rexroth
A Bosch Company

IndraWorks 14VRS

WinStudio 7.4

Reference Book
R911341585

Edition 08



Title IndraWorks
14VRS
WinStudio 7.4

Type of Documentation Reference Book

Document Typecode DOK-IWORKS-WINSTUD*V14-AP08-EN-P

Internal File Reference RS-94c518d2f913c9b90a347ea50068d039-7-en-US-3

Change Record Edition 08, 2019-04
Refer to tab. 1-1 "Change Record" on page 1

Copyright © Bosch Rexroth AG 2019
All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.

Liability The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.

Editorial Department Engineering Software and Control Firmware - Visualization and Diagnostics (ThoBe), Editorial Department (MiNi/PiaSt)

Table of Contents

Page

1 About this documentation ... 1
1.1 Validity of this documentation ... 1
1.2 Documentation structure ... 1
1.3 Using safety instructions ... 3
1.3.1 Structure of the safety instructions ... 3
1.3.2 Explaining signal words and safety alert symbol ... 3
1.3.3 Symbols used ... 5
1.3.4 Explaining the signal alert symbol on the device ... 5
1.4 Conventions this documentation ... 5
1.5 Required and supplementing documentation ... 6

2 About this application ... 7
2.1 Product Overview ... 7
2.2 Product features ... 8
2.3 How the software works ... 9
2.3.1 Internal structure and data flow ... 9
2.3.2 Executing/switching modules ... 13
2.3.3 Executing/switching the background task ... 17

3 Installation ... 21
3.1 Comparing the Rexroth WinStudio software components ... 21
3.2 Installing the complete Rexroth WinStudio software ... 22
3.3 Installing EmbeddedView or CEView on a target device ... 27

4 Licensing ... 33
4.1 License Protection Types ... 33
4.2 License settings ... 33
4.3 License types ... 34
4.4 Product versions ... 36
4.5 WinStudio License Activation ... 37
4.6 Invalid Licenses ... 38

5 Development environment ... 39
5.1 Development Environment, General Information ... 39
5.2 WinStudio toolbars ... 39
5.2.1 WinStudio Toolbars, Customizing ... 39
5.2.2 WinStudio Execution Functions Toolbar ... 40
5.2.3 WinStudio Alignment Functions Toolbar ... 41
5.2.4 WinStudio Tag Properties Toolbar ... 42
5.2.5 WinStudio Object Properties Toolbar ... 42
5.2.6 WinStudio table functions toolbar ... 43
5.2.7 WinStudio Debug Functions Toolbar ... 44
5.2.8 WinStudio Dynamic Functions Toolbar ... 44

Page

5.3 Project Explorer ... 45
5.3.1 Project Explorer, General Information ... 45
5.3.2 WinStudio folder ... 46
5.3.3 Variables folder ... 46
5.3.4 Tasks folder ... 47
5.3.5 Communication folder ... 49
5.4 Configuring and testing an application ... 50
5.4.1 Verifying an application ... 50
5.4.2 Starting the application ... 51
5.4.3 Stopping the application ... 51
5.4.4 Debug application ... 52
5.4.5 WinStudio target platform ... 52
5.4.6 Project settings ... 53
Information Tab ... 53
Options tab ... 54
Options tab, general information ... 54
Saving the alarm history to an external database ... 55
Configuring a default database for all task histories ... 56
Database configuration ... 56
Viewer tab ... 61
Viewer tab, General information ... 61
Keyboard ... 66
Virtual keyboard ... 66
Communication tab ... 67
Preferences tab ... 69
Web tab ... 71
5.5 WinStudio tools ... 76
5.6 Screen/Worksheet Editor ... 77
5.7 Database spy ... 78
5.8 LogWin output window ... 79
5.9 Cross reference list ... 82
5.10 Text search ... 83
5.11 Status Bar ... 84

6 WinStudio projects ... 85
6.1 Creating a new project ... 85
6.1.1 Creating a new project, general information ... 85
6.1.2 Target platforms and product types ... 86
6.1.3 Changing the target system of an existing project ... 88
6.2 E-Mail Configuration ... 88
6.3 FTP Configuration ... 90
6.4 Execution tasks dialog ... 91
6.5 Project execution under Windows services ... 93

Page

7 Working with tags ... 103
7.1 Tags and project database ... 103
7.1.1 Tags and the project database, general information ... 103
7.1.2 Application variables ... 104
Application variables, general information ... 104
Extending the Application Variables Datasheet ... 105
7.1.3 Classes folder ... 106
7.1.4 Distributed variables (Shared database folder) ... 107
7.1.5 Internal Variables (System tags folder) ... 108
7.2 Creating tags ... 109
7.2.1 Tag Name Syntax ... 109
7.2.2 Choosing the tag type ... 109
7.2.3 Choosing the Tag Data Type ... 111
7.2.4 Using Array Tags ... 112
7.2.5 Using classes ... 115
7.2.6 Using Indirect Tags (Pointers) ... 117
7.2.7 Choosing the Tag Scope ... 118
7.3 Creating database tags ... 119
7.3.1 Creating tags on the datasheet ... 119
7.3.2 Creating tags ""on-the-fly"" ... 120
7.3.3 Editing Tags ... 120
7.4 Creating classes ... 121
7.5 Tag properties ... 122
7.5.1 Tag properties ... 122
7.5.2 Referencing a tag property instead of a project tag ... 123
7.5.3 Editing the tag database during runtime ... 124
7.5.4 Properties of Integer and Real tags ... 126
7.5.5 Properties of Boolean tags ... 131
7.5.6 Properties of String tags ... 134
7.5.7 List of tag properties ... 136
7.5.8 Change handling of tag values outside the validity range ... 141
7.6 Using tags in the project ... 142
7.7 Deleting a tag from the project database ... 143
7.8 Sorting or filtering the lines of a worksheet ... 143
7.9 Renaming and removing tags ... 146
7.9.1 Replacing tags globally ... 146
7.9.2 Replacing Tags in a Document or Screen Object ... 146
7.9.3 Removing Unused Tags from the Project Database ... 147
7.9.4 Tags database reset ... 149
7.9.5 Using available tags ... 149
7.9.6 Changing tag properties ... 150
7.10 Importing an external database ... 151
7.10.1 Import Wizard ... 151
7.10.2 Importing from ... 154
Importing from Other Project Databases ... 154
Importing from OPC Server Databases ... 156

Page

Importing from CSV Databases ... 157
Importing from ODBC Databases ... 158
Importing Studio XML screens ... 159
7.11 Tag integration ... 161
7.11.1 Tag integration, general information ... 161
7.11.2 Sharing tags on IndraLogic devices ... 162
7.11.3 Renaming integrated tags in the project ... 166

8 Screens and graphics ... 169
8.1 WinStudio project structure ... 170
8.1.1 WinStudio folder ... 170
WinStudio folder, general information ... 170
Screen attributes dialog ... 170
Background color or picture of a WinStudio screen ... 173
8.1.2 Screen groups folder ... 175
8.1.3 Layout tool ... 177
8.2 Using screen objects and animations ... 178
8.2.1 Editing screen objects ... 178
Editing screen objects, general information ... 178
Editing Object Properties ... 178
Replacing Project Tags in a Document or Screen Object ... 178
Arranging objects ... 180
Locking Objects in the Screen Editor ... 180
Undo ... 180
Grid Settings ... 181
8.2.2 Static objects ... 181
Static objects, general information ... 181
Line Object ... 181
Open polygon object ... 182
Closed polygon object ... 183
Rectangle object ... 184
Rounded Rectangle Object ... 186
Ellipse Object ... 187
Adding graphics to a WinStudio screen ... 188
Changing properties of multiple screen objects (static objects) ... 190
8.2.3 Active objects ... 191
Active objects, general information ... 191
Text object ... 192
Draw and configure a Text box object ... 193
Button object ... 195
Pushbutton object ... 197
Checkbox object ... 201
Radio button object ... 204
Operating Modes for Checkbox and Radio Button Objects ... 207
Combo box object ... 208
List box object ... 212

Page

Smart message object ... 216
Changing the properties of multiple screen objects (active objects) ... 221
8.2.4 WinStudio libraries ... 222
Symbols ... 222
Symbols - General information ... 222
Saving own project symbols ... 223
ActiveX control ... 231
.NET controls ... 237
Adding a link to an external graphic file ... 242
8.2.5 Applying animations to screen objects ... 246
Applying animations to screen objects, general information ... 246
Command animation ... 246
Hyperlink animation ... 249
Bargraph Animation ... 250
Text data link animation ... 251
Color Animation ... 253
Position Animation ... 255
Resize Animation ... 257
Rotation animation ... 259
8.2.6 Using user-defined property values when opening image objects ... 261
8.3 Formatting screen objects ... 265
8.3.1 Formatting screen objects, general information ... 265
8.3.2 Changing the properties of multiple screen objects (formatting) ... 265
8.3.3 Determining the tab order of screen objects ... 267
8.3.4 "Move to foreground" and "Move to background" ... 269
8.3.5 "To the foreground" and "To the background", by one level ... 269
8.3.6 "Group " and "ungroup objects" ... 270
8.3.7 "Align", "Center" and "Distribute" ... 271
8.3.8 "Rotating" and "mirroring" ... 273
"Rotating" ... 273
"Mirroring vertically" ... 273
"Mirroring horizontally" ... 274
8.3.9 Adjusting position ... 274
8.3.10 "Same width" / "same height" for selected objects ... 275
8.3.11 Fill color ... 276
8.3.12 Line color ... 276
8.3.13 Color Interface ... 277
8.3.14 Fonts ... 280
8.4 Data input ... 282
8.5 Multi-touch in your project ... 285
8.5.1 Using multi-touch in your project ... 285
8.5.2 Multi-touch settings for project screens ... 286
Multi-touch settings for project screens, General information ... 286
Configuring multi-touch settings for all project screens ... 286
Configuring the standard multi-touch settings for an individual project screen ... 289
8.5.3 The different multi-touch gestures ... 292

Page

The different multi-touch gestures ... 292
Using multi-touch gestures in project screens ... 292
Using multi-touch gestures in data objects ... 294
Using multi-touch gestures in object animations ... 297
8.5.4 Touch events ... 299
Using touch events ... 299
Adding a touch event to a screen object ... 299
Adding a touch event to a project screen ... 300
8.5.5 On touch ... 301
8.5.6 OnTouchStart ... 304
8.5.7 OnTouchDelta ... 304
8.5.8 OnTouchCompleted ... 306
8.6 Converting screen resolution ... 307
8.7 Importing Studio XML templates ... 310

9 Alarms, events and trends ... 313
9.1 Alarm worksheet ... 313
9.1.1 Alarm worksheet header ... 314
Email settings for alarm worksheet ... 315
Advanced settings for alarm worksheet ... 318
9.1.2 Alarm worksheet body ... 319
9.1.3 Sorting or filtering the lines of a worksheet ... 320
9.1.4 Saving the alarm history to an external database ... 322
9.1.5 Format of the alarm history ... 324
9.2 Events and event logs ... 326
9.2.1 Events and event logs, general information ... 326
9.2.2 Enabling the event logger ... 326
9.2.3 Saving the event log to an external database ... 328
9.2.4 Format of the event log ... 329
9.3 Alarm/event control object ... 331
9.4 Trend worksheet ... 340
9.4.1 Sorting or filtering the lines of a worksheet ... 343
9.4.2 Converting trend history files from binary to text ... 345
9.4.3 Converting trend history files from text to binary ... 346
9.4.4 Batch creation of history files ... 346
9.5 Trend control object ... 349
9.5.1 About the Trend control runtime interface ... 350
9.5.2 Object properties: Trend control dialog ... 353
Points dialog ... 354
Pen style ... 355
Options ... 357
SPC ... 360
Axes ... 360
Toolbar ... 365
Data source ... 366
Legend dialog ... 369

Page

Advanced ... 371
9.5.3 Using a Text File as Data Source ... 374
9.5.4 Using a Database as Data Source ... 377
9.6 Grid object ... 381
9.6.1 Data dialog ... 383
9.6.2 Columns dialog ... 385
9.6.3 Advanced dialog ... 388

10 Background tasks ... 391
10.1 Alarm worksheet, general information ... 391
10.2 Trend worksheet, general information ... 392
10.3 Recipe worksheet ... 395
10.4 Report worksheet ... 396
10.5 ODBC worksheet ... 397
10.6 Math worksheet ... 400
10.7 Script worksheet ... 401
10.7.1 Script worksheet, general information ... 401
10.7.2 Script Worksheet Startup ... 402
10.8 Scheduler Worksheet ... 403
10.9 Database/ERP worksheet ... 404
10.10 Sorting or filtering the lines of a worksheet ... 410

