-1) (delta error)
I[t] = ( E[t] * T ) (integral term)
fddD[t] = ddD[t] filtered, first order lag, time
constant Tdf
ddD[t] = dE[t] - dE[t-1] if Derr = true (delta delta
error)
ddD[t] = dPV[t-1] - dPV[t] if Derr = false (delta
delta PV)
dPV[t] = PV[t] - PV[t-1] (delta PV)
```

$\mathrm{E}[t-1]$ is the stored error value. This is the value during the last sample.
$\mathrm{PV}[\mathrm{t}-1]$ is the stored process variable. This is the value during the last sample.
T is the elapsed time in milliseconds since the last sample.
The results of calculation are given as outputs:

```
DQP = Kp * dE[t] (delta proportional
term)
DQI = Kp * I[t]/Ti (delta integral term)
DQD = KP * fddD[t] * Td / T (delta derivative
term)
DXout = DQP + DQI + DQD (delta total output)
```

When $\mathrm{Ti}<10 \mathrm{~ms}$, the integral term sets to zero to avoid excessive action.
Using the Fwd input, swap the action between reverse and direct acting. When Fwd = False, a positive Kp gives a rising output on falling PV (reverse action). When Fwd = True, a positive Kp gives a falling output on falling PV (direct action). Setting a negative value to Kp also inverts the action.

After calculating Xout, applies the increment stop (INCstop), decrement stop (DECstop) and the clamp (Xmax and Xmin). When INCstop = TRUE and Xout has increased since the last sample or if DECstop = TRUE and Xout has decreased since the last sample, Xout sets to its previous value. When Xout > Xmax or Xout < Xmin:

- Xout clamps to the range Xmin to Xmax
- LIMIThi sets to Xout > Xmax
- LIMITlo sets to Xout < Xmin

When INCstop, DECstop or the clamp are active, the output Xout stays at its last value. The results of calculation (DQP, DQI, DQD, DXout) continue updating. On initialization, INCstop and DECstop have no effect.

## MANUAL Mode Operation

For new samples where Auto = FALSE, the output Xout immediately follows the adjustment value (XO) clamped to the range Xmin to Xmax. LIMIThi sets to XO > Xmax and LIMITlo sets to XO < Xmin. The stored error term sets to (SP -

PV) and the stored delta terms set to zero. The increment and decrement stop (INCstop and DECstop) have no effect.

## Function Block Diagram Example

(*FBD example with IPID function block*)


## Structured Text Example

```
IPID(Process1, 25.5, Mode, Adjust, Proportion,
IntegralT, DerivativeT, T#20ms, ConstantT, 101.2, 91.8,
Increment, Decrement, ActionSwitch, GainAction);
Result:= PID_II.Xout;
High := PID_II.LIMIThi;
Low := PID_II.LIMITlo;
DeltaP := IPID.DQP;
DeltaI := IPID.DQI;
DeltaD := IPID.DQD;
DeltaTotal := IPID.DXout;
```


## See also

## Error Statistics (ERRSTAT) on page 829

Selects the lowest value between two process variables.
OUT = Min(PV1, PV2)


Parameter values for the LVS function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV1 | REAL | Uses any REAL value |
| PV2 | REAL | Uses any REAL value |
| OUT | REAL | Lowest REAL value between PV1 and PV2 |
| SEL | BOOL | FALSE when PV1 is the lowest value. <br> TRUE when PV2 is the lowest value. |

When PV1 or PV2 is infinity or Not a Number (NaN) value, OUT sets to PV1, SEL to FALSE and the ERRSTAT Param error counter increments.

## Function Block Diagram Example



## Structured Text Example

```
LVS(Process1, Process2);
OutValue := LVS.OUT;
Result := LVS.SEL;
```


## See also

Error Statistics (ERRSTAT) on page 829
Median Value Select (MVS) on page 839
High Value Select (HVS) on page 832
Enables manual set-point tracking.


Parameters for the MANTRK function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| Xout | REAL | PID_Il function block output. |
| XO | REAL | Manual set-point input. |
| AUTO | BOOL | Automatic (TRUE) or manual (FALSE) control mode. |
| XOout | REAL | When AUTO is TRUE, XOout sets to the current value of XOut. <br> When AUTO changes from TRUE to FALSE, XOout remains at the previous <br> value until the value of the manual set-point input (XO) changes. When the <br> XO value then changes, XOout sets to the value of XO. |

Function Block Diagram Example


## Structured Text Example

```
pid_ii_1(Process1, 25.5, Mode, Adjust, Proportion,
IntegralT, DerivativeT, T#20ms, 101.2, 91.8, Increment,
Decrement);
Tracking := mantrk_1(pid_ii_1.Xout, 25.0, ControlMode);
```


## See also

PID II on page 840

## MASSFLOW

Provides the mass flow corrected for the current operating conditions.

$$
\text { OUT }=(\mathrm{C} * \mathrm{Z} * \mathrm{~m} * \mathrm{e} * \operatorname{sqrt(DP)}) *(\text { ro * }((\mathrm{P}+14.7) /(\mathrm{T}+273)))
$$



Parameter values for the MASS_FLOW function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| $D P$ | REAL | Differential pressure |
| $P$ | REAL | Pressure |
| $T$ | REAL | Temperature, in Celsius. |
| $C$ | REAL | Mass |
| $Z$ | REAL | Compressibility |
| $m$ | REAL | Mass, in kg. |
| $e$ | REAL | Mass |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| ro | REAL | Plank's constant |
| OUT | REAL | Mass flow, in kg/hr. <br> When an overflow, underflow, or divide by zero error occurs during the <br> execution of the function, OUT returns a value of -1 and the corresponding error <br> counter increments. |

## See also

## MANTRK on page 837

RATE on page 844

## Median Value Select (MVS)

Selects the median value between three process variables.


Parameter values for the MVS function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV1 | REAL | Uses any REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. |
| PV2 | REAL | Uses any REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. |
| PV3 | Uses any REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. |  |
| DEVIATION | REAL | Maximum deviation between PV1, PV2, and PV3. Infinity or Not a Number <br> (NaN) values are not supported by this input. |
| OUT | Mean value when the deviation between PV1, PV2, and PV3 is less than <br> DEVIATION. <br> Median value when the deviation between PV1, PV2, and PV3 is greater than <br> DEVIATION. |  |
| LIMIT | BOOL | FALSE when the deviation between PV1, PV2, and PV3 is less than <br> DEVIATION. <br> TRUE when the deviation between PV1, PV2, and PV3 is greater than <br> DEVIATION. |

When an overflow error occurs during the execution of the function block, OUT is set to the median value, LIMIT to TRUE and the ERRSTAT Overflow error counter increments.

## Function Block Diagram Example



## Structured Text Example

```
MVS(PlantA, PlantB, PlantC, 1.5);
Value := MVS.OUT;
Dev := MVS.LIMIT;
```


## See also

Error Statistics (ERRSTAT) on page 829
Low Value Select (LVS) on page 836
High Value Select (HVS) on page 832
Provides the pressure compensated drum level. PC_DRUM also compensates for the reverse acting output of the level input (increasing DP value of decreasing differential pressure).
OUT $=(101-(5.43 e-3 * D P)-L V) /(0.88-(0.173 e-3 * D P))$


Parameter values for the PC_DRUM function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| DP | REAL | Differential pressure, in kilopascals. |
| LV | REAL | Level, in percentage, of full-scale. |
| OUT | REAL | Pressure compensated drum level. Returns a percentage <br> value from 0.0 to 100.0. |

When an underflow, overflow, or zero divide error occurs during execution, OUT sets to zero and the the corresponding error counter increments.

## Example

(*FBD example with PC_DRUM function block*)

(*ST example*)