11 Communication with other devices ... 413
11.1 Configuring direct communication with a remote device ... 413
11.1.1 Configuring direct communication with a remote device, general information ... 413
11.1.2 Driver sheets ... 421
11.2 Inserting tags using the tag wizard ... 423
11.3 Adding classes using the class wizard ... 425
11.4 Configuring an OPC client connection to an OPC server (OPC) ... 427
11.5 Configuring an OPC UA client connection to an OPC UA server ... 433
11.5.1 Configuring an OPC UA client worksheet ... 433
11.5.2 Connecting to an OPC UA server using self-signed certificates ... 442
11.5.3 Connecting to an OPC UA server using CA-signed certificates ... 444
11.5.4 List of read/write status codes and messages for OPC UA ... 445
11.5.5 Activating the OPC UA trace log ... 451
11.6 Configuring a TCP/IP Client Connection to Another Project ... 453
11.7 Configuring a DDE Client Connection to a DDE Server ... 455
11.8 Sorting or filtering the lines of a worksheet ... 458

12 Project security ... 461
12.1 Security Modes ... 461
12.2 Security access levels ... 462
12.3 Using the security system configuration wizard ... 463
12.4 Configuring server settings for security modes ... 466
12.4.1 Configuring server settings for security modes, general information ... 466

Page

12.4.2 Extending the LDAP Schema to Allow Saving of Security Rights ... 469
12.5 Creating and configuring groups ... 474
12.6 Creating and configuring users ... 478
12.7 Security system dialog ... 480
12.8 Backing up the security system configuration ... 481
12.9 Logging On/Off ... 482
12.10 Blocking or unblocking users ... 483
12.11 Password protection for screens, symbols and worksheets ... 484

13 Project localization ... 487
13.1 Project localization, general information ... 487
13.2 Adding a target language to the translation table ... 487
13.3 Configuring fonts for a target language ... 490
13.3.1 Configuring fonts for a target language, Basics ... 490
13.3.2 Configuring fonts for a target language, Examples ... 491
13.4 Translating application texts to additional languages ... 492
13.5 Activating a language at runtime ... 493
13.5.1 Activating a Language at Runtime, General Information ... 493
13.5.2 Setting the project language at program start ... 493
13.5.3 Setting the project language at runtime ... 494
13.6 Disabling Translation of Selected Screen Objects ... 495
13.7 Configuring the advanced translation settings ... 495

14 Testing and debugging projects ... 497
14.1 Output (LogWin) ... 497
14.2 Database spy ... 500
14.3 Using the LogWin module ... 501
14.4 Using remote tools ... 502
14.4.1 Remote database spy ... 502
14.4.2 Remote LogWin ... 503

15 Web thin client and secure viewer ... 505
15.1 Introduction into thin clients ... 505
15.2 Creating a simple thin client program ... 507
15.3 Underlying Technology ... 515
15.4 ISSymbol control layer ... 516
15.5 Examples of Client/Server Architecture ... 519
15.6 Configuring the data server ... 523
15.7 Configuring a web server as host for project pages ... 527
15.8 Installing the web tunneling gateway ... 530
15.9 Configuring the thin client ... 532
15.10 Customization of the “Viewer” program icon in the task bar ... 536
15.11 Implementing security for web-based applications ... 537
15.12 Port Usage ... 542
15.13 Example: Configuring a WinStudio web application ... 543

Page

16 Downloading to a remote device ... 545
16.1 Remote management ... 545
16.2 Install or update a softkey license for EmbeddedView or CEView (remote) ... 545
16.3 Enabling security in the remote agent and adding users ... 547
16.4 Executing or stopping your project on the target device ... 549
16.5 Configuring the Remote Agent to automatically run a project ... 551

17 Database interface ... 553
17.1 Database interface, general information ... 553
17.2 SQL relational databases ... 554
17.3 Linking the database through a remote DB provider ... 555
17.4 Studio database gateway ... 556
17.5 Database configuration ... 563
17.6 Configuring a default database for all task histories ... 567
17.7 Database debugging ... 568
17.8 Appendices ... 571
17.8.1 Using ODBC databases ... 571
17.8.2 Using Microsoft SQL server ... 573
17.8.3 Using ORACLE databases ... 574
17.8.4 Using Microsoft Access databases ... 575
17.8.5 Using SQL server CE ... 578
Using SQL server CE, general information ... 578
Installing SQL server on a Windows Embedded device ... 579
17.8.6 Using Sybase ... 580
17.8.7 Using Microsoft Excel ... 580
17.8.8 Using MySQL ... 583

18 Debugging and FAQs ... 585
18.1 General Troubleshooting ... 585
18.2 Frequently asked questions ... 586
18.2.1 Database and security system ... 586
18.2.2 Graphics ... 587
18.2.3 Tasks ... 588
18.2.4 Communication ... 589
18.2.5 General Information ... 590

19 Appendix: WinStudio-internal scripting language ... 595
19.1 Appendix: Internal scripting language, general information ... 595
19.2 List of Available Functions ... 595
19.3 Structure of the Functional Descriptions ... 608
19.4 Log Message Functions ... 609
19.4.1 Log Message Functions – General Information ... 609
19.4.2 Trace ... 610
19.5 Arithmetic Functions ... 610
19.5.1 Arithmetic Functions – General Information ... 610

Page

19.5.2 Arithmetic Operators ... 611
19.5.3 Abs ... 611
19.5.4 Div ... 612
19.5.5 Format ... 612
19.5.6 GetBit ... 616
19.5.7 Mod ... 617
19.5.8 Pow ... 617
19.5.9 ResetBit ... 618
19.5.10 Round ... 619
19.5.11 SetBit ... 619
19.5.12 Sqrt ... 620
19.5.13 Swap16 ... 620
19.5.14 Swap32 ... 621
19.5.15 Trunc ... 621
19.6 Statistical Functions ... 622
19.6.1 Statistical Functions – General Information ... 622
19.6.2 Avg ... 622
19.6.3 Max ... 623
19.6.4 Min ... 624
19.6.5 Rand ... 625
19.7 Logarithmic Functions ... 625
19.7.1 Logarithmic Functions – General Information ... 625
19.7.2 Exp ... 625
19.7.3 Log ... 626
19.7.4 Log10 ... 626
19.8 Logical functions ... 627
19.8.1 Logical functions – General Information ... 627
19.8.2 Logical Operators ... 627
19.8.3 False ... 628
19.8.4 If ... 628
19.8.5 Toggle ... 629
19.8.6 True ... 630
19.9 String Functions ... 630
19.9.1 String Functions – General Information ... 630
19.9.2 Asc2Str ... 631
19.9.3 CharToValue ... 632
19.9.4 CharToValueW ... 633
19.9.5 ClassMembersToStrVector ... 633
19.9.6 NCopy ... 634
19.9.7 Num ... 635
19.9.8 Str ... 636
19.9.9 Str2Asc ... 636
19.9.10 StrCompare ... 636
19.9.11 StrCompareNoCase ... 637
19.9.12 StrFromInt ... 638
19.9.13 StrFromReal ... 638

Page

19.9.14 StrFromTime ... 639
19.9.15 StrGetElement ... 640
19.9.16 StrLeft ... 641
19.9.17 StrLen ... 641
19.9.18 StrLower ... 642
19.9.19 StrRChr ... 642
19.9.20 StrRight ... 642
19.9.21 StrSetElement ... 643
19.9.22 StrStr ... 643
19.9.23 StrStrPos ... 644
19.9.24 StrTrim ... 644
19.9.25 StrTrimAll ... 645
19.9.26 StrUpper ... 646
19.9.27 ValueToChar ... 646
19.9.28 ValueWToChar ... 647
19.10 Date and Time Functions ... 647
19.10.1 Date & Time functions – General Information ... 647
19.10.2 ClockGetDate ... 648
19.10.3 ClockGetDayOfWeek ... 649
19.10.4 ClockGetTime ... 651
19.10.5 DateTime2Clock ... 652
19.10.6 DateTime2UTC ... 653
19.10.7 GetClock ... 655
19.10.8 GetTimeZone ... 655
19.10.9 GetTimeZoneCount ... 657
19.10.10 GetUTC ... 657
19.10.11 Hour2Clock ... 657
19.10.12 SetSystemDate ... 658
19.10.13 SetSystemTime ... 658
19.10.14 SetTimeZone ... 659
19.10.15 UTC2DateTime ... 660
19.11 Trigonometric Functions ... 661
19.11.1 Trigonometric Functions – General Information ... 661
19.11.2 ACos ... 661
19.11.3 ASin ... 662
19.11.4 ATan ... 662
19.11.5 Cos ... 662
19.11.6 Cot ... 663
19.11.7 Pi ... 663
19.11.8 Sin ... 664
19.11.9 Tan ... 664
19.12 Screen Functions ... 665
19.12.1 Screen Functions – General Information ... 665
19.12.2 Close ... 665
19.12.3 Open ... 666
19.12.4 OpenPrevious ... 670

Page

19.12.5 ShowInplaceInput ... 671
19.12.6 ShowMessageBox ... 673
19.13 Security Functions ... 675
19.13.1 Security Functions – General Information ... 675
19.13.2 BlockUser ... 675
19.13.3 CheckESign ... 676
19.13.4 CheckSecurityLevel ... 677
19.13.5 CreateUser ... 678
19.13.6 ExportSecuritySystem ... 680
19.13.7 GetLastESignUser ... 681
19.13.8 GetSecuritySystemStatus ... 681
19.13.9 GetUserFullName ... 683
19.13.10 GetUserNames ... 683
19.13.11 GetUserPwdAging ... 684
19.13.12 GetUserState ... 685
19.13.13 ImportSecuritySystem ... 685
19.13.14 RemoveUser ... 687
19.13.15 SetPassword ... 688
19.13.16 UnblockUser ... 689
19.14 Module Activity Functions ... 690
19.14.1 Module Activity Functions – General Information ... 690
19.14.2 AppActivate ... 692
19.14.3 AppIsRunning ... 693
19.14.4 AppPostMessage ... 694
19.14.5 AppSendKeys ... 695
19.14.6 CleanReadQueue ... 696
19.14.7 CloseSplashWindow ... 696
19.14.8 DisableMath ... 697
19.14.9 EnableMath ... 697
19.14.10 EndTask ... 697
19.14.11 ExitWindows ... 698
19.14.12 IsScreenOpen ... 699
19.14.13 IsTaskRunning ... 699
19.14.14 IsViewerInFocus ... 700
19.14.15 KeyPad ... 700
19.14.16 LogOff ... 702
19.14.17 LogOn ... 702
19.14.18 Math ... 703
19.14.19 PostKey ... 703
19.14.20 Recipe ... 704
19.14.21 Report ... 705
19.14.22 RunGlobalProcedureAsync ... 706
19.14.23 RunGlobalProcedureAsyncGetStatus ... 708
19.14.24 RunGlobalProcedureOnFalse ... 710
19.14.25 RunGlobalProcedureOnServer ... 711
19.14.26 RunGlobalProcedureOnTrigger ... 712

Page

19.14.27 RunGlobalProcedureOnTrue ... 714
19.14.28 RunVBScript ... 715
19.14.29 SecureViewerReload ... 716
19.14.30 SendKeyObject ... 716
19.14.31 SetAppPath ... 719
19.14.32 SetViewerInFocus ... 720
19.14.33 SetViewerPos ... 720
19.14.34 ShutDown ... 721
19.14.35 StartTask ... 721
19.14.36 ViewerPostMessage ... 722
19.14.37 Wait ... 723
19.14.38 WinExec ... 724
19.14.39 WinExecIsRunning ... 726
19.15 File functions ... 727
19.15.1 File functions - General information ... 727
19.15.2 DeleteOlderFiles ... 728
19.15.3 DirCreate ... 728
19.15.4 DirDelete ... 729
19.15.5 DirLength ... 730
19.15.6 DirRename ... 730
19.15.7 FileCopy ... 731
19.15.8 FileDelete ... 732
19.15.9 FileLength ... 732
19.15.10 FileReadFields ... 733
19.15.11 FileReadMessage ... 734
19.15.12 FileRename ... 736
19.15.13 FileWrite ... 736
19.15.14 FileWriteFields ... 737
19.15.15 FileWriteMessage ... 739
19.15.16 FindFile ... 740
19.15.17 FindPath ... 741
19.15.18 GetFileAttributes ... 742
19.15.19 GetFileTime ... 742
19.15.20 GetHstInfo ... 743
19.15.21 GetLine ... 744
19.15.22 HST2TXT ... 746
19.15.23 HST2TXTIsRunning ... 748
19.15.24 ImportXML ... 749
19.15.25 LookupContains ... 751
19.15.26 LookupGet ... 751
19.15.27 LookupLoad ... 752
19.15.28 PDFCreate ... 752
19.15.29 Print ... 753
19.15.30 RDFileN ... 754
19.15.31 WebGetFile ... 755
19.16 Graphic Functions ... 756

Page

19.16.1 Graphic functions – General Information ... 756
19.16.2 AutoFormat ... 756
19.16.3 ExecuteCustomCall ... 757
19.16.4 GetScrInfo ... 758
19.16.5 PrintSetup ... 760
19.16.6 PrintWindow ... 761
19.16.7 ResetDecimalPointsTable ... 762
19.16.8 RGBColor ... 763
19.16.9 RGBComponent ... 763
19.16.10 SaveScreenShot ... 764
19.16.11 SetDecimalPoints ... 767
19.16.12 SetDisplayUnit ... 768
19.16.13 SetTagDisplayUnit ... 768
19.17 Translation functions ... 769
19.17.1 Translation functions – General information ... 769
19.17.2 Ext ... 770
19.17.3 SetLanguage ... 770
19.17.4 SetTranslationFile ... 771
19.17.5 TranslationLoad ... 772
19.17.6 TranslationLookupClose ... 773
19.17.7 TranslationLookupGet ... 774
19.17.8 TranslationLookupLoad ... 774
19.18 Multimedia Functions ... 775
19.18.1 Multimedia Functions – General Information ... 775
19.18.2 Play ... 775
19.19 System information functions ... 776
19.19.1 System information functions – General information ... 776
19.19.2 DbVersion ... 778
19.19.3 GetAppHorizontalResolution ... 778
19.19.4 GetAppPath ... 779
19.19.5 GetAppVerticalResolution ... 779
19.19.6 GetComputerIP ... 779
19.19.7 GetComputerName ... 780
19.19.8 GetCursorX ... 780
19.19.9 GetCursorY ... 780
19.19.10 GetDisplayHorizontalResolution ... 781
19.19.11 GetDisplayVerticalResolution ... 781
19.19.12 GetFreeMemoryCE ... 781
19.19.13 GetHardKeyModel ... 782
19.19.14 GetHardKeySN ... 782
19.19.15 GetIPAll ... 783
19.19.16 GetLicenseGracePeriod ... 784
19.19.17 GetMemoryCE ... 784
19.19.18 GetNetMACID ... 785
19.19.19 GetOS ... 786
19.19.20 GetPerformanceMetric ... 787

Page

19.19.21 GetPrivateProfileString ... 791
19.19.22 GetProductPath ... 791
19.19.23 GetRegValue ... 791
19.19.24 GetRegValueType ... 792
19.19.25 GetServerHostName ... 793
19.19.26 GetTickCount ... 794
19.19.27 InfoAppAlrDir ... 794
19.19.28 InfoAppHSTDir ... 794
19.19.29 InfoDiskFree ... 795
19.19.30 InfoResources ... 795
19.19.31 IsActiveXReg ... 796
19.19.32 IsAppChangedOnServer ... 796
19.19.33 NoInputTime ... 797
19.19.34 ProductVersion ... 798
19.19.35 RegSaveCE ... 798
19.19.36 ReloadAppFromServer ... 799
19.19.37 SaveAlarmFile ... 800
19.19.38 SetAppAlarmPath ... 800
19.19.39 SetAppHSTPath ... 801
19.19.40 SetDateFormat ... 801
19.19.41 SetKeyboardLanguage ... 802
19.19.42 SetRegValue ... 802
19.19.43 SetWebConfig ... 804
19.19.44 SNMPGet ... 806
19.19.45 SNMPSet ... 807
19.19.46 WritePrivateProfileString ... 808
19.20 Tags Database Functions ... 809
19.20.1 Tags Database Functions – General Information ... 809
19.20.2 ExecuteAlarmAck ... 811
19.20.3 ForceTagChange ... 812
19.20.4 GetAlarmCount ... 812
19.20.5 GetAlarmInfo ... 813
19.20.6 GetTagValue ... 815
19.20.7 SetTagValue ... 816
19.20.8 TagsDBAddClass ... 817
19.20.9 TagsDBAddClassMember ... 817
19.20.10 TagsDBAddTag ... 819
19.20.11 TagsDBBeginEdit ... 820
19.20.12 TagsDBEndEdit ... 821
19.20.13 TagsDBGetAlarm ... 822
19.20.14 TagsDBGetClassMember ... 825
19.20.15 TagsDBGetClassMemberCount ... 826
19.20.16 TagsDBGetFirstClass ... 826
19.20.17 TagsDBGetFirstClassMember ... 827
19.20.18 TagsDBGetFirstTag ... 828
19.20.19 TagsDBGetNextClass ... 829

Page

19.20.20 TagsDBGetNextClassMember ... 829
19.20.21 TagsDBGetNextTag ... 830
19.20.22 TagsDBGetTagCount ... 831
19.20.23 TagsDBGetTagProperty ... 832
19.20.24 TagsDBGetTrend ... 833
19.20.25 TagsDBRemoveAlarm ... 834
19.20.26 TagsDBRemoveClass ... 836
19.20.27 TagsDBRemoveClassMember ... 836
19.20.28 TagsDBRemoveTag ... 837
19.20.29 TagsDBRemoveTrend ... 838
19.20.30 TagsDBSetAlarm ... 839
19.20.31 TagsDBSetTagProperty ... 842
19.20.32 TagsDBSetTrend ... 844
19.21 Loop Functions ... 846
19.21.1 Loop Functions – General Information ... 846
19.21.2 For … Next ... 846
19.22 ODBC Functions ... 847
19.22.1 ODBC Functions – General Information ... 847
19.22.2 ODBCBeginTrans ... 848
19.22.3 ODBCBindCol ... 849
19.22.4 ODBCCanAppend ... 850
19.22.5 ODBCCanTransact ... 850
19.22.6 ODBCCanUpdate ... 851
19.22.7 ODBCClose ... 851
19.22.8 ODBCCommitTrans ... 852
19.22.9 ODBCDelete ... 852
19.22.10 ODBCExecuteSQL ... 853
19.22.11 ODBCInsert ... 853
19.22.12 ODBCIsBOF ... 854
19.22.13 ODBCIsDeleted ... 854
19.22.14 ODBCIsEOF ... 855
19.22.15 ODBCIsFieldNULL ... 856
19.22.16 ODBCIsFieldNullable ... 856
19.22.17 ODBCMove ... 857
19.22.18 ODBCMoveFirst ... 857
19.22.19 ODBCMoveLast ... 858
19.22.20 ODBCMoveNext ... 858
19.22.21 ODBCMovePrev ... 859
19.22.22 ODBCOpen ... 859
19.22.23 ODBCQuery ... 860
19.22.24 ODBCRollBack ... 861
19.22.25 ODBCSetFieldNULL ... 861
19.22.26 ODBCSetFilter ... 862
19.22.27 ODBCSetSort ... 863
19.22.28 ODBCUnbindCol ... 863
19.22.29 ODBCUpdate ... 864

Page

19.23 Email Functions ... 864
19.23.1 Email Functions – General Information ... 864
19.23.2 CnfEmail ... 865
19.23.3 GetStatusSendEmailExt ... 867
19.23.4 SendEmail ... 868
19.23.5 SendEmailExt ... 869
19.24 Dial-up Functions ... 871
19.24.1 Dial-up Functions – General Information ... 871
19.24.2 DialError ... 871
19.24.3 DialGetClientIP ... 877
19.24.4 DialGetServerIP ... 879
19.24.5 DialStatus ... 880
19.24.6 DialUp ... 882
19.24.7 DialUpToCE ... 884
19.24.8 FindAllDevices ... 885
19.24.9 FindModem ... 886
19.24.10 HangUp ... 886
19.24.11 PhoneDialUp ... 887
19.24.12 PhoneDisableListen ... 888
19.24.13 PhoneEnableListen ... 888
19.24.14 PhoneHangUp ... 889
19.24.15 PhoneStatus ... 890
19.25 ActiveX and .NET control functions ... 890
19.25.1 ActiveX and .NET control functions – General Information ... 890
19.25.2 XGet ... 891
19.25.3 XRun ... 892
19.25.4 XSet ... 893
19.26 Event Logger Functions ... 894
19.26.1 Event Logger Functions – General Information ... 894
19.26.2 SendEvent ... 894
19.27 FTP Functions ... 895
19.27.1 FTP Functions – General Information ... 895
19.27.2 CnfFTP ... 895
19.27.3 FTPGet ... 896
19.27.4 FTPPut ... 898
19.27.5 FTPStatus ... 899
19.28 Database/ERP Functions ... 900
19.28.1 Database/ERP Functions – General Information ... 900
19.28.2 DBCursorClose ... 901
19.28.3 DBCursorColumnCount ... 902
19.28.4 DBCursorColumnInfo ... 903
19.28.5 DBCursorCurrentRow ... 904
19.28.6 DBCursorGetValue ... 905
19.28.7 DBCursorMoveTo ... 905
19.28.8 DBCursorNext ... 906
19.28.9 DBCursorOpen ... 907

Page

19.28.10 DBCursorOpenSQL ... 909
19.28.11 DBCursorPrevious ... 911
19.28.12 DBCursorRowCount ... 912
19.28.13 DBDelete ... 913
19.28.14 DBExecute ... 914
19.28.15 DBInsert ... 916
19.28.16 DBSelect ... 917
19.28.17 DBUpdate ... 919
19.28.18 SyncAlarm ... 920
19.28.19 SyncAlarmStatus ... 921
19.28.20 SyncEvent ... 921
19.28.21 SyncEventStatus ... 922
19.28.22 SyncTrend ... 923
19.28.23 SyncTrendStatus ... 924

20 Appendix: VBScript language ... 925
20.1 Overview of VBScript ... 925
20.2 VBScript interfaces in the software ... 925
20.2.1 VBScript interfaces in the software - General information ... 925
20.2.2 Global procedures ... 927
20.2.3 Graphics module ... 930
Graphic Script ... 930
Screen Script ... 932
Command animation ... 933
ActiveX events ... 934
20.2.4 Background task ... 936
20.3 Language reference ... 936
20.3.1 Language reference – Overview ... 936
20.3.2 Operators ... 937
20.3.3 Constants ... 938
20.3.4 Objects and Collections ... 941
20.3.5 Properties ... 942
20.3.6 Statements ... 943
20.3.7 Methods ... 945
20.3.8 Functions ... 946
20.3.9 Keywords ... 946
20.3.10 Errors ... 947
20.3.11 List of system tags ... 949
20.4 Tips & tricks ... 951
20.4.1 Tips & tricks – Overview ... 951
20.4.2 VBScript Editor IntelliSense ... 952
20.4.3 VBScript compared to VBA ... 954
20.4.4 Screen Events ... 955
20.4.5 MsgBox and InputBox Functions ... 956
20.4.6 VBScript Procedures ... 956
20.4.7 Creating Constants ... 957

Page

20.4.8 Declaring Variables ... 958
20.4.9 Scope and Lifetime of Variables ... 958
20.4.10 Boolean tags and Boolean variables ... 959
20.4.11 Writing Real values to Integer tags ... 960
20.4.12 Precedence of VBScript Operators ... 960
20.4.13 Logical operator NOT ... 961
20.4.14 Using Conditional Statements ... 962
20.4.15 Looping with Code ... 963
20.4.16 Support for ActiveX Control Elements ... 966
20.4.17 Windows Embedded Support ... 966
20.5 Troubleshooting VBScript ... 966
20.5.1 Troubleshooting VBScript, General information ... 966
20.5.2 The Debug tab ... 967
20.5.3 Setting breakpoints in your VBScript code ... 968
20.5.4 Running the project in Debug mode ... 970
20.5.5 Monitor the current status in the Database spy window ... 972
20.5.6 Stepping through your VBScript code ... 973

21 Appendix: Driver error codes ... 977

22 Service and support ... 979

Index ... 981

1 About this documentation

Editions of this documentation

Edition	Release Date	Note
01	2015-10	First edition 14VRS
02	2015-12	WinStudio v7.3+SP4, work status
03	2016-08	WinStudio v7.3+SP4, work status, Update: functions_translation_setlanguage, tag properties New: WebClient license, ViewerClient license
04	2017-01	WinStudio v7.3+SP4 changed in WinStudio v7.4, Extended functional scope of the WinStudio screen conversion
05	2017-10	Chapter “License types" -> Note added Chapter "Screen groups folder" revised
06	2018-03	New chapter: 5.10 Text search / 8.2.6 Using user-defined property values when opening image objects / 8.4 Data entry / 11.5.5 Activating the OPC UA trace log
07	2019-02	**Additions:** Information about IT security • "IT security" on page 461 • "IT security" on page 85
08	2019-04	**Additions:** Requirements to use the process variable browser, see "Requirements" on page 423

Tab. 1-1: Change Record

1.1 Validity of this documentation

This "User manual and technical reference" has been designed to achieve the best results using the "Rexroth WinStudio"™ software. This document provides technical information and step-by-step instructions for the creation of web-capable HMI/SCADA programs.

Target group

This "User manual and technical reference" is a comprehensive document designed to provide useful information for both new and experienced users of "WinStudio".