```
OUT := PC_DRUM(DP, LV);
```

PID_II

Process regulator that uses the feedback concept to regulate the process variable according to the difference between its current value and the value of the set point. PID achieves calculation by a method of sampling.


Parameter values for the PID_II function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Process variable. Must be a REAL value. Infinity or Not a Number <br> (NaN) values are not supported by this input. |
| SP | REAL | Set point. Must be a REAL value. Infinity or Not a Number (NaN) <br> values are not supported by this input. |
| Auto | Determines whether PID_Il operates in AUT0 or MANUAL mode. <br> TRUE for AUTO mode <br> FALSE for MANUAL mode |  |
| XO | REAL | Adjustment value. Must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. <br> In MANUAL mode, Xout = X0 |
| Kp | TIME | Proportionality constant. Must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| TiME | Integral time constant. <br> Possible values range from Oms to 23hr59m59s999ms. |  |
| Td | Derivative time constant. <br> Possible values range from Oms to 23hr59m599s999ms. |  |
| Ts | Minimum sample period. <br> If Ts is less than the application scan time, uses the application <br> scan time as the effective sample interval. <br> The maximum effective sample period is Ts plus the <br> application scan time. <br> Possible values range from Oms to 23hr59m59sg999ms. |  |
| Xmax | BOOL | REAL |
| REAL | REAL | Maximum value of Xout. Must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| XECstop | Minimum value of Xout. Must be a REAL value. Infinity or Not a <br> Number (NaN) values are not supported by this input. |  |
| Increment stop. |  |  |
| Decrement stop. |  |  |
| PID_II function block output. <br> action required on the system to keep the regulation. Xout <br> recalculates each time a sample is taken. |  |  |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| LIMIThi | BOOL | TRUE if Xout is clamped at its maximum value (Xmax). |
| LIMITIo | BOOL | TRUE if Xout is clamped at its minimum value (Xmin). |

Tip: Loss of precision and underflow may occur.

When Xmax < Xmin, the function assumes that the values are reversed and swaps the values. The Reversed error counter increments each time the function has to swap the parameters.

When overflow occurs during execution, the stored integral and error terms set to zero.

In AUTO mode:

- Xout sets to zero
- LIMIThi sets to FALSE
- LIMITlo sets to FALSE

In MANUAL mode:

- Xout sets to XO clamped to the range Xmin to Xmax
- LIMIThi sets to XO > Xmax
- LIMITlo sets to XO < Xmin


## AUTO Mode Operation

When a new sample is made and Auto = TRUE, calculates the output Xout:

```
Xout = Kp * ( E[t] + I[t]/Ti + D[t] * Td)
```

where

```
E[t] = SP - PV (error)
I[t] = I[t-1] + ( E[t] * T ) (integral term)
D[t] = ( E[t] - E[t-1] ) / T (derivative term)
```

$\mathrm{E}[\mathrm{t}-1]$ is the stored error value, that is the value during the last sample.
$\mathrm{I}[\mathrm{t}-1]$ is the stored integral term, that is the value during the last sample.
T is the elapsed time in milliseconds since the last sample.
On initialization, the stored integral term sets to zero, and the action is calculated using proportional action only:

Xout $=K p$ * $[t]$
When $\mathrm{Ti}=0$, the stored integral term sets to zero and Xout is calculated using proportional and derivative naction only:

Xout $=K p^{*}(E[t]+D[t] * T d)$
Once Xout is calculated, the increment stop (INCstop), decrement stop (DECstop) and the clamp (Xmax and Xmin) are applied. When INCstop = TRUE and Xout has increased since the last sample or if DECstop = TRUE and Xout has decreased since the last sample, Xout sets to its previous value. When

Xout > Xmax or Xout < Xmin, Xout is clamped to the range Xmin to Xmax, LIMIThi sets to Xout > Xmax, and LIMITlo sets to Xout < Xmin.

When INCstop, DECstop, or the clamp are active, the previous integral term is retained, that is the stored integral term is not set to the current integral term. On initialisation, INCstop, and DECstop have no effect.

## MANUAL Mode Operation

If a new sample is made and Auto $=$ FALSE, the output Xout immediately follows the adjustment value (XO) clamped to the range Xmin to Xmax. LIMIThi sets to XO > Xmax and LIMITlo sets to XO < Xmin. The stored error term sets to (SP - PV) and the stored integral term sets to zero. The increment and decrement stop (INCstop and DECstop) have no effect.

## Example

(*FBD example with PID_II function block*)

(*ST example*)

```
PID_II(Process1, 25.5, Mode, Adjust, Proportion,
IntēgralT, DerivativeT, T#20ms, 101.2, 91.8, Increment,
Decrement);
Result:= PID_II.Xout;
High := PID_II.LIMIThi;
Low := PID_II.LIMITlo;
```


## See also

Error Statistics (ERRSTAT) on page 829

Selects between two REAL analog values.


Parameter values for the R_SEL function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| SEL | BOOL | Select signal. <br> When TRUE, OUT=PV2 <br> When FALSE, OUT=PV1 |
| PV1 | REAL | Process variable having a REAL value. |
| PV2 | REAL | Process variable having a REAL value. |
| OUT | REAL | REAL value PV1 or PV2. This output is a finite value when the selected <br> process variable has a finite value. |

## Function Block Diagram Example



## Structured Text Example

```
blink_1(Run, Cycle1);
Result := R_SEL(blink_1.Q, Process1, Process2;)
```


## See also

REG2FL on page 845
FL2REGH on page 831
FL2REGL on page 831
Detects when the absolute rate of change per second of the input process variable, over the specified time interval, exceeds the maximum rate of change.


Parameter values for the RATE function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Input process variable. |
| Ts | TIME | Specified sample time interval. |
| R | REAL | Maximum rate of change. <br> This value must be a positive REAL value. |
| H | REAL | Absolute rate of change per second (hysteresis). This value must be a <br> positive REAL value less than R. |
| BOOL | FALSE on initialization. <br> TRUE when PV, during the time frame specified in Ts, exceeds R. Returns <br> to FALSE when the rate of change has fallen below R by the hysteresis <br> value (H). |  |
| Tip:When the elapsed period between application scan times is greater than <br> the sample interval, Ts uses the application program scan time. The <br> maximum period between samples is Ts plus the application scan time. |  |  |

When an overflow or underflow condition occurs during execution, Q sets to FALSE and the corresponding ERRSTAT Overflow or Underflow error counter increments.

When R or H is negative, the inputs use the value zero and the ERRSTAT Param error counter increments.

When H is larger than $\mathrm{R}, \mathrm{Q}$ never sets to FALSE after it sets to TRUE.

## Function Block Diagram Example



## Structured Text Example

```
RATE(Process1, T#30ms, MaxRate, Hysteresis);
RateofChange := RATE.Q;
```


## See also

Error Statistics (ERRSTAT) on page 829
Combines the top and bottom 16-bit values of two integers to produce a 32bit floating point number. REG2FL enables sending float numbers over Modbus as a pair of 16 -bit word values. The Modbus protocol does not support floats.


Parameter values for the REG2FL function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| High | DINT | Top 16-bits of the float value |
| Low | DINT | Bottom 16-bits of the float value |
| 0 | REAL | Float value by combining the two 16-bit values. |
| Tip: | REG2FL does not convert integers to floats. Numbers are still in floating <br> point format. To convert the integer value to a float, use the REAL operator. |  |

## Function Block Diagram Example



## Structured Text Example