- **New users**: This publication uses a step-by-step, hands-on approach to the project development process. Read the introduction describing the product's features and development environment.
- **Experienced users**: The document contains instructions for specialists as well as tips and information for troubleshooting.

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

1.2 Documentation structure

The information in this document is organized into the following sections:

- chapter 2 "About this application" on page 7: Includes a product overview, a list of important product features as well as information on the software functionality
- chapter 3 "Installation" on page 21: Includes an installation instruction for "Rexroth WinStudio", Engineering, EmbeddedView and CEView
- chapter 4 "Licensing" on page 33: Describes the type of license protection, license settings, activation of WinStudio licenses and the WinStudio behavior in case of invalid or insufficient licenses
- chapter 5 "Development environment" on page 39: Describes the "WinStudio" interface (or "development environment") and explains some basic skills and techniques you have to understand before creating a new project.
- chapter 6 "WinStudio projects" on page 85: Provides step-by-step instructions for creating and configuring a new project
- chapter 7 "Working with tags" on page 103: Describes basic concepts about the project database, tag types (arrays, classes and pointers), tag values and parameters. Besides conceptional considerations, creating and processing of tags in projects is described
- chapter 8 "Screens and graphics" on page 169: Describes the development tools of "WinStudio" for creating individual images and graphics
- chapter 9 "Alarms, events and trends" on page 313: Describes the application of alarm and trend tasks to log historical data and the use of the objects alarm/event and trend for display of historical data in a project image
- chapter 10 "Background tasks" on page 391: Explains how to create and configure the different "WinStudio" "task" worksheets for your projects
- chapter 11 "Communication with other devices" on page 413: Describes the configuration of projects for reading and writing process data and provides instructions for driver configuration and OPC, TCP/IP and DDE communication

  New: Inserting tags using the tag wizard, adding classes using the class wizard
- chapter 12 "Project security" on page 461: Explains how to set-up and manage a security system for your projects
- chapter 13 "Project localization" on page 487: Describes the translation of user texts with the support of the WinStudio translation tool
- chapter 14 "Testing and debugging projects" on page 497: Describes the test and debugging process for projects by means of tools like the database spy and output windows
- chapter 15 "Web thin client and secure viewer" on page 505: Describes configuration and execution of projects as web application
- chapter 16 "Downloading to a remote device" on page 545: Describes transmission, monitoring and debugging in projects via remote access
- chapter 17 "Database interface" on page 553: Explains how to connect "WinStudio" to compatible databases
- **Scripting languages**
  - chapter 19 "Appendix: WinStudio-internal scripting language" on page 595: Describes all functions of the internal WinStudio script language. Each functional description includes complete syntax, possible output values and examples of usage

– chapter 20 "Appendix: VBScript language" on page 925: "WinStudio" implements Visual Basic Scripting Edition 5.5 or higher

  A overview of the use of VBScript commands in "Rexroth WinStudio" is provided. For quick lookup, the most commonly used features of the language are summarized. For a complete description of the language as well as the interfaces and functions, please refer to Microsoft Developer Network

- chapter 18 "Debugging and FAQs" on page 585: Provides instructions for verifying projects, describes some common development errors and explains what to do if you need to contact a support representative

1.3 Using safety instructions

1.3.1 Structure of the safety instructions

The safety instructions are structured as follows:

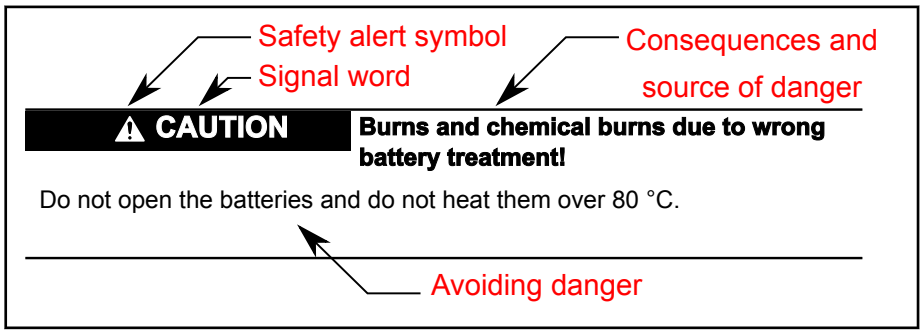


Fig. 1-1: Structure of the safety instructions

1.3.2 Explaining signal words and safety alert symbol

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

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

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

DANGER

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

WARNING

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

⚠ CAUTION

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

NOTICE

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

1.3.3 Symbols used

Pointers are displayed as follows:

 This is a note.

Tips are displayed as follows:

 This is a tip.

1.3.4 Explaining the signal alert symbol on the device



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

1.4 Conventions this documentation

In this documentation, standardized formatting and terminology is used for easier understanding.

Text conventions

In this documentation, special text formatting is used to support quick identification of certain terms:

- Titles, names, new terms and messages are indicated using "monospace" text (for example, "Object Properties").
- File names, screen text and text to be entered are indicated as program type (for example, `D:\Setup.exe`).
- Buttons, menu options and keyboard keys are indicated using a **bold** typeface (for example, **File** menu).

Mouse and selection conventions

In this documentation, the following mouse and selection conventions are used:

A PC mouse configured for right-handed use is described, so that the left mouse button is the primary button and the right mouse button is the secondary button.

- **Click** and **Select** both mean to click once on an element with the left mouse button. In general, "click" on buttons and "select" from menus.
- **Double-click** means to quickly click twice on an element with the left mouse button.
- **Right-click** means to click once on an element with the right mouse button.
- **Select** also refers to using the cursor to highlight or specify an element on the computer screen.

  Selecting an element with a touch screen is usually the same as selecting with a mouse, except that screen objects or sections are touched (selected) with the finger.

  To select elements by keyboard, the Tab key is typically used to move around options, the Enter key to open menus and the <Alt> key with a letter key to select an object that has an underlined letter.

- **Drag** refers to pressing down the respective mouse button and move the mouse before releasing the button. Usually an outline of the element moves with the mouse cursor.

Windows conventions

In this documentation, the following Windows conventions are used:

- **Dialogs** are windows that enable configuration of settings and entering information.
- **Text fields** are areas in dialogs where text can be typed in.
- **Radio buttons** are white circles in which a black dot appears or disappears when it is clicked on.

  The dot indicates that the option or function is "enabled" (selected). No dot indicates the option or function is "disabled" (not selected).
- **Checkboxes** are white squares in which a check appears or disappears when it is clicked on with the cursor.

  The check indicates that the option or function is "enabled" (selected). No check indicates that the option or function is "disabled" (not selected).
- **Buttons** are icons in boxes that appear "pressed" when they are clicked on.
- **Lists** are panes (white boxes) in windows or dialogs containing two or more selectable options.
- **Combo boxes** have an arrow that, when clicked, displays a part or all of an otherwise concealed list.
- **Dockable windows** are windows that can be dragged to an edge of the user interface and dock to that edge.

1.5 Required and supplementing documentation

In addition to this technical reference, consult the following manuals, if necessary:

"Drivers user manual": Explain how to configure individual direct communication drivers, according to their unique protocol characteristics. One customized user manual is included with each driver.



All manuals are located in the "Documentation" folder on the "WinStudio" installation CD.

Technical information can also be accessed from the **Help** menu:

IndraWorks online help ▸ IndraWorks components ▸ HMI tools ▸ HMI connection ▸ Communication interfaces ▸ BRWS driver (Bosch Rexroth WinStudio driver)

2 About this application

"Rexroth WinStudio" (or simply "WinStudio" or Studio) is a powerful, fully integrated software program that enables designing and creating "HMI" (Human-Machine Interface) or "SCADA" (Supervisory Control and Data Acquisition) projects for:

- Data acquisition
- Local supervisory stations
- Remote supervisory stations
- Data concentration on distributed processes
- Data communications with corporate systems

2.1 Product Overview

"Rexroth WinStudio" projects run on visualization devices (e.g. IndraControlV) connected in real-time to machines or processors through programmable controls, remote I/O devices, and other data-acquisition equipment.

These projects consist of animated images, configurable PLC (programmable logic control) drivers and other controllable I/O devices, a project tags database, and optional modules such as alarm monitoring, logic, trend charts, recipes, schedulers, and a security system. "WinStudio" projects communicate with industrial I/O systems and other Windows applications in the runtime environment using the following protocols:

- ODBC ("Open Database Connectivity")
- DDE ("Dynamic Data Exchange")
- NetDDE ("Network Dynamic Data Exchange")
- OPC ("Open Connectivity")
- TCP/IP ("Transmission Control Protocol/Internet Protocol")

After a project is created, it can be executed in the development environment or (via a TCP/IP connection) downloaded to the target device and executed with the "Rexroth WinStudio" runtime software.

The processes of the runtime environment communicate with the connected devices and process or visualize the projected data.

The product consists of two parts:

- Development environment: The WinStudio development environment is part of IndraWorks Engineering and can be run on desktop PCs, notebooks or industrial PCs if a supported Microsoft Windows operating system for desktop or server is installed.
- Runtime system software: Operable on visualization devices (e.g. IndraControlV) if a currently supported Microsoft Windows desktop operating system or Windows Embedded is installed.

The runtime client for Windows CE operating systems (CEView) is often pre-installed on the HMI.

If necessary, the CE View version can be updated by downloading the current version from the development computer to the HMI.

2.2 Product features

"Rexroth WinStudio" provides the following features:

- Integrated Development environment on page 39 in IndraWorks Engineering

  with editors, toolbars, dialogs and menus:

  - Customizable toolbars
  - Tasks, objects and controls displayed in the Project Explorer Project Explorer on page 45
- Versatilely configurable objects and animations (the ability to modify object properties, execute commands or write values to tags used to create display screens at application runtime):
  - Configurable objects such as buttons, rectangles, ellipses, polygons, lines and text
  - Object animations on page 246 such as bar graphs, colors, resizing, change in position, hiding/showing, rotation, command execution, hyperlink and text input/output
  - Online and historical display of alarm lists on page 313
  - Online and historical trends on page 340
  - The Formatting screen objects on page 265 is supported by the toolbars "Alignment functions" and "Object properties". Tools are provided to arrange, reposition and resize selected screen objects as well as to modify the objects' styles and fonts.
  - Background color or picture of a WinStudio screen on page 173 can be edited or created.
  - Graphics import
  - Containers for ActiveX objects on page 231 and .Net controls
- Remote management and configuration of the runtime environment
- Microsoft DNA architecture compliance, with full OPC and XML support
- Activation of a web interface to convert images to an internet-capable format and to display them by a client (web-browser). Data is exchanged online via the TCP/IP protocol
- Symbols library on page 222 with more than 100 available objects, such as pushbuttons, measuring instruments, sliders, switches, text and numeric displays, LED displays, lines, pumps, icons, vehicles, valves, frames, motors, gauges and controls
- Troubleshooting tools:
  - Database Spy window on page 78 to monitor/set tag values and execute functions
  - LogWin module on page 79 to record OPC, DDE and TCP/IP protocols, to activate modules, trace tags, etc.
  - Cross reference search for tags within a WinStudio application
  - System and network diagnostics at runtime
- Powerful and flexible tag database (Boolean, Integer, Real and String tags), array tags, classes and indirect tags (pointers)
- Open architecture with API, exchange of tag values with external software
- Translation editor on page 487 for translation of a project to various languages and switching between the languages at project runtime

- Client and server communication based on TCP/IP on page 453 to exchange tag values and configure redundant systems
- Drivers for direct communication on page 413 to connect to Rexroth controls (BR_WS)
- OPC server and OPC client with integrated OPC browser on page 427
- Runtime security on page 461 with password protection for images and objects (256 levels)
- Logical expressions and a scripting language on page 595 with more than 200 functions
- Tools to create recipes on page 395 and reports on page 396 (in ASCII, UNICODE and RTF format)
- Scheduler on page 403 for events based on date, time or data conditions (100 ms resolution)
- Import/export function for easy implementation of images and worksheets in other projects.
- Dial-up functions on page 595 to establish, monitor and disconnect dial-up connections with the RAS server of remote stations
- Functions to send e-mails on page 595 from "WinStudio" (or CEView)

2.3 How the software works

2.3.1 Internal structure and data flow

The "WinStudio" runtime environment is executed on a visualization device (Windows 2K/XP/Vista/7/CE) and consists of the following modules or "Threads" (program elements that can be executed independently of each other):

- **Background task** (supervisory task): Executes other internal tasks ("WinStudio" worksheets). For example, the "Background" Task executes scripts configured in the Math on page 400 and Scheduler on page 403 worksheets and manages parameters configured in the Alarm on page 313, Recipe on page 395, Report on page 396 and Trend on page 340 worksheets.
- **Database spy** (error analysis tool):
  - Executes functions and/or expressions for testing purposes
  - Reads data (such as tag values) from the "Tags" database
  - Writes data (such as tag values) to the "Tags"database
- **DDE client**: Manages the DDE communication with a DDE server (local or remote), according to parameters configured in the DDE client worksheets on page 455.
- **DDE server**: Manages DDE communication with a DDE Client (local or remote).
- **LogWin** (error analysis tool): Records messages generated from other modules/tasks.
- **Driver runtime program**: Manages the read/write commands configured in the Driver worksheets on page 413.
- **OPC client (OPC DA and OPC UA)**: Manages OPC communication with an OPC server (local or remote), according to parameters configured in the OPC client worksheets on page 427.
- **OPC DA server**: Manages OPC DA communication with an OPC DA client (local or remote)

- **ODBC runtime**: Manages ODBC data communication with any SQL relational database, according to parameters configured in the ODBC worksheets on page 397.
- **TCP/IP client**: Manages TCP/IP communication with a remote TCP/IP server module (from "WinStudio"), according to parameters configured in the TCP/IP client worksheets on page 453.
- **TCP/IP server**: Manages TCP/IP communication messages with a remote TCP/IP client module (from "WinStudio").
- **Viewer**: Executes all scripts (On Open, On While, On Close, Command, Hyperlink, etc.) configured for screens and updates the screen objects.

None of the preceding runtime modules exchanges data directly with another module or task data. Instead, the runtime modules transmit data to the "Tags" database or receive data from there. This can be referred to as the "heart" of "WinStudio".

The "Tags" database manages the flow of data between modules. In addition, the "Tags" database stores all tag values and the status of all properties associated with each tag (such as alarm conditioning, time stamp, quality and so forth).



Tags are variables (such as communication points to field equipment, calculation results, alarm points and so forth) that are used in screens and worksheets.

For detailed information on tags, tag values and tag properties, refer to Working with tags on page 103.

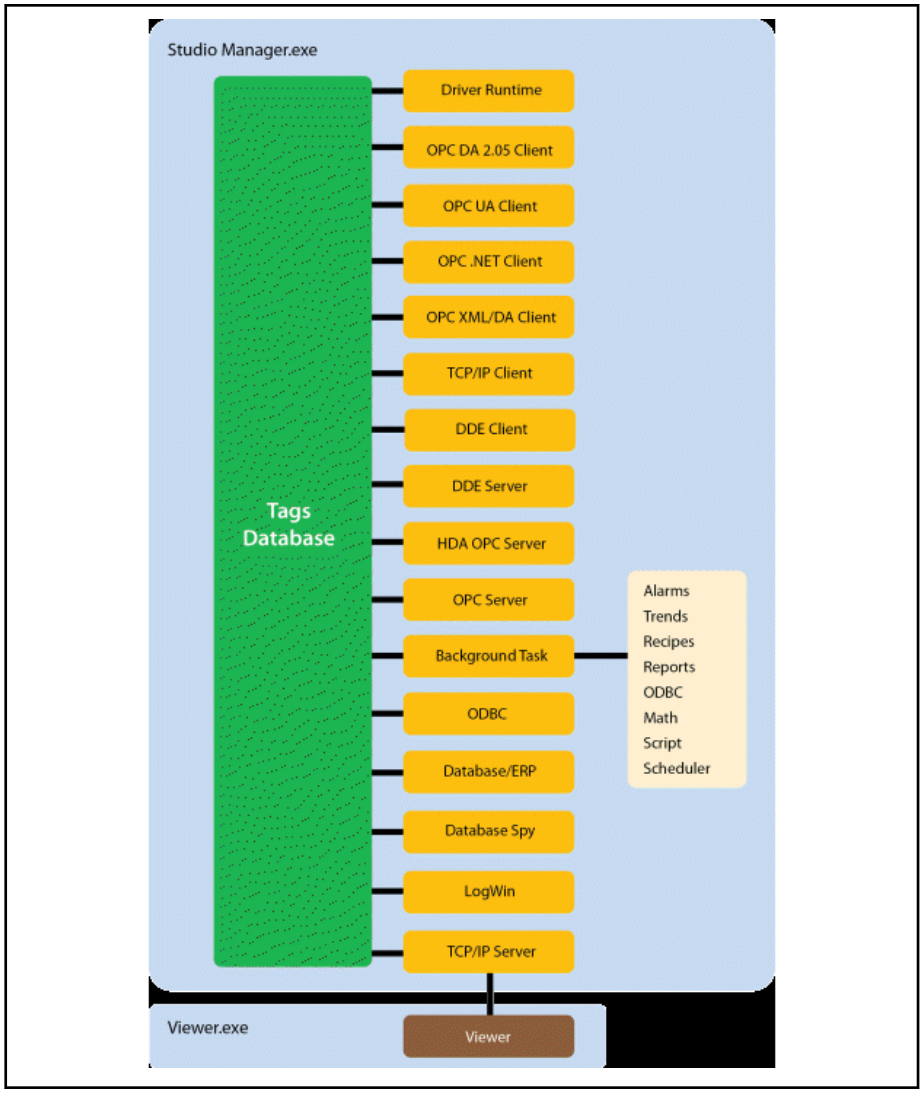


Fig. 2-1: Architecture of the project runtime environment under Windows

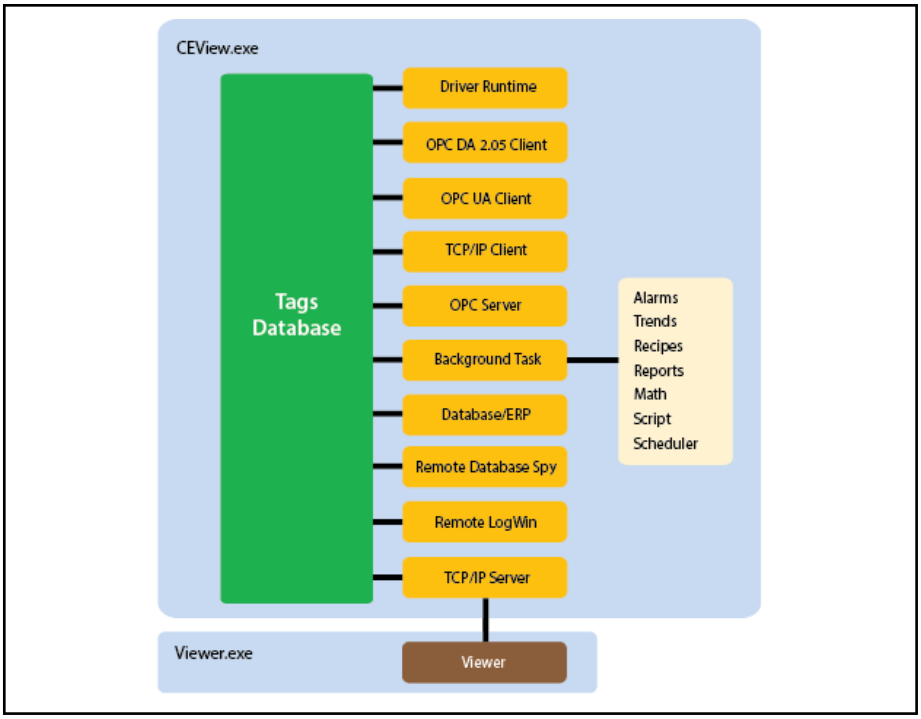


Fig. 2-2: Architecture of the project runtime environment under Windows Embedded

Each runtime module contains a virtual table of the tags that are currently relevant for that module. The tag database uses this table to determine which information has to be updated in each module. For example, the Viewer contains a virtual table that lists all tags configured for all of the open project screens. If a tag value changes, the tag database sends a message to the Viewer and then the Viewer updates the value in all objects for which the tag is configured.

For example, if a driver reads a new value from the PLC, the driver updates the tag associated with this value in the tag database. Then, if this new information has to be displayed on the project screen, the tag database sends the new tag value to the “Viewer” task and the “Viewer” updates the screen.

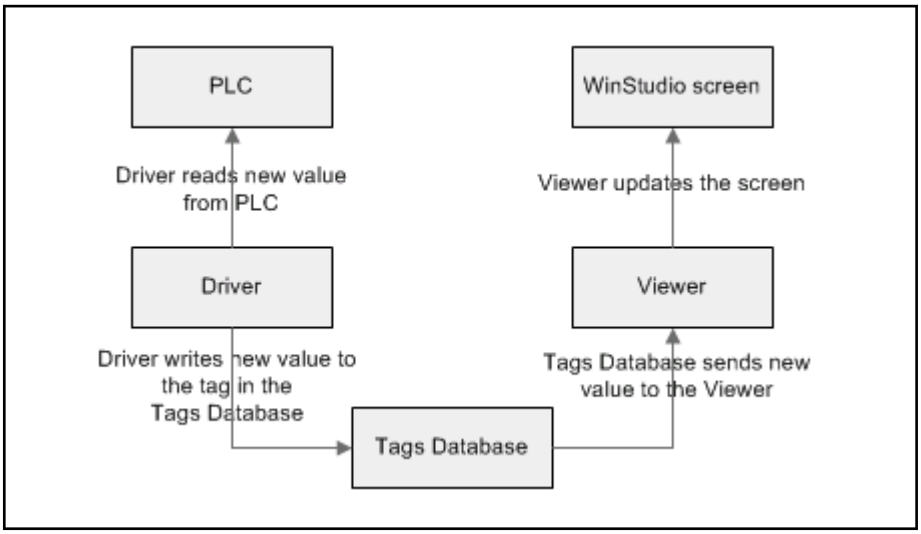


Fig. 2-3: Data flow example

Note that the driver does not send new tag values directly to the "Viewer". In addition, there is no polling between tasks – the "Tags" database receives the

updated information and immediately forwards it to all runtime tasks requiring that information.



The Viewer module only updates an object when (at least) one of the object's tag values change.

If an object animation (such as Text data link, on page 251) is configured with a function that does not require a tag (for example NoInputTime on page 797), the "Viewer" does not update the object.

The architecture of "WinStudio" significantly improves the internal data flow performance and makes it easy to add new internal tasks. Even though each task works independently, it can access information from any other task through the "Tags" database.

2.3.2 Executing/switching modules

"WinStudio" is a SCADA system composed of modules that have to be executed simultaneously. Based on the multi-tasking concept, each runtime module ("Viewer", "Driver", etc.) is a "thread" and the operating system automatically switches from one thread to another. It is a common misconception that a SCADA system is executed simultaneously with a PLC program.

A PLC program contains a simple loop:

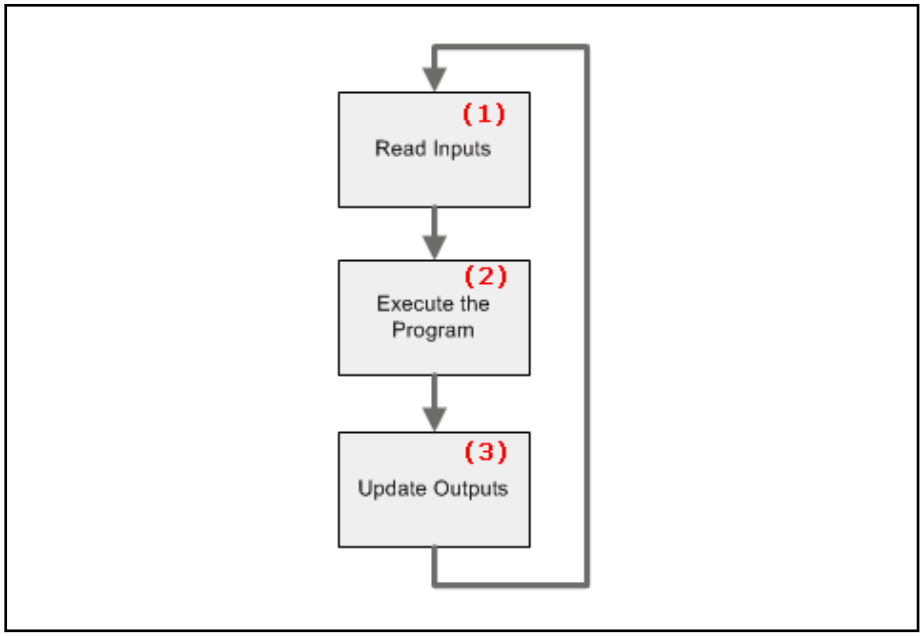


Fig. 2-4: PLC program loop

However, in a SCADA system, there are several modules running simultaneously, most of which can read and write data.

As a SCADA system continuously modifies data (tag values) during task execution, the preceding diagram is "not" applicable.

Rexroth WinStudio only has one runtime process: `Studio Manager.exe` under Windows or `CEView.exe` under Windows Embedded. When a project is executed, this process starts the tag database and all runtime modules configured for the project. It can be specified which modules (such as Viewer and Driver) start at runtime.