```
Result := Reg2Fl(RegHigh, RegLow);
```


## See also

REAL on page 760
FL2REGH on page 831

## FL2REGL on page 831

Linearly scales the input value (PV) according to the low input (IPmin), high input (IPmax), low output (OUTmin) and high output (OUTmax) values. The input value is bound to the range set by IPmin and IPmax.

OUT = ( ( (Min(Max(PV, IPmin), IPmax) - IPmin) / (IPmax - IPmin) ) * (OUTmax OUTmin) ) + OUTmin


Parameter values for the SCALE function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Input value |
| IPmax | REAL | High input REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. <br> When PV > IPmax, scales PV using the IPmax value. |
| IPmin | REAL | Low input REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. <br> When PV < IPmin, scales PV using the IPmin value. |
| OUTmax | REAL | High output REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| OUTmin | REAL | Low output REAL value. Infinity or Not a Number (NaN) values are not <br> supported by this input. |
| OUT | REAL | When PV is within the range set by IPmax and IPmin, OUT scales the input <br> value. <br> When PV is < = IPmin, the value set is OUTmin <br> When PV is >= IPmax, the value set is OUTmax |
| LIMIT | BOOL | Indicates PV is out of range. <br> When FALSE, PV is within the range set by IPmin and IPmax. <br> When TRUE, PV is out of range. |

Tip: Loss of precision may occur when using this function.

When IPmax < IPmin, the function block assumes that the values are reversed and swaps them. The ERRSTAT Reversed error counter increments each time the function has to swap the parameters.

When IPmax = IPmin, OUT sets to the value of OUTmin, LIMIT sets to FALSE and the ERRSTAT ZeroDivide error counter increments.

When PV is Not a Number ( NaN ) value, OUT sets to the value of OUTmin, LIMIT sets to FALSE and the ERRSTAT Param error counter increments.

When an overflow or underflow condition occurs during execution of the function block, OUT sets to OUTmax or OUTmin respectively, LIMIT sets to FALSE and the corresponding ERRSTAT Overflow or Underflow error counter increments. If both occur, OUT sets to OUTmin, LIMIT sets to FALSE and both Overflow and Underflow error counters increment.

## Function Block Diagram Example



## Structured Text Example

```
SCALE(Input, MaxValue, MinValue, OutMaxV, OutMinV);
ScaledValue := SCALE.OUT;
Result := SCALE.LIMIT;
```


## See also

Error Statistics (ERRSTAT) on page 829

Tracks the process variable with the output. The absolute rate of change per second of OUT is limited to the maximum RATE.


Parameter values for the SLEW function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Input process variable having a REAL value. |
| RATE | REAL | Absolute rate of change per second. This input must have a positive REAL <br> value. |
| OUT | REAL | On initialization OUT sets to PV. |

When an overflow or underflow condition occurs during the execution of the function block, OUT sets to PV and the corresponding ERRSTAT Overflow or Underflow error counter increments.

When RATE is negative, the ERRSTAT Param error counter increments and RATE uses 0 as a value.

## Function Block Diagram Example



## Structured Text Example

```
OutValue := (Processl, Rate_pv);
```


## See also

Error Statistics (ERRSTAT) on page 829

## Square Root Extraction (SORTX)

Within a specified range, calculates the square root of a REAL value as a percentage:

OUT = sqrt( ((Min(Max(PV, IPmin), IPmax) - IPmin) / (IPmax - IPmin)) ) * 100


Parameter values for the SQRTX function:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Process variable, scaled to a value of $0 . .1$, according to IPmax and IPmin. <br> This value must be a positive REAL value. |


| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| IPmax | REAL | Maximum positive REAL value of the process variable (PV). Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| IPmin | REAL | Minimum positive REAL value of the process variable (PV). Infinity or Not a <br> Number (NaN) values are not supported by this input. |
| OUT | REAL | Square root of the input value calculated as a percentage. When the PV <br> value is greater than IPmax, the function returns a value of 100.0. When <br> the PV value is less than IPmin, the function returns a value of 0.0. |
| Tip: |  |  |

Tip: Loss of precision may occur when using this function.

When IPmax < IPmin, the function assumes that the values are reversed and swaps them. The ERRSTAT Reversed error counter increments each time the function has to swap the parameters.

When IPmax = IPmin, the function returns a value of zero and the ERRSTAT Zero Divide error counter increments.

When PV is Not a Number ( NaN ) value, OUT is set to zero and the ERRSTAT Param error counter increments.

When an overflow or underflow condition occurs during execution of the function, the function returns zero and increments the corresponding ERRSTAT Overflow or Underflow error counter.

## Function Block Diagram Example



## Structured Text Example

```
Result := SCALE(PV, 2.0, 0.2);
```


## See also

Error Statistics (ERRSTAT) on page 829
SQRT on page 804

Calculates the average value of the process variable over the specified time period. The calculation of the time averaged value (PVavg) is time weighted and achieved through a method of sampling.


Parameter values for the T_AVG function block:

| Parameter | Data Type | Description |
| :--- | :--- | :--- |
| PV | REAL | Process variable having a REAL value. Infinity or Not a Number (NaN) <br> values are not supported by this input. |
| Tp | TIME | Specified time period for calculating the average for PV. <br> Possible values range from Oms to 23hr59m59s999ms. |
| Ts | TIME | Minimum time interval between samples of average values. <br> Collects new samples when the elapsed time since the last sample is <br> $>=$ Ts. <br> Possible values range from Oms to 23hr59m59s999ms. |
| EN | BOOL | When TRUE, T_AVG calculates the average value of the process variable. |
| OUT | When EN is TRUE, PV=PVavg <br> On initialization, sets to PV. Recalculates the average value when <br> collecting new samples on subsequent scans. <br> When EN is FALSE, sets to PV. |  |

Tip: Each T_AVG instance can store a maximum of 64 samples. Exceeding the 64 samples may result in too few stored samples to support Tp. OUT sets to the time averaged value of the available samples.

When an overflow or underflow condition occurs during execution, the OUT sets to PV and the corresponding ERRSTAT Overflow or Underflow error counter increments.

## Calculation of the Time Averaged Value (PVavg)

PVavg is calculated using this method:

- $N s$ is the maximum number of stored samples (currently 64 )
- n is the current number of stored samples ( $1<=\mathrm{n}<=\mathrm{Ns}$ )
- PV[1], PV[2], ..., PV[n] are the values of the previous $n$ samples (PV[1] is the least recent sample, $\mathrm{PV}[\mathrm{n}]$ the most recent)
- $\mathrm{T}[1], \mathrm{T}[2], \ldots ., \mathrm{T}[\mathrm{n}]$ are the times of the previous n samples (strictly increasing)
Tip: If $(T[n]-T[n-1])<T p$ then PVavg $=P V$ else if there exists a natural number $k$
Such that:
(T[n]-T[n-k])>=Tp
PVavg is calculated by method A.
If no such $k$ exists, then if $n<N s$, PVavg is calculated by method $B$.
If $n=N s$, PVavg is calculated by method $C$.


## Method A

```
PVavg =
    (
    ( 0.5 * (PV[n] + PV[n-1]) * (T[n] - T[n-1]) ) +
    (0.5 * (PV[n-1] + PV[n-2]) * (T[n-1] - T[n-2]) ) +
    ...... +
    ( 0.5 * (PV[n-k+1] + PV[n-k]) * (T[n-k+1] - T[n-k])
    )
    )
    / (T[n] - T[n-k])
```

Method B
Calculating PVavg using this method assumes that before time T[1], PV[1] was constant longer than Tp.

```
PVavg =
    (
        ( 0.5 * (PV[n] + PV[n-1]) * (T[n] - T[n-1]) ) +
        (0.5 * (PV[n-1] + PV[n-2]) * (T[n-1] + T[n-2]) ) +
        ...... +
        ( 0.5 * (PV[2] + PV[1]) * (T[2] - T[1]) ) +
        ( PV[1] * (Tp - (T[n] - T[1]) ) )
    )
        / (Tp)
```


## Method C

Use Method Cwhen requiring more than Ns samples to support Tp. Method C calculates the time averaged value over all Ns samples. This corresponds to an effective time period shorter than Tp.