Each Windows process lists "active" threads for the operating system. Actually, each process activates and deactivates each thread at runtime, according

to the algorithm of each process. Also, specify a priority value when starting a thread. The operating system continuously scans all currently active threads and executes the threads according to their priority value - executing the higher-priority threads first. When threads with higher-priority values are active, the threads with lower-priority values are not executed at all. If there is more than one thread with the same priority value and there are no other threads with higher-priority values, the operating system keeps switching between the threads with the same priority.

☞ All "WinStudio" threads are set to priority "7", which is `THREAD_PRIORITY_NORMAL`. (Most programs contain this priority value.)

Real-time program (such as SoftPLCs and Device Drivers) threads are assigned a higher-priority value (`THREAD_PRIORITY_HIGHEST`); however, these programs have to provide a mechanism to keep them inactive for some period of time, otherwise the threads with normal priority are never executed.

"WinStudio" uses the `UNICOMM.DLL` library for serial drivers.

This library creates a `THREAD_PRIORITY_HIGHEST` thread that "sleeps" (remains inactive) until data arrives in the serial channel.

When "WinStudio" detects new data in the serial channel, the `THREAD_PRIORITY_HIGHEST` thread "wakes up" (becomes active) and transfers the data from the operating system buffer to the thread buffer, where it can be read by the Driver. This thread is the only highest-priority thread created by "WinStudio".

If threads are allowed to remain active all the time, the CPU usage is 100% all the time, which has to be avoided for performance reasons. Every program provides a mechanism to prevent threads from staying active permanently.

"WinStudio" uses the following parameters to prevent that threads permanently remain active:

- **TimeSlice** (from operating system): Causes the operating system to switch automatically between active threads with the same priority value.

  By default, the operating system executes each active thread for approximately 20 ms and then switches to the next active thread. In other words, if there are multiple active threads with the same priority value waiting to be executed, the operating system will not execute any one active thread for more than 20ms.

- **TimeSlice** (from "WinStudio"): Specifies how long each "WinStudio" thread can remain continuously active.

  This parameter is used in addition to the operating system's TimeSlice parameter. A TimeSlice value is configured for each "WinStudio" thread (except the Background Task) and specify how long each thread can remain continuously active. As long as a thread is active, the operating system can switch to that thread.

- **Period** (from "WinStudio"): Specifies the maximum amount of time each "WinStudio" thread (except the Background Task) can remain inactive.



It is "highly" recommended not to modify these default values unless it is absolutely necessary. Configuring any of these parameters incorrectly can cause the entire system to malfunction (for example, CPU load rises to100 %) and/or cause some tasks to perform poorly.

If the parameter defaults are to be changed, take a note of the values before changes are made to be able to return to the original settings in case of a malfunction.

To change the default values of the WinStudio parameters TimeSlice and Period:

1. From the `C:\Program Files\installation folder\Bin`), double-click `\BIN` to open the folder.
2. Double-click on the program file `Settings.INI` to open the file in Microsoft® Notepad.

   The following is a list of all parameters contained in this .ini file and their default values (in milliseconds).

Program Settings.INI

```
[Period]
DBSpy=1000
UniDDEClient=200
UniDDE=200
Driver=20
LogWin=100
UniODBCRT=100
OPCClient=20
OPCServer=20
TCPClient=100
TCPServer=100
Viewer=50

[TimeSlice]
UniDDEClient=100
Driver=10
OPCClient=10
OPCServer=10
TCPClient=200
TCPServer=200
Viewer=200
```

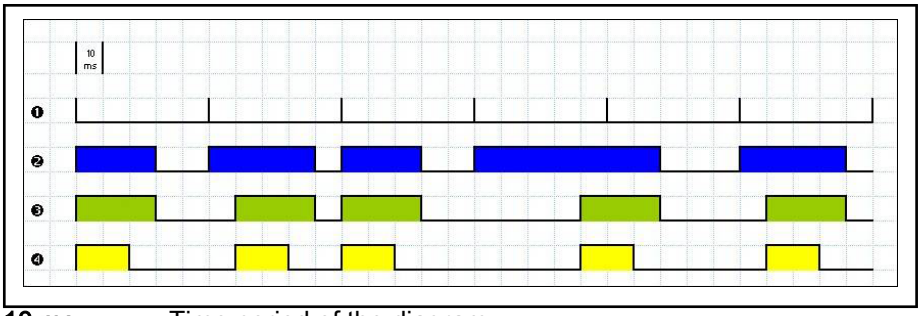


It may be the case that not all listed parameters can be found in the file `Program Settings.ini`. Nevertheless, "WinStudio" uses parameters not included and assigns default values to these parameters.

- To change the default values of displayed parameters, overwrite the current value and save the file.
- If a parameter value is to be changed that is not included in the file, add the parameter name and the desired value as illustrated in Program Settings.INI on page 15.

3. Save the file (**File** ▸ **Save**) and close Notepad (**File** ▸ **Exit**).

The following figure illustrates how "WinStudio" executes a generic thread (such as the "Viewer").



10 ms Time period of the diagram

Fig. 2-5: Executing a generic thread

The following apply here:

- Signal **1** is the "Period" time period (set to 50 ms for this example)
- Signal **2** displays when the thread for the operating system is active
- Signal **3** is the "TimeSlice" time period (set to 30 ms for this example)
- Signal **4** shows the execution of the thread itself

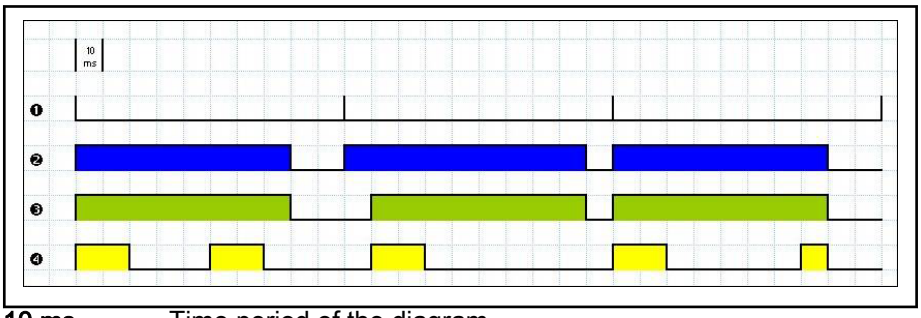
In this example, "WinStudio" generates a Period message every 50 ms (signal **1**). When "WinStudio" generates this message, its thread becomes active and remains active until the specified TimeSlice time period (from "WinStudio") expires. The thread then remains inactive until "WinStudio" generates the next Period message (signal **1**).

While the thread is active, the operating system is responsible for executing that thread. A thread being active does not mean that the operating system executes it immediately. For example, this is the case if the operating system is currently executing different threads.

When the operating system executes the thread, the "TimeSlice" timer starts counting and the thread is executed for 20 ms (TimeSlice of the operating system). After the 20 ms period, the operating system automatically switches to the next active thread (such as the driver) and so on.

In the above example, the TimeSlice time was set to 30 ms, which means the operating system is not supposed to execute the thread more than once in each "WinStudio" TimeSlice. However, if higher values are specified for the "WinStudio" TimeSlice time period, it is likely that the operating system executes the same thread more than once in the same TimeSlice time period.

In the next example, the Period and the TimeSlice values were changed as follows, but the default operating system TimeSlice period (20 ms) was not changed.



10 ms Time period of the diagram

Fig. 2-6: Setting a higher TimeSlice

The following apply here:

- Signal **1** is the Period time period (set to 100 ms)
- Signal **2** displays when the thread for the operating system is active

- Signal **3** is the "WinStudio" TimeSlice time period (set to 80 ms)
- Signal **4** shows the execution of the thread itself

Please note that the thread can be executed more than once in the same TimeSlice time period. When the "WinStudio" TimeSlice time period expires, the operating system interrupts the thread execution; however, even though the "WinStudio" Period and TimeSlice parameters are set to 100 ms and 80 ms respectively, the operating system does not execute this thread continuously for more than 20 ms, because the operating system TimeSlice time period is set to 20 ms.

If the operating system is not executing the "Viewer" thread, the CPU can execute any other thread or remain inactive (if there are no other active threads to execute). Remember: The "WinStudio" Period and TimeSlice parameters were created to prevent all threads from being active at the same time to prevent 100 % CPU usage.

During thread execution, the thread has to handle its pending messages. For example, the Viewer module has to update any related screen objects. If there are no messages pending, the thread deactivates itself and gives control back to the operating system. The operating system immediately switches to the next active thread. In other words, a thread can interrupt its own execution – even if the operating system TimeSlice time period has not yet expired (which occurs frequently in real-world applications).

☞ The "Database Spy", "DDE Server", "LogWin" and "ODBC Runtime" modules do not support a TimeSlice parameter. Thus, the thread becomes inactive after processing all pending messages until the next Period message for the respective thread occurs.

The background task is an exception of the described thread execution mechanism. The execution of the background task is described in the following.

2.3.3 Executing/switching the background task

The background task executes scripts from the "Math" and "Scheduler" worksheets (for example, messages from "Alarm" and "Trend" worksheets). In addition, the Background Task executes all recipe and report commands when the **Recipe()** or **Report()** functions are executed at runtime.

Although the "Alarm", "Math", "Scheduler" and "Trend" tasks are no threads, their Period time can be specified or changed in the `Program Settings.ini` file located in the "WinStudio" program directory.

The “Period” default values (in milliseconds) are as follows:

Program Settings.INI

```
[Period]
Math=100
Sched=50
Alarm=100
Trend=1000
```

These values mean that every 100 ms, "WinStudio" generates a “Period” message to the “Math” task. Every 50 ms, "WinStudio" generates a “Period” message and sends it to the “Scheduler” task and so on.

NOTICE **It is highly recommended that background task default values are not modified unless it is absolutely necessary.**

Configuring any of these parameters incorrectly can cause the entire system to malfunction (for example, CPU usage rises to 100%) and/or cause some tasks to perform poorly.

If the parameter default values are to be changed, take note of the values to be able to return to the original settings in case of a malfunction.

Keep in mind that the Background Task thread has the same priority as other threads in "WinStudio" ("Drivers", "Viewer", etc.), which means that the operating system does not execute this task continuously for more than 20 ms.

The background task executes the Recipe on page 395 and Report on page 396 worksheets when the Recipe on page 704 and Report on page 705 functions are called. As the `Recipe()` and `Report()` functions are "synchronous", once the Background Task starts executing the functions, it does not switch to another task ("Alarm", "Math", "Scheduler" or "Trend") until it has executed the functions completely. Executing a `Recipe()` or `Report()` function usually takes a few milliseconds.

The Background Task has to switch between the Alarm, Math, Scheduler and Trend tasks. When the background task switches to the Scheduler task, it does not switch to another task (Alarm, Math, or Trend) until all Scheduler worksheets are processed. After processing all Scheduler worksheets, the background task does not execute the Scheduler again until it receives the next Period message for the Scheduler task.

The background task applies the same behavior when executing the Alarm and Trend tasks — when the background task switches to the Alarm or Trend task, it does not switch to another task until all pending messages have been processed. The background task does not execute the Alarm or Trend task again until WinStudio generates the next Period message for each of these tasks.

The background task typically executes the Alarm, Scheduler and Trend tasks in a few milliseconds. However, it can take longer to execute the Math task because it usually contains loops and complex scripts. Consequently, the mechanism used to execute the Alarm, Scheduler and Trend tasks cannot be applied to the Math task.

The background task executes the Math task for no more than 10 ms continuously before switching to another task (such as the Scheduler). The background task cannot execute the Math task again for the next 50 ms; however, the Background Task can execute other tasks (Alarm, Recipe, Report, Scheduler, or Trend) during this 50 ms period. After the background task has executed all of the math worksheets, it does not begin a new scan of the math worksheets until WinStudio generated a new Period message for the math task.

It is important to re-emphasize that this process was created to prevent 100 % CPU usage all the time.

NOTICE

Be careful when using the Math on page 703 function in a Scheduler worksheet or for a screen object (such as command animation).

If the "Scheduler" task executes a `Math()` function, no other task can be executed by the Background Task until the "Scheduler" has executed the entire "math" worksheet called by the `Math()` function. This process can take several milliseconds or even seconds, depending on the configuration of the script in the math worksheet (especially for loops).

If a Math function for a screen object is configured, the Viewer stops updating the screen until the Viewer has executed the entire math worksheet called by the Math function.

If the Math function has to be used for the Scheduler task or a screen object, proceed as follows to prevent process delays:

1. Specify one auxiliary tag with the value 1 and the Scheduler or Viewer task will send a message to the tag database to update this tag value.
2. Configure the tag in the “Execution” field of the “Math” worksheet to be executed. When the background task scans the Math worksheet, WinStudio executes the worksheet.
3. Reset the tag in the last line of the Math worksheet (write the value 0 to the auxiliary tag).

As a result, the background task will not execute the Math worksheet in the next scan unless the auxiliary tag is set to the value 1 again.

3 Installation

This chapter provides information on the installation of Rexroth WinStudio, EmbeddedView and CEView.