```
PVavg =
    (
    (0.5 * (PV[n] + PV[n-1]) * (T[n] - T[n-1]) ) +
    (0.5 * (PV[n-1] + PV[n-2]) * (T[n-1] + T[n-2]) ) +
    ...... +
    ( 0.5 * (PV[2] + PV[1]) * (T[2] - T[1]) )
    )
        / (T[n] - T[1])
```


## Function Block Diagram Example



## Structured Text Example

```
T_AVG(Process1, T#1m, IntervalT, Calculate);
av_value := T_AVG.OUT;
```


## See also

Error Statistics (ERRSTAT) on page 829
AVERAGE on page 807

## Activity Logger

## Log Entries

## To display the Activity Logger

From the Application View, right-click the project, and then click Activity Logger.

The Activity Logger displays major actions and modifications performed in a project. Each entry in the Activity Logger includes the local date and time of the action or modification and the user name. Importing a Trusted ${ }^{\circledR}$ Toolset project displays the existing history of modifications in the Activity Logger, including the original date and time. Imported entries display N/A instead of the user name.

The Activity Logger displays a maximum of 10,000 entries. Exceeding the maximum number of entries deletes the oldest entry.

## See also

Log Entries on page 852
Legacy Log Entries on page 855
Search the Activity Logger on page 857
Copy and Paste Activity Logger Entries on page 858
For Trusted ${ }^{\circledR}$ projects, the Activity Logger lists these types of actions and modifications. All entries include the date, time, and user account that performed the action or modification.

| Log Entry | Description |
| :---: | :---: |
| Create Project "ProjectName" ProjectComment | Create the project or import the legacy Trusted Toolset project: <br> - Includes the comment for the project <br> - Truncates to a maximum length of 256 bytes |
| Rename project "ProjectName" to "NewProjectName" | Save the project "ProjectName" as "NewProjectName". The new project loads in the Application View. |
| Set project password | Set the project password. |
| Change project password | Modify the project password. |
| Remove project password | Remove the project password. |
| Modify project description | Modify the project description. |
| Make application code | Code result when compiling the application code: <br> - No error detected <br> - Error(s) detected |
| APPLICATION VERSION NUMBER = \# | Version number of the application. |
| Verify program 'ProgramName' | Code result when verifying the Program Organization Unit (POU): <br> - No error detected <br> - Error(s) detected |
| Clean application code | Clean the application code. |
| Start application | Successfully start the application. <br> A "Start Application" entry does not appear in the log if the controller initiates the process. For example, the controller starts the application after the download. |
| Stop application | Successfully stop the application. <br> A "Stop Application" entry appears in the log after the user confirms they want to stop running the application, for example, before the download. |
| Application download successful Application download failed | Success or failure downloading the application to the Trusted controller. |
| Application update successful Application update failed | Success or failure updating the application on the Trusted controller. |
| - WARNING: Cannot determine if on-line update is possible <br> - WARNING: On-line update is not possible | On-line update situation: <br> - Unable to determine if an on-line update is possible <br> - On-line update is not possible |
| - Target requires APPLICATION VERSION NUMBER = \# for update | Application version number required to perform an update. |
| - On-line update will use normal method <br> - On-line update will use Intelligent Update method | Method to use for on-line update: <br> - Normal update <br> - Intelligent update |
| Add I/O device "//ODeviceName" | Add the specified I/0 device. |
| Remove I/0 device "//ODeviceName" | Remove the specified I/0 device. |
| Create conversion table(s) | Create one or more conversion tables. |
| Delete conversion table(s) | Delete one or more conversion tables. |
| Modify Cycle Timing (ms) device property | Modify the amount of time, in milliseconds, allocated to a cycle. |
| Modify Cycle Timing value to "NewCycleTimingValue" | Successfully modify the cycle timing, in milliseconds, on the Trusted controller. |


| Log Entry | Description |
| :--- | :--- |
| Modify Memory for Retain device property | Modify the size of the memory space reserved for storing the values <br> of retained variables. |
| Modify Nb stored errors device property | Modify the number of entries storing detected errors. |
| Modify Generate debug information device <br> property | Enable or disable the generation of debug and monitoring <br> information. |
| Modify Enable ISA68M device property | Enable or disable the ISA68M target. |
| Modify Enable Simulation device property | Enable or disable the application simulation. |
| Rename "DeviceName" to "NewDeviceName" | Rename the device. |
| Create "POUILLanguage" "POUType" <br> "PouName" in "DeviceName" | Create a program, function, or function block in the device. Details <br> the programming language and name for the POU. |
| Delete "POUName" from "DeviceName" | Delete the POU from the device. |
| Delete "POUName" and "Sub-programs" | Delete the POU and its sub-programs from the device. |
| from "DeviceName" | Successfully lock the variable. |
| Create "POULanguage" "POUType" <br> "PouName" in "POUParentName" | Create a child function for the parent program or function. Details <br> the programming language and name for the Pou. |
| Delete "POUName" from "POUParentName" | Delete the child function from the parent program or function. |
| Delete "POUName" and "Sub-programs" | Delete the child function from the parent program or function. |
| from "POUParentName" |  |


| Log Entry | Description |
| :---: | :---: |
| Unlock variable "VariableName" | Successfully unlock the variable. |
| Create global defined word(s) | Create one or more global defined words. |
| Delete global defined word(s) | Delete one or more global defined words. |
| Modify global defined word(s) | Modify one or more global defined words. |
| Create local defined word(s) in "POUName" | Create one or more local defined words for the POU. |
| Delete local defined word(s) from "POUName" | Delete one or more local defined words for the POU. |
| Modify local defined word(s) in "POUName" | Modify one or more local defined words for the POU. |
| Validator \# Exit without viewing all dependency information <br> Validator \# Report printed to file <br> Validator \# Report sent to printer <br> Validator \# File scan complete <br> Validator \# File scan failed or aborted <br> Validator \# scanning TIC code: <br> <ProjectPath> <br> Validator \# scanning file: <ProjectPath> | Validate various actions performed with these validators: Cross Reference Checker (\#1) or TIC Dependency Checker (\#2). |
| Create ActivityLog_YYYYMMDDHHMMSS.xml backup file | Create a backup xml file for the existing Activity Logger entries and a new ActivityLog.xml file. The backup file name includes the year (YYYY), month (MM), day (DD), hour (HH), minute (MM), and second (SS) of file creation. A backup file is created when the ActivityLog.xml file is corrupted. <br> The default backup file is located in \%USERPROFILE \% MMy DocumentsISIS Workstation <version>\Projects\ProjectName\ProjectName\Backup. |
| Activity Logger version is "NewVersionValue" | Report the Activity Logger version when the version changes. |

## See also

Activity Logger on page 852
Legacy Log Entries on page 855
Copy and Paste Activity Logger Entries on page 858
Search the Activity Logger on page 857
When importing a Trusted ${ }^{\circledR}$ Toolset project, the Activity Logger lists these types of actions and modifications. All entries include the date, time, and user account that performed the action or modification.

| Log Entry | Description |
| :--- | :--- |
| BACKUP | Generate a project archive (*.pia). |
| Make application code | Code result when compiling the application code: <br> $\bullet$ No error detected <br> $\bullet$ Error(s) detected |
| APPLICATION VERSION NUMBER = \# | Version number of the application <br> Verify program 'ProgN' <br> Code result when verifying the Program Organization Unit <br> (POU): <br> $\bullet$ No error detected <br> $\bullet ~ E r r o r(s) ~ d e t e c t e d ~$ |


| Log Entry | Description |
| :---: | :---: |
| Download application | Download the application to the Trusted controller. |
| WARNING: Cannot determine if on-line update is possible | Unable to determine if an on-line update is possible. |
| Target requires APPLICATION VERSION NUMBER = \# for update | Application version number required to perform an update. |
| - On-line update will use normal method <br> - On-line update will use Intelligent Update method | Method to use for on-line update: <br> - Normal update <br> - Intelligent update |
| Update application code | Perform an on-line update of the application code running on the Trusted controller. |
| Application run time options | Modify the application run time options. |
| Setup compiler options | Modify the compiler options. |
| IEC1131 TOOLSET - <Project>- Resource definition | Modify the application (resource) definition. |
| IEC1131 TOOLSET - <Project>- Project descriptor | Modify the project description. |
| IEC1131 TOOLSET - <Project>-1/O connection | Save an I/O connection. |
| Conversion tables | Save a conversion table. |
| Create project - Name is 'PrjName' | Create a project. |
| Copy project 'PrjName' to 'NewPrj' | Copy a project to the new project name. |
| Rename project 'PrjName' as 'NewName' | Rename the project. |
| IEC1131 TOOLSET- [Project:Spotlight](Project:Spotlight) - SpotLight | Save a Spotlight HMI. |
| IEC1131 TOOLSET - [Project:SpyList](Project:SpyList) - List of variables | Save the Spy List. |
| Create program 'ProgN' | Create a POU. |
| Copy program 'ProgN' to 'ProgNew' | Copy the POU to the new POU name. |
| Move or rename program 'ProgN' (as 'NewName') | Rename or modify the POU. |
| Delete program 'ProgN' | Delete the POU. |
| ProgN: imported from project <Project> | Import the specified POU from another project. |
| IEC1131 TOOLSET - [Project:Program](Project:Program)- Diary | Edit the Diary file for the specified POU. |
| FBD/LD Program 'ProgN' | Save the specified FBD or LD POU. |
| SFC Program 'ProgN' | Save the specified SFC POU. |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - ST <br> Program | Save the specified ST POU. |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - Quick LD <br> Program | Save the specified quick LD POU. |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - IL Program | Save the specified IL POU. IL POUs are not supported in AADvance ${ }^{\oplus}$-Trusted ${ }^{\oplus}$ SIS Workstation software. |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - Flow Chart | Save the specified FC POU. FC POUs are not supported in AADvance-Trusted SIS Workstation software. |
| ```IEC1131 TO0LSET - <Project:Program> - Local messages IEC1131 T00LSET - <Project:Program> - Local timers IEC1131 TOOLSET - <Project:Program> - Local integers/reals IEC1131 TOOLSET - <Project:Program> - Local booleans IEC1131 TOOLSET - <Project> - Global messages IEC1131 TOOLSET - <Project> - Global timers IEC1131 TOOLSET - <Project> - Global integers/reals IEC1131 TOOLSET - <Project> - Global booleans``` | Save variables having the specified local or global data type in the dictionary. |


| Log Entry | Description |
| :--- | :--- |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - Local <br> function blocks <br> IEC1131 TOOLSET - <Project> - Global function <br> blocks | Save local or global function block instances in the <br> dictionary. |
| IEC1131 TOOLSET - [Project:Program](Project:Program) - Local | Save local or global function block instances in the |
| defines |  |
| IEC1131 TOOLSET - <Project> - Global defines |  |
| IEC1131 TOOLSET - <Project> - Common defines |  |
| Validator \# Exit without viewing all dependency | Validate various actions performed with these validators: |
| information | Cross Reference Checker (\#1) or TIC Dependency Checker |
| Validator \# Report printed to file | (\#2). |
| Validator \# Report sent to printer |  |
| Validator \# File scan complete |  |
| Validator \# File scan failed or aborted |  |
| Validator \# scanning TIC code: <project path> |  |
| Validator \# scanning file: <project path> |  |

Tip: This list may not be exhaustive.

## See also

Activity Logger on page 852
Search the Activity Logger on page 857
Copy and Paste Activity Logger Entries on page 858
Cross Reference Checker on page 632
TIC Dependency Checker on page 633

## Search the Activity Logger

Search the Activity Logger to highlight specific information throughout the entries. Searching enables quickly locating information, such as the user name responsible for modifying a program.

## To search the Activity Logger

1. In the Search box in the Activity Logger, enter the text to find. Entries meeting the search criteria appear highlighted.
Tip: Pressing Ctrl+F in the Activity Logger accesses the search box.
2. To select the next or previous highlighted entry, click Next or Previous
Tip: $\quad$ Pressing F selects the next highlighted entry. Pressing Shift +F 3 selects the previous highlighted entry.

## See also

Activity Logger on page 852
Log Entries on page 852

## Copy and Paste Activity Logger Entries

To print the list of entries in the Activity Logger, copy and paste the content into a text editor like Notepad or Microsoft ${ }^{\circledR}$ Word. The Activity Logger deletes the oldest entry when exceeding the maximum of 10,000 entries. Making a copy of the contents ensures a complete log.

## To copy and paste Activity Logger entries

1. Open the Activity Logger.
2. Select the entries:

- All entries -- Press Ctrl+A.
- Selected entries -- Select the first entry and then press Shift+Down Arrow to select consecutive entries, or hold Ctrl and click to select individual entries.

3. Press Ctrl+C.
4. In a text editor, press $\mathbf{C t r l}+\mathbf{V}$ to paste the entries.

## See also

Activity Logger on page 852
Log Entries on page 852
Diary
Each program organization unit (POU) in a Trusted ${ }^{\circledR}$ application includes a Diary file. The Diary text file enables adding notes and comments for each POU in a project. Add or modify information at any time. When copying and pasting a POU, the content of the diary file is not copied.

Using the Document Generator, print the information displayed in the Diary file for selected POUs.

## To add or modify an entry to the Diary file of a POU

1. In the Application View, expand the node for the required POU, and then double-click <POU>-Diary.
2. In the Diary window, enter or modify the required information, and then save the Diary file.

## See also

Programs on page 525
Functions on page 527
Function Blocks on page 529

## Activity Logger on page 852

## License Trusted applications

Use the AADvance ${ }^{\text {® }}$-Trusted ${ }^{\circledR}$ SIS Workstation software to create virtual machines that run on hardware components, called targets.

For Trusted applications, the AADvance-Trusted SIS Workstation software is available for use with the following types of software licenses:

- Demo License: delivered with the product and available for testing the product. This license provides a 60 -day trial of the fully operational version of the AADvance-Trusted SIS Workstation software for Trusted applications.
- Full License: obtained by manually activating an unlicensed version of the product. The Full License is a fully operational version of the product that enables the creation of Trusted projects that contain one device.