3.1 Comparing the Rexroth WinStudio software components

The Rexroth WinStudio software suite includes several individual components which can be installed on different computers to perform different functions.

Component	Functions	Platforms
Rexroth WinStudio	Project development Tag integration Remote management Project runtime server Project runtime client	Windows Windows server Windows Embedded Standard
EmbeddedView (with remote agent)	Agent for remote management: Project runtime server (limited number of tags) Project runtime client	Windows Embedded Standard
CEView (with remote agent)	Agent for remote management: Project runtime server (limited number of tags) Project runtime client	Windows Embedded Compact
Web tunneling gateway (WTG)	Enables a public web server to transfer data between your project runtime server and Web Thin Clients if your project runtime server is in a secure internal network.	Microsoft IIS for Windows
Mobile Access Runtime	Enables your project runtime server to output project screens which are compatible with HTML5 to tablets and smartphones.	Microsoft IIS for Windows
Secure Viewer	Project runtime client as individual application.	Windows Windows server Windows Embedded Standard Windows Embedded Compact
Web Thin Client (also IS-Symbol)	Project runtime client as browser plug-in.	Internet Explorer for Windows

Tab. 3-1: Comparing the Rexroth WinStudio software components

The architecture of your completed WinStudio project depends on which components you install, where you install them and how you link them.

In most cases, the complete Rexroth WinStudio software is to be installed on your primary workstation first. In this way, not only your project development environment is installed on your computer but the rest of the components is unzipped as well so that they can be installed on other computers.

Separate installation programs for the Thin Clients are available for download.

3.2 Installing the complete Rexroth WinStudio software

Install the complete Rexroth WinStudio software on your Windows computer to develop WinStudio projects or to use the computer as a project runtime server or client.

The following is required to install and run the full Rexroth WinStudio software:

- A Windows-compatible computer with a standard keyboard, a pointer input (e.g. mouse, touch pad or touch screen) and a monitor which at least complies with the SVGA standard;
- One of the following Windows operating systems:
  - Microsoft Windows XP Service Pack 3
  - Microsoft Windows Vista Service Pack 2
  - Microsoft Windows 7 Service Pack 1
  - Microsoft Windows 8 or 8.1 (without Windows RT)
  - Microsoft Windows Server 2003 Service Pack 2
  - Microsoft Windows Server 2008 Service Pack 2
  - Microsoft Windows Server 2008 R2 Service Pack 1
  - Microsoft Windows Server 2012
  - Microsoft Windows XP Embedded Service Pack 3
  - Microsoft Windows Embedded Standard 7
- Microsoft .NET Framework 3.5.1 (refer to the following note);
- Microsoft Internet Explorer 6.0 or later
- 2 GB of free memory on the hard drive or a permanent memory; and
- An Ethernet or Wi-Fi network adapter for the TCP/IP connection.

We recommend using the Windows editions Home Premium, Professional, Enterprise or Ultimate, since the Microsoft Internet Information Services (IIS) are integrated as a pre-installed function which can be enabled. IIS can be used to make your project accessible to Thin Clients and mobile devices. We do not recommend using the editions Starter or Home Basic since IIS is not integrated in them. They can be used, however, if no web-based functions are planned.

The following elements are optional, but recommended:

- A DVD ROM drive for the installation of the software from CD.

  This is optional since the installation program can also be downloaded on your computer.
- A USB or serial port for use with the hard key license of the software.

  This is optional since a softkey license is also available.
- Serial COM ports and adapters for direct communication with PLCs and other devices.

  This is optional since many more recent device protocols use TCP/IP communication instead of serial communication.
- Microsoft IIS installed and enabled to make your project accessible to mobile devices. For further information, refer to the description in the following section Mobile Access Runtime.

  This is optional since the Mobile Access Runtime function does not absolutely have to be installed.

- Microsoft Visual Studio Team Explorer 2010 installed to enable the cooperation of workgroups and version control. For further information, refer to the description in the following section Cooperation.

  This is optional since the Cooperation function does not absolutely have to be installed.



- Administrator rights are required on the PC in order to install the software.
- The so-called Microsoft "Mainstream support" only applies to Windows 7, Windows 8, Windows Server 2008 R2 and Windows Server 2012. This means that these systems are actively maintained and additional service packs can be released for the in the future. The rest of the listed operating systems is subject to the so-called “Extended Support” by Microsoft. This means that they are no longer actively maintained.
- Rexroth WinStudio can be installed on Windows XP 64 bit but we do not recommend it.
- Even if the complete Rexroth WinStudio software can be installed on a computer with Windows XP Embedded or Windows Embedded Standard 7, as long as you do not plan on developing projects on this computer and only use it as a project runtime server or client, consider installing EmbeddedView instead. EmbeddedView does not support as many project tags as the complete software but required fewer system resources and can be remotely installed and managed.
- Microsoft .NET Framework 3.5.1 is pre-installed with newer versions of Windows but may not be enabled. To enabled it, use Programs and Features (**Start ▸ Control Panel ▸ Programs ▸ Programs and Features ▸ Turn Windows features on or off**).

For older Windows versions, Microsoft.NET Framework 3.5.1 is to be installed separately. The Rexroth WinStudio installation program tries to do this for you (see step 2 below) but, depending on the security settings of your computer, the installation may fail without you being informed about it. If problems occur later when you try to execute WinStudio, at least check if Microsoft .NET Framework 3.5.1 has been installed successfully. To do this in Windows XP or Windows Server 2003, use Add/Remove programs in the Control Panel (**Control Panel ▸ Add/Remove programs**).

Please note that Microsoft .NET Framework 3.5.1 is not integrated in Microsoft .NET Framework 4.

Installing the complete Rexroth WinStudio software:

1. Do one of the following:
   - Download the zipped installation program on your computer either from our website (www.boschrexroth.com) or from another location in your network where it was saved before. Unzip the zip archive, open the folder inside and open `setup.exe` by a double-click.

- Insert the installation CD into your CD-ROM drive. If it does not start automatically, start it manually `D:\bosch.htm`. If the page opens in your browser, follow the instructions for product installation.

The installation wizard is started and asks you to select a language for the installation.

2. Select a language from the list and click on **OK**. The selection determines which language the installation wizard uses and in which language the user interface of the project development application is displayed. The language of the user interface can be changed after the installation of the software. The wizard prepares the installation. In this step, it automatically installs the Sentinel drivers from SafeNet (as a part of the software licensing mechanism) as well as Microsoft.NET Framework 3.5.1.
3. Click on **Next** on the initial page of the wizard to continue with the installation.
4. On the license agreement page, either click **Yes** to accept the agreement and continue or **No** to reject the agreement and exit the wizard.
5. On the client information page, enter your user name and the name of your company and click **Next**.
6. On the select location page, select the folder in which the software is to be saved and click **Next**. The software is installed by default under: `C:\Program Files\Rexroth\WinStudio\`
7. On the features page, select the features and components which are to be installed on your computer and click **Next**. The following functions are available:

Program files

The main program files for project development application, project runtime server and project runtime client. The selection of this feature cannot be undone.

Demos

Pre-configured projects which demonstrate the functions of the Rexroth WinStudio software.

Hard key

Support of additional drivers to support the use of hardkey licenses.

OPC components

Additional components for the communication with other OPC-compliant devices. This includes OPC DA (also OPC Classic), OPC UA, OPC .NET (also OPC Xi) and OPC XML-DA.

PDF print

Software for printing runtime reports directly as PDF files.

Security system device driver

An additional keyboard driver to verify the user input for increased project security.

Symbols library

A library of prepared but configurable screen objects such as pushbuttons, toggle switches, gages, dials, indicator lights and so on.

Windows CE Runtime

Also referred to as CEView – Project runtime software (server/client but not development) for Windows Embedded Compact on many different processors. Check in the documentation of your device which processor it uses.

☞ The selection of this function does not actually install CEView on your computer. The installation files are only unzipped and copied in your program folder so that you can install CEView on a Windows Embedded device later

Windows Embedded Runtime

Also referred to as EmbeddedView – project runtime software (server/client but not development) for computers with Windows Embedded Standard.

☞ The selection of this function does not really install EmbeddedView on your computer. The installation files are only unzipped and copied in your program folder so that you can install EmbeddedView later on a Windows Embedded Standard computer

Windows Mobile

Additional runtime software for older Windows mobile devices.

Mobile Access Runtime

Additional software for Microsoft IIS which makes your project runtime environment accessible to mobile devices such as tablets and smart phones.

For this feature, Microsoft IIS needs to be switched on and configured with ASP, ASP.NET and ISAPI extensions. The Rexroth WinStudio installation wizard tries to verify whether this is done and otherwise does not install this feature.

For further information, refer to, Enable Microsoft IIS for access of the Thin Client.

This feature does not have to be installed at this moment. It can be installed later after enabling IIS or it can be installed on a different computer which serves as runtime server of your project. There is a separate installation program for Mobile Access Runtime (`MobileAccessSetup.exe`) which is unzipped with the rest of the software.



Your software license for Rexroth WinStudio needs to have the required add-on so that you can use Mobile Access Runtime. It is also possible to develop projects which include the Mobile Access features without the license add-on but mobile devices cannot access them.

Cooperation

Additional tools for the cooperation of workgroups and version control in the development environment.

For this feature, Microsoft Visual Studio Team Explorer 2010 has to be installed on the same computer. The Rexroth WinStudio installation wizard tries to verify whether this is done and otherwise does not install this feature.

The Team Explorer module is integrated in some versions of Microsoft Visual Studio 2010. This means that if Visual Studio is already installed on your computer, it may be possible to select and install the Cooperation feature. If Visual Studio is not installed or if you are unsure whether your version of Visual Studio includes the Team Explorer module, it is possible to download and install Team Explorer 2010 separately and free of charge.

Download Team Explorer 2010 at www.microsoft.com/en-us/download/details.aspx?id=329

Note: Team Explorer 2012 and Team Explorer 2013 are not supported for the client regarding the cooperation feature at the moment. To use the cooperation feature, Microsoft Visual Studio Team Foundation Server 2010 or 2012 should also run somewhere in your network. If this is not the case, however, you can still install the cooperation feature now.

Note: Team Foundation Server 2013 is not supported for the server regarding the cooperation feature at the moment.

Your software license for Rexroth WinStudio needs to have the required add-on for cooperation so that you can use the cooperation feature. To purchase the add-on, please contact your software provider.

8. On the "Ready for installation" page, click on **Install**. The installation process is continued.
9. On the last page, click on **Finish**. The wizard is closed.

When the installation is finished the Rexroth WinStudio software should be located in your start menu under **Start** ▸ **All Programs** ▸ **Rexroth** ▸ **WinStudio**. It includes the following components:

Rexroth WinStudio v7.4

Project development application, project runtime server and project runtime client.

Register

A utility which manages your software license for Rexroth WinStudio.

StartUp

A shortcut which automatically starts the project runtime server and executes the most recently used project.

Remote Agent
A utility which ensures that Rexroth WinStudio which runs on other computers can connect to your computer and can send projects.

Quick Start Guide
A quick start guide for the installation and use of the project development application including a tutorial for the creation of a simple project.

Manual
A complete technical reference and user manual for the Rexroth WinStudio software.

Release Notes
A list of changes in the Rexroth WinStudio software.

A shortcut symbol should be on your desktop as well.

During installation, the following error message may be displayed:

Error 1628: Script-based installation could not be completed.

For further information on this error and possible remedies, refer to:

consumerdocs.installshield.com/selfservice/viewContent.do?externalId=Q108464

To execute the software, select one of the following methods:

- Double-click the shortcut icon on the desktop; or
- Click on **Start** ▸ **All Programs** ▸ **Rexroth** ▸ **WinStudio** ▸ **WinStudio v7.4**.

3.3 Installing EmbeddedView or CEView on a target device

Install EmbeddedView on a Windows Embedded Standard computer or CEView on a Windows Embedded Compact device to use it as project runtime server and/or client.

The complete Rexroth WinStudio software has to be installed already on your computer either using the installation CD or using the downloadable installation program since the EmbeddedView and CEView (Redistributable) software is in the Rexroth WinStudio program folder.

To install and execute EmbeddedView or CEView, the following is required:

- A computer or device (the target device) which is compatible with Windows Embedded;
- A Windows Embedded Standard or Windows Embedded Compact operating system currently supported by Microsoft, which includes at the moment:
  - Microsoft Windows XP Embedded Service Pack 3
  - Microsoft Windows Embedded Standard 7
  - Microsoft Windows Mobile or Windows CE 5.x/6.x
  - Microsoft Windows Embedded Compact 7
- 100 MB of free memory on the hard drive or a permanent memory (flash card or hard drive space) for the runtime environment. Depending on the size of your project, more space may be required.

- 32 MB of free RAM. Depending on the size of your project, more space may be required.
- An Ethernet or Wi-Fi network adapter for the TCP/IP connection.