The obtained license is specific to a computer. The Full license persists through all uninstall and reinstall operations. To move the license to another computer, contact the support team to remove the license and then reauthorize it on another computer.

## See also

Obtain an authorized Full license on page 859
Remove an authorized license on page 860
After the Demo License expires, obtain a Full license to continue using the AADvance-Trusted SIS Workstation software for Trusted applications.

## To obtain an authorized Full license

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required to manage licenses, rightclick the SIS Workstation Windows desktop shortcut and then select Run as administrator.
2. From the Help menu, select Licensing CAM Trusted.

The licensing information displays along with three User Codes.
3. Send an email that contains the required license type and all three User Codes to the support team:
keymaster@ra.rockwell.com
4. The support team emails you back Registration Keys 1 and 2.
5. From the Help menu, select Licensing CAM Trusted.
6. Enter the registration keys in their respective fields and select Validate.

Tip: The User Codes may have changed since they were last displayed; the Registration Keys returned by the support team becomes invalid. You must request new Registration Keys by sending the displayed User Codes.
The AADvance-Trusted SIS Workstation software is licensed for use with Trusted applications.

## See also

Remove an authorized license on page 860
License Trusted applications on page 859

## Remove an authorized license

To move the license to another computer, contact the support team to remove the license and reauthorize it on another computer.

## Prerequisites

- Gather authorized license information.
- Contact the support team.


## To remove an authorized license

1. To start AADvance-Trusted SIS Workstation software with the Windows administrator privileges required to manage licenses, rightclick the SIS Workstation Windows desktop shortcut and then select Run as administrator.
2. From the Help menu, select Licensing CAM Trusted.

The licensing information displays along with three User Codes.
3. Send an email that contains all three user codes to the support team: keymaster@ra.rockwell.com
4. The support team emails you back Registration Key 1.
5. From the Help menu, select Licensing CAM Trusted.
6. Enter the registration key in Registration Key $\mathbf{1}$ and select Validate.

A confirmation code appears.
7. Send an email that contains the confirmation code to the support team:
keymaster@ra.rockwell.com
The AADvance-Trusted SIS Workstation software is no longer licensed for Trusted applications.

## See also

Obtain an authorized Full license on page 859
When setting up Trusted ${ }^{\circledR}$ control systems, read these manuals on the Rockwell Automation ${ }^{\circledR}$ Literature Library, http://www.rockwellautomation.com/literature/:

| Publication Title | Publication Number |
| :--- | :--- |
| Trusted TMR System Safety Manual | ICSTT-RM459 |
| 8000 Series TMR System: Operator and Maintenance Manual | ICSTT-RM318 |

Product descriptions that define the hardware components used to set up Trusted control systems:

Tip: To view the latest publication for a product, access http://www.rockwellautomation.com/literature/ and search by the product catalog number.

| Publication Title | Catalog Number | Publication Number |
| :---: | :---: | :---: |
| Trusted SC300E Bridge Module Product Description | 8161 | ICSTT-RM403 |
| Trusted CS300 Bridge Module Product Description | 8162 | ICSTT-RM404 |
| Trusted Cabinets Product Description | CF800 | ICSTT-RM238 |
| Trusted TMR Processor Interface Adapter Product Description | T812X | ICSTT-RM239 |
| Trusted Processor Interface Adapter Product Description | T813X | ICSTT-RM240 |
| Trusted Power System Product Description | T823X | ICSTT-RM241 |
| Trusted Power System Product Description | T824X | ICSTT-RM242 |
| Trusted Sequence of Events and Process Historian Package Product Description | T8013 | ICSTT-RM243 |
| Trusted Toolset Application Validator Software Package Product Description | T8015 | ICSTT-RM244 |
| Trusted Peer to Peer Communications Software Package Product Description | T8017 | ICSTT-RM254 |
| Trusted Process Control Algorithm Software Package Product Description | T8019 | ICSTT-RM246 |
| Trusted OPC Server Package Product Description | T8030 | ICSTT-RM247 |
| Trusted Valve Manager Product Description | T8031 | ICSTT-RM248 |
| Trusted Toolset Suite Product Description | T8082 | ICSTT-RM249 |
| Trusted Controller Chassis Product Description | T8100 | ICSTT-RM250 |
| Trusted TMR Processor Product Description | T8110B | ICSTT-RM251 |
| Trusted Demonstration Unit Product Description | T8141 | ICSTT-RM252 |
| Trusted Communications Interface Product Description | T8151B | ICSTT-RM253 |
| Trusted Communication Interface Adapter Product Description | T8153 | ICSTT-RM254 |
| Trusted TMR Interface Product Description | T8160 | ICSTT-RM255 |
| Trusted Gateway Module Product Description | T8170 | ICSTT-RM256 |
| Trusted Gateway Adaptor Product Description | T8173 | ICSTT-RM402 |
| Trusted High Integrity Power Supply System (110-240 Vac and 24 Vdc ) Product Description | T8200 | ICSTT-RM257 |
| Trusted 24 Vdc Fan Assembly Rack Mounted Product Description | T8270 | ICSTT-RM258 |
| Trusted 24 Vdc Fan Tray Roof Mounted Product Description | T8271 | ICSTT-RM259 |
| Trusted Output Power Distribution Unit Product Description | T8290 | ICSTT-RM260 |
| Trusted Power Distribution Unit MCB 24Vdc Product Description | T8292 | ICSTT-RM261 |
| Trusted Power Distribution Unit 15-Way Fused Product Description | T8293 | ICSTT-RM262 |
| Trusted Utilities Distribution Unit Product Description | T8296 | ICSTT-RM263 |
| Trusted Output Power Distribution Unit (Diode Protected) Product Description | T8297 | ICSTT-RM264 |
| Trusted Expander Chassis Product Description | T8300 | ICSTT-RM265 |
| Trusted TMR Expander Processor Product Description | T8310 | ICSTT-RM266 |
| Trusted TMR Expander Interface Product Description | T8311 | ICSTT-RM267 |
| Trusted Expander Interface Adaptor Unit Product Description | T8312 | ICSTT-RM268 |
| Trusted Fiber TX/RX Unit Product Description | T8314 | ICSTT-RM269 |
| Trusted Dual 24 Vdc Digital Input Module - 60 Channel Product Description | T8402 | ICSTT-RM270 |
| Trusted 24 Vdc Digital Input Module - 40 Channel Product Description | T8403 | ICSTT-RM271 |
| Trusted TMR 35 to 120 Vdc Digital Input Module - 40 Channel Product Description | T8423 | ICSTT-RM272 |
| Trusted TMR 120 Vac Digital Input Module - 40 Channel Product Description | T8424 | ICSTT-RM450 |
| Trusted TMR 24 Vdc Analogue Input Module - 40 Channel Product Description | T8431 | ICSTT-RM273 |
| Trusted Dual 24 Vdc Analogue Input Module - 60 Channel Product Description | T8432 | ICSTT-RM274 |
| Trusted TMR Isolated 4-20 mA Analogue Input Module - 20 Channel Product Description | T8433 | ICSTT-RM275 |
| Trusted TMR Speed Monitor Product Description | T8442 | ICSTT-RM276 |
| Trusted TMR Pulse Generator and Monitoring Module Product Description | T8444 | ICSTT-RM277 |
| Trusted TMR 24 Vdc Zone Interface Module - 40 Channel Product Description | T8448 | ICSTT-RM278 |
| Trusted TMR 24 Vdc Valve Monitor Module - 40 Channel Product Description | T8449 | ICSTT-RM279 |

Chapter 3 Working with Trusted Applications

| Publication Title | Catalog Number | Publication Number |
| :---: | :---: | :---: |
| Trusted TMR 24 Vdc Digital Output Module - 40 Channel Product Description | T8451 | ICSTT-RM280 |
| Trusted TMR 24 Vdc/48 Vdc Output Module - 40 Channel Product Description | T8461 | ICSTT-RM451 |
| Trusted TMR 120 Vdc Digital Output Module - 32 Channel Product Description | T8471 | ICSTT-RM281 |
| Trusted TMR 120 Vac Digital Output Module - 16 Channel Product Description | T8472 | ICSTT-RM283 |
| Trusted TMR Analogue Output Module - 40 Channel Product Description | T8480 | ICSTT-RM284 |
| Trusted 40 Channel 24 Vdc Digital Input FTA Product Description | T8800 | ICSTT-RM285 |
| Trusted 40 Channel 24 Vdc Digital Input FTA Non-Incendive Product Description | T8801 | ICSTT-RM286 |
| Trusted 60 Channel 24 Vdc Digital Input Product Description | T8802 | ICSTT-RM287 |
| Trusted 60 Channel 24 Vdc Digital Input FTA Non-Incendive Product Description | T8805 | ICSTT-RM288 |
| Trusted 40 Channel 120 Vdc Digital Input FTA Product Description | T8821 | ICSTT-RM289 |
| Trusted 40 Channel 120 Vac Digital Input FTA Product Description | T8824 | ICSTT-RM290 |
| Trusted 40 Channel Analogue Input FTA Product Description | T8830 | ICSTT-RM291 |
| Trusted 40 Channel Analogue FTA Non-Incendive Product Description | T8831 | ICSTT-RM292 |
| Trusted 60 Channel Analogue Input FTA Product Description | T8832 | ICSTT-RM293 |
| Trusted 20 Channel Isolated AI/DI FTA Loop Powered Product Description | T8833 | ICSTT-RM294 |
| Trusted 20 Channel Isolated AI/DI FTA External Powered Product Description | T8834 | ICSTT-RM295 |
| Trusted 60 Channel Analogue Input FTA Non Incendive Product Description | T8835 | ICSTT-RM305 |
| Trusted 8 Channel Temperature FTA Product Description | T8840 | ICSTT-RM296 |
| Trusted 8 Channel RTD FTA Product Description | T8841 | ICSTT-RM297 |
| Trusted Versatile FTA Product Description | T8842 | ICSTT-RM298 |
| Trusted 40 Channel Digital Input Versatile FTA Product Description | T8843 | ICSTT-RM299 |
| Trusted 40 Channel Analogue Input Versatile FTA Product Description | T8844 | ICSTT-RM300 |
| Trusted 40 Channel Analogue or Digital Output Versatile FTA Product Description | T8845 | ICSTT-RM301 |
| Trusted Speed Monitor Input FTA (SIFTA) Product Description | T8846 | ICSTT-RM302 |
| Trusted 40 Channel Fire Input Versatile FTA Product Description | T8847 | ICSTT-RM303 |
| Trusted 40 Channel Analogue or Digital Output FTA Product Description | T8850 | ICSTT-RM452 |
| Trusted 32 Channel 120 Vdc Digital Output FTA Product Description | T8870 | ICSTT-RM306 |
| Trusted 16 Channel 120 Vac Digital Output FTA Product Description | T8871 | ICSTT-RM307 |
| Trusted 16 Channel 120 Vac Isolated Digital Output FTA Product Description | T8873 | ICSTT-RM308 |
| Trusted Speed Monitor Output FTA (SOFTA) Product Description | T8891 | ICSTT-RM309 |
| Trusted TCOOO Power Cables Product Description | TCOOO | ICSTT-RM310 |
| Trusted I/O Companion Slot Cables Product Description | TC200 | ICSTT-RM311 |
| Trusted Communications Cables Product Description | TC300 | ICSTT-RM312 |
| Trusted TMR I/O SmartSlot Slot Cables Product Description | TC500 | ICSTT-RM313 |
| Trusted Dual I/O SmartSlot Cables Product Description | TC600 | ICSTT-RM314 |
| Trusted Dual I/O Companion Slot Cables Product Description | TC700 | ICSTT-RM315 |
| Trusted Special I/O Cables Product Description | TC800 | ICSTT-RM316 |

## See also

## Working with Trusted Applications on page 515

## Legal Notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the Legal Notices page of the Rockwell Automation website.

## Software and Cloud Services Agreement

Review and accept the Rockwell Automation Software and Cloud Services Agreement here.

## Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

You can view a full list of all open source software used in this product and their corresponding licenses by opening the oss_license.txt file located your product's OPENSOURCE folder on your hard drive. This file is divided into these sections:

- Components

Includes the name of the open source component, its version number, and the type of license.

- Copyright Text

Includes the name of the open source component, its version number, and the copyright declaration.

- Licenses

Includes the name of the license, the list of open source components citing the license, and the terms of the license.

The default location of this file is:
C:\Program Files (x86)\Rockwell Automation\SIS Workstation <version number>\Help\SISW\OPENSOURCE\oss_licenses.txt
You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s). Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the Contact form on the Rockwell Automation website: http://www.rockwellautomation.com/global/aboutus/contact/contact.page. Please include "Open Source" as part of the request text.

## Index

## A

AADvance $112,113,117,123,128,130,131$, 132, 133, 135, 136, 149, 151, 429, 503
application $138,139,140,144,145,146$, 147, 148
AADvance Discover 321, 323, 325
AADvance-Trusted SIS Workstation
software utilities 24, 27, 29, 38, 39,
40, 41, 42, 47, 49, 51, 55, 57, 59, 108
access control 131, 132
project 131, 577
target 132
action blocks 421, 422, 423
Activity Logger 916, 919, 921
add 569, 571, 574
analog input
I/O modules 292
peer-to-peer 245
analog output
I/0 modules 311
peer-to-peer 247
application
run 138, 603
simulate 149,608
Application View 27, 117, 123, 126, 128, 429, 503

B
binding 220, 222, 226
Block Library 38, 82
Block Selector 47, 48

## C

CIP
consume link 232
consumer variables 233
produce link 230
producer variables 231
close 130
code 135, 139, 604
download 139,604
generate $135,136,137,138,600,601$, 602
coils 341, 342, 343, 364, 365, 366, 367
Communication View 192, 193, 207, 210,
211, 216, 217, 228, 236, 249
contacts $344,345,346,347,368$
controllers 117, 139, 144, 146, 148, 188,
604, 605, 607, 608
clear 145
configure 254, 281
connect 146,605
disconnect 148,608
SNTP 211, 212, 214, 216
start 144, 607
status 188
stop 144, 607
conventions 155, 612
create 44, 67, 112, 222, 230, 232, 557, 663,
667, 671, 698
Cross Reference Browser 83, 184, 185,
186, 677, 679, 681
cycle timing $147,149,431,509$

## D

data types 437, 447, 448, 799, 800, 801, 803
IEC 61131-3 types $438,440,441,442$, $444,445,446,799,803$
defined words $85,436,663,797$
Description window 108
development environment, options 75,
80, 82, 83, 84, 89, 98, 107
development environment, working 61,
62, 63, 65, 66, 67, 70, 73
diagnostic information 606
Dictionary 175, 176, 177, 179, 181, 663, 666
digital input
I/O modules 288
peer-to-peer 243
digital output
I/O modules 303
peer-to-peer 246
Documentation Generator 84
download 139, 604, 661
code 139,604

## E

edit 45, 664, 699, 778, 779
environment 76,77,78,79
startup 79
Equipment View 253, 254, 256, 281, 283
Ethernet
communications 207, 209
forwarding 325
ports 192,660
export 70,71
function 571
function block 573
program 569
settings 70,71
variables data 133, 577

## F

FBD 327, 328, 330, 331, 332, 334, 349, 700, 702, 704, 719
debug 702
elements $334,335,336,337,338,339$, 340, 344, 348, 358, 703, 704, $705,708,709,714,718,729,740$, 741
file system paths 159, 615
Find and Replace 55, 56, 76
Quick Find 55
Quick Replace 56
function blocks 483,503,504,505,506,
$507,508,509,510,512,513,514,515$,
$516,518,519,573,574,868,869,870$,
871, 872, 874, 875, 876, 877, 878, 879,
880, 881, 882, 883, 884, 885
Series 9000_L 520,521,522,523
functions $123,476,477,478,479,480$,
481, 482, 483, 484, 485, 486, 487, 488,
489, 490, 491, 492, 493, 494, 495, 496,
497, 498, 499, 500, 501, 502, 503, 571,
825, 836, 837, 838, 839, 840, 841, 842,
843, 845, 846, 847, 848, 849, 850, 851,
852, 853, 854, 855, 856,857, 858, 859,
$860,861,862,863,864,865,866,867$

## G

global library functions and function
blocks 887, 888, 890, 892, 893, 894,
895, 899, 900, 901, 902, 903, 906, 907,
908, 909, 910, 911, 912
grid options 84, 85, 86, 87, 88
Arrays View 84
Defined Words View 85
Dictionary View 86
Parameters Grid 86
Structures View 87
Variable Groups View 88
Variable Selector Settings 88

## I/O modules

analog input modules 292, 293, 294, 295, 297, 298, 301, 302
analog output modules 297, 298,311, $312,313,314,315,319$
digital input modules 288, 289, 290, 291
digital output modules $303,304,305$,
306, 308, 309, 310
$1 / 0$ wiring 691, 700
import 70,71,72,77, 113, 133, 559
settings $71,72,77$
variables data 133, 577
ISaView 98, 99
animation settings 98
define animation effects for objects
783, 784, 786, 787, 788, 789
edit objects 778,779,780,781,782
edition settings 99
insert objects 768, 769, 770, 771,772, 775, 776, 777
object settings 100, 101, 102, 103, 104, 105, 106
preview screens 789, 790, 791
screens 789
L
Language Editor 29, 30
LD 376, 426, 721, 722, 723, 725, 726, 745
elements 332, 357, 361, 368, 372, 373, 374, 714, 723, 725, 726, 729, 740, 741
coils 712, 713, 731, 732, 733
contacts 716, 717
libraries 128
publish 171
Trusted 575
license 553, 922

## M

Modbus 193, 195, 196, 197, 200, 203, 204
modify 147
0
offline 107, 171
online
grid 108
updates 616,617,619,620,622,624
open 130, 576
operators 449, 451, 452, 453, 454, 455,
456, 457, 458, 459, 460, 461, 462, 463,
464, 465, 466, 467, 468, 469, 470, 471,
472, 473, 474, 804, 806, 807, 808, 809,
810, 811, 812, 813, 814, 815, 816, 817, 818,
819, 820, 821, 824
options 75, 79, 80, 84, 89, 98
IEC Language $90,93,94,96$
ISaVIEW 98,99

## P

peer-to-peer 236, 237, 238, 239, 240,
242, 243, 245, 246, 247, 248
programs 429, 569, 792
project $112,113,130,131,136,138,557,559$,
561, 576

| 0 | U |
| :---: | :---: |
| Quality of Service 249, 250 | update 140, 616, 617, 620 upload 661 |
| R |  |
|  | V |
| report 624 |  |
| reserved keywords 432,795 | validators 684, 685, 687, 688, 689 |
| reset 72 | variable dependencies 186 |
|  | Variable Selector 44, 45, 46 |
| S | variables 82, 126,574 |
| serial | CIP 207, 209 |
| serial | control processor variables 266, 267, |
| ports 210,659 | 268,269 |
| tunneling 216 | directly represented 434,797 |
| settings 70, 71, 72, 75, 76, 82, 83, 84, 89, | force the value 151,609 |
| 98, 99, 100, 101, 102, 103, 105, 106, 107, | map 207, 209, 231, 233 |
| 132, 659 | monitor the value 152,611 |
| SFC 401, 403, 404, 405, 406, 419, 424, | RTC processor variables 269, 270, 271, |
| 426, 427 | 272, 273, 274, 275, 276, 277, 278, |
| elements 411, 413, 414, 418 | 279, 280, 281 |
| sequence controls 415,416,417 | status processor variables 258,259, |
| shortcut 73, 78, 349, 377, 399, 427, 700, | 260, 261, 262, 263, 264, 265, 266 |
| 719, 745 | verify 137,326 |
| simulate 149, 150 | version control |
| specify 79, 80, 82, 83, 84, 207 | check in 162, 164 |
| spy list 57 | check out 163, 164 |
| settings 107, 108 | elements 162, 163, 164, 165, 172, 173 |
| ST 379, 381, 382, 383, 385, 392, 399, 425, | repository 161, 166, 168, 169, 170, 171 |
| 747, 752, 759, 763 | version 165, 166 |
| elements and statements 385, 386, 387, 388, 390, 391, 392, 753, 755 | W |
| extensions 393,395, 396,397,760 |  |
| start | windows 63, 64, 79 |
| controllers 144,607 |  |
| stop |  |
| controllers 144,607 |  |
| System Configuration Tool 626, 648, 654, 659, 661 |  |
| communications port 659,660 |  |
| SYSTEM.INI |  |
| configure 649,650,651,653 |  |
| example 654 |  |
| generate 654 |  |
| templates 628,631,635,637,641,642, |  |
| 643, 644, 645, 646, 647 |  |
| LED 631,633,635 system 638,639,640 |  |
| system type 282, 283 |  |
| T |  |
| TIC 685, 687, 688 |  |
| Toolbox 41 |  |
| Trusted 83, 557, 559, 561, 565, 575, 576, 601,602 |  |

## Rockwell Automation support

Use these resources to access support information.

| Technical Support Center | Find help with how-to videos, FAOs, chat, user forums, and product notification <br> updates. | rok.auto/support |
| :--- | :--- | :--- |
| Knowledgebase | Access Knowledgebase articles. | rok.auto/knowledgebase |
| Local Technical Support Phone Numbers | Locate the telephone number for your country. | rok.auto/phonesupport |
| Literature Library | Find installation instructions, manuals, brochures, and technical data publications. | rok.auto/literature |
| Product Compatibility and Download Center <br> (PCDC) | Get help determining how products interact, check features and capabilities, and <br> find associated firmware. | rok.auto/pcdc |

## Documentation feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

## Waste Electrical and Electronic Equipment (WEEE)

At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

[^15]rockwellautomation.com


[^0]:    Tip: $\quad$ When changing the values in Tools > Options > IEC Languages > Ladder Diagram (LD), close and then reopen the software before using Reset All Visual Settings or Reset Container Visual Settings to apply the new values to the LD Container properties.

[^1]:    Comment Style

[^2]:    Output Variable (Write) Behavior

[^3]:    Tip: $\quad$ To retain customized display settings, save the Dictionary instance before closing.

[^4]:    Tip: Default values for the error behavior and update behavior are unsupported by variable bindings using complex data types such as structures and arrays; use the last value.

[^5]:    Tip: When configuring a CIP consume link, define the producer name and the
    CIP path to the producer. If the IP address of the producer is modified, update the path to the producer.

[^6]:    Tip: The required system type must be set before adding I/O modules to a controller.

[^7]:    When using HART Pass-Through in a safety system, follow these precautionary guidelines:

    - Ensure HART Pass-Through is only enabled under control of the application.
    - Ensure HART Pass-Through is enabled only when necessary.
    - Configure the application to start an alarm if HART Pass-Through is enabled on any safety-critical channel of any module.

[^8]:    Tip: Before monitoring, remove the strings defined for the Message True and Message False variable properties for contacts and coils.

[^9]:    Tip: When you force variable values, the computer running AADvance-Trusted SIS Workstation software or AADvance Standalone OPC server sends an IXL write message to the controller. If the IXL write message is processed right after Execute target independent code (TIC) in the application execution loop (step 4), the forced value is used for the producer binding, wired output, or retained variable.
    If the IXL write message is processed right after Save retain variables in the current application execution loop (step 8), the step Execute target independent code (TIC) in the next application execution loop may overwrite the forced value with a different value for the producer binding, wired output, or retained variable.
    To help achieve deterministic behavior, avoid forcing an:

    - internal variable that is bound as a producer or retained (internal variables cannot be locked)
    - unlocked output variable

[^10]:    Tip: For information on the Speed Monitor template, refer to the Trusted ${ }^{\circledR}$ T8442 TMR
    Speed Monitor product description.

[^11]:    Tip: De-energized Short Circuit templates will not work on older 5V HIU hardware builds. An attempt to swap from a 3.3 V module with this template, to a 5 V module, will fail.

[^12]:    Tip: For Companion Slot modules, enter a module in the primary slot. Enable simulation and enter the partner (secondary) chassis and slot location. Do NOT enter a module in the secondary slot.

[^13]:    Tip: Update each instance of a function or function block after adding, removing, renaming, changing data type, or changing direction of parameters in all POUs using the instance. Identify these POUs with the Cross Reference Browser, then open each POU and reselect the modified instance using the Block Selector.

[^14]:    Important: Avoid using this function. This function is intended to be obsolete.

[^15]:    Allen-Bradley, expanding human possibility, Logix, Rockwell Automation, and Rockwell Software are trademarks of Rockwell Automation, Inc.
    EtherNet/IP is a trademark of ODVA, Inc.
    Trademarks not belonging to Rockwell Automation are property of their respective companies.
    Rockwell Otomayson Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752, İçerenkÖy, İstanbul, Tel: +90 (216) 5698400 EEE YÖnetmeliğine Uygundur

    > Connect with us. f