The following elements are optional, but recommended:

- A USB port; and
- Serial COM ports and adapters for direct communication with PLCs and other devices.

  This is optional since many more recent device protocols use TCP/IP communication instead of serial communication.

Installing EmbeddedView or CEView on a target device is a process with two steps. First copy the Remote Agent utility to the target device and execute it. Using Remote Agent, a connection from the project development application to the target device can be established. Then the rest of the EmbeddedView or CEVIEW software can be installed via this connection.

☞ If EmbeddedView or CEView is pre-installed on the target device, skip these steps. Many hardware manufacturers pre-install Remote Agent on their devices as part of a larger Rexroth WinStudio package.

Installing EmbeddedView or CEView:

1. Start up the target device and ensure that it is connected to your TCP/IP network. If Remote Agent is pre-installed on your target device, it is executed automatically at start-up and the following step can be skipped. Many hardware manufacturers pre-install Remote Agent on their devices as part of a larger Rexroth WinStudio package.
2. Copy the Remote Agent utility to the target device and execute it:

   2.1 Browse for the correct version of the Remote Agent utility (CEServer.exe) for the target device. All versions are saved in your Rexroth WinStudio program folder.

   Remote Agent for Windows Embedded Standard can be found here:

   `C:\Program Files\Rexroth\WinStudio\Redist\WinEmbedded\Bin\CEServer.exe`

   Remote Agent for Windows Embedded Compact can be found here:

   `C:\Program Files\Rexroth\WinStudio\Redist\WinCE5.0\processor\Bin\CEServer.exe`

   ..the processor is the processor used by your device. For further information, refer to the manufacturer's documentation.

   2.2 Copy `CEServer.exe` to the target device by either downloading it via the network, transferring it to a USB flash drive or synchronizing it with Microsoft ActiveSync. (ActiveSync is also called Windows Mobile Device Center in Windows Vista or Zune software in Windows 7.) Save the file at any location on your target device as long as it is a permanent memory and not the main folder (i.e., C:\ or \).

   2.3 Set the Remote Agent so that it is automatically executed at start-up. With most Windows Embedded Compact devices, this can be achieved via a link in `\Windows\Startup`. If this does

not work for your device, please refer to the manufacturer's documentation.

2.4 Execute Remote Agent.

The Remote Agent window is displayed.

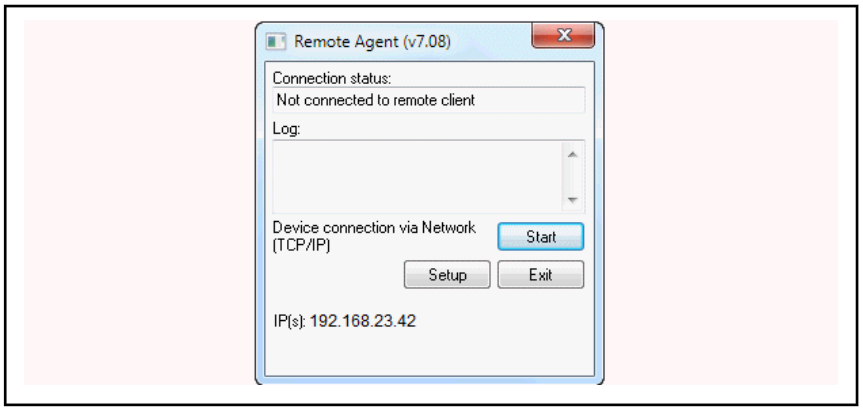


Fig. 3-1: "Remote agent" dialog

3. Configure the communication settings for the Remote Agent:

3.1 Click on **Setup**. The "Setup" dialog is displayed:

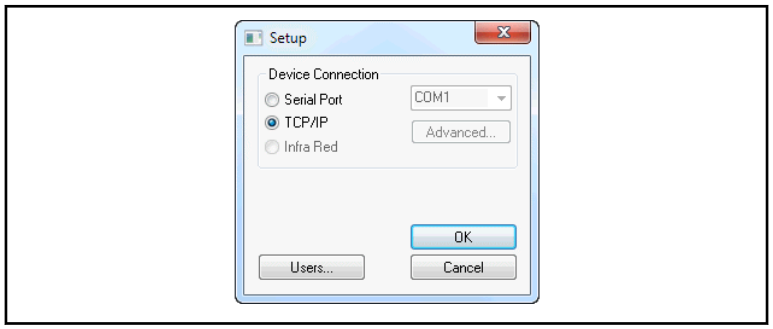


Fig. 3-2: Setup dialog

3.2 Select the appropriate device connection: **Serial port** or **TCP/IP**. If **Serial port** is selected, also select the COM port and check the advanced settings. If a connection to the target device has already been established, no other connection has to be selected at the moment. It has to be considered, however, how the target device is actually used during project runtime.

3.3 Click on **OK**.

3.4 If TCP/IP is selected as the device connection, note down the IP address.

3.5 Ensure that the Remote Agent is still running on the target device.

4. Establish a connection to the target device using the Remote management tool:

4.1 Start the development application on your computer.

4.2 Right-click on the **WinStudio** folder and click **WinStudio target platform...**.

The "Remote management" dialog is displayed.

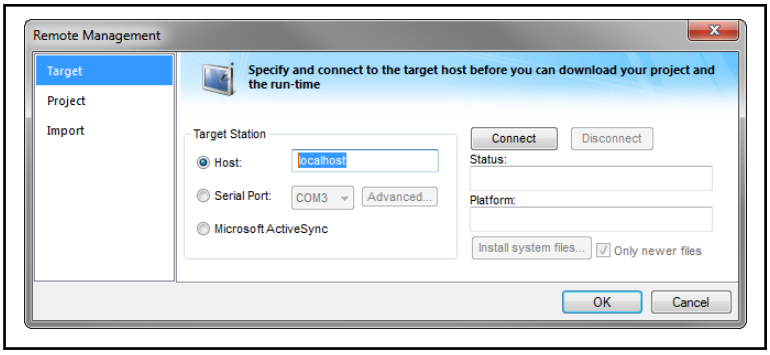


Fig. 3-3: "Remote management" dialog

4.3 Select the appropriate connection type for the target device: **Network IP**, **Serial Port** or **Microsoft ActiveSync**. If **Network IP** is selected, also enter the IP address of the target station. If **Serial port** is selected, also select the COM port and check the advanced settings for serial communication.

4.4 Click **Connect**. When the connection has been established successfully, the connection status is displayed in the Status field and the specifications of the target stations are shown in the Platform field.

If there is no connection, check the connection settings as well as the physical connections. In particular if **Network IP** was selected as the connection type, check if port 4322 is open in any firewalls between your computer and the target station or on the target station itself.

☞ In some cases, the remote management tool may not be able to connect via Microsoft ActiveSync to a device running Windows CE 6.0 or later. This is caused due to a problem in the default configuration of Windows CE 6.0. You can fix the problem by using a small utility that is included with Rexroth WinStudio. The utility is located at:

`C:\Program Files\Rexroth\WinStudio\Redist\ActiveSyncUnlock.exe`

Copy this file to the device using the stand-alone version of Microsoft ActiveSync and then execute the file on the device. It doesn't matter where on the device the file is located. When this is done, try again to use the remote management tool to connect to the device.

If it is still not possible to establish a connection via Microsoft ActiveSync, empty the `\Temp` directory of the device and try again.

---

5. Install the rest of the EmbeddedView or CEView software on the target device:
   - In the Remote management, click on Install system files.

   When the installation is completed, the updated status of the target device is displayed in the Status field.

If EmbeddedView or CEView is installed on the target device, you can now use it as a project runtime server or client.



The following runtime features are supported by the complete Rexroth WinStudio software but not by EmbeddedView and CE-View:

- DDE task worksheets.
- ODBC task worksheets and embedded functions. (The ADO interface is supported.)
- .NET Controls by third parties.
- OPC HDA server.
- OPC UA and OPC .NET clients
- Some minor graphic features such as fill effects for shapes other than rectangle, image formats other than BMP or JPG and the Ellipses style.
- Some embedded functions. For more information, please refer to the individual function descriptions.

Please keep that in mind when developing your project.

4 Licensing

4.1 License Protection Types

Rexroth WinStudio, EmbeddedView, and CEView support the following license protection types:

Hardkey

An encapsulated chip that has to be physically connected to the USB interface of the computer.

The software license is included in the hardkey and this license cannot be shared simultaneously with more than one other copy of software in the network. If the hardkey is connected to another computer, the license is effectively transferred to that computer.

While using the USB hardkey, the respective USB port cannot be used for any other device.

Softkey

When installing Rexroth WinStudio, EmbeddedView, or CEView, the program generates a unique code that is referred to as "Site Code". This Site Code can be sent to the software vendor, who then generates a license code, the "Site Key", to match the respective Site Code.

The Site Key is used to install the Rexroth WinStudio, EmbeddedView, or CEView license on the computer or Windows Embedded device.

4.2 License settings

Both hardkey and softkey licenses set the following parameters:

Version

The overall version on page 36 of the "WinStudio" software, e.g., "7.4". (This does "not" include the service pack, if available.)

Drivers

The number of direct communication drivers on page 413 that can be enabled simultaneously.

Product type

Specifies which features and restrictions are enabled for the application (e.g., the maximum number of tags supported).

Execution mode

Specifies one of the following options:

- **Engineering only**: Can develop a "WinStudio" project and then run it for short-term testing only. This license cannot be used as a long-term runtime license.
- **Runtime only**: Can run a "WinStudio" project for an unlimited period. This license cannot be used to develop or modify the project.
- **Engineering + Runtime**: Can develop a "WinStudio" project and then run it for an unlimited period.

For more information, refer to Execution modes, page 34

Thin clients

The number of thin clients on page 532 that can connect simultaneously to the server. Each license enables the connection of a single thin client. If more than one client are to be used, a special license is required.

Secure viewers

The number of secure viewer clients on page 532 that can connect simultaneously to the server. Each license enables the connection of a single secure viewer client. If more than one client are to be used, a special license is required.

SMA clients

The number of <Studio Mobile Access Clients> (SMA clients) that can connect simultaneously to the server. Each license enables the connection of a single SMA client. If more than one client are to be used, a special license is required.

4.3 License types

"Rexroth WinStudio" and CEView support the following license types that vary according to the available functions:

License type	WinStudio	EmbeddedView / CEView
Demo mode	✓	✓
Engineering license	✓	
Runtime license	✓	✓
Engineering and runtime license	✓	
WebClient license	✓	
ViewerClient license	✓	

Demo mode

Allows to download projects to remote stations and to run projects for testing or demonstration purposes. Runtime tasks can be executed and debugging tools can be used (LogWin and database spy), but they shut down automatically after running for two hours continuously.

The demo mode can be restarted and run for another two hours, etc.

WinStudio screens, worksheets or project settings cannot be created or edited in demo mode.

Licensed for engineering only

Enables all engineering options for an unlimited period.

In this mode, the runtime tasks and debugging tools (database spy, output window and LogWin module) can be executed continuously for 72 hours. These tasks close again after this period. However, they can be restarted and executed any number of times for 72 hours. This license can only be used for development and testing.

Licensed for runtime only

The following licensing levels are available: 1,5K, 4 K, 16 K, 64 K or 512 K variables.

Allows to run all runtime tasks and debugging tools (database spy, output window and LogWin module) for an unlimited time. However, screens and/or worksheets can neither be created nor changed.

The menu options available at runtime only mode are the same as the options listed for the demo mode (see previous table).

☞ An additional 0.5k runtime license is available for VR21x/VH21 devices. In contrast to the CE Lite license, the complete WinStudio functional scope (Windows CE/WEC7) can be used with this license. To edit this project type, an engineering and runtime license of at least 1.5k is required.

Licensed for engineering and runtime

The following licensing levels are available: 1,5K, 4 K, 16 K, 64 K or 512 K variables.

Enables all engineering options, runtime tasks and debugging tools (database spy , output window and LogWin module) for an unlimited period of time.

☞ The remote management tool (**WinStudio target platform** in the context menu of the "WinStudio" folder) is always available, independent of the execution mode. Thus, files on remote stations can always be downloaded.

WebClients licensed
The following licensing levels are available: 1, 4, 8 or 32 WebClients. Allows to display and operate a WinStudio application running on a server using WebClients. When using WebClients, it is required that the WinStudio application has already been created and that the computer executing the WinStudio application has been configured as web server. For further information, refer to chapter 15 "Web thin client and secure viewer" on page 505

ViewerClients licensed
The following licensing levels are available: 1, 4, 8 or 32 ViewerClients. Allows to display and operate a WinStudio application running on a server using SecureViewer clients. When using SecureViewer clients, it is required that the WinStudio application has already been created and that the computer executing the WinStudio application has been configured as web server. For further information, refer to chapter 15 "Web thin client and secure viewer" on page 505

☞ WebClient licenses and ViewerClient licenses have to be activated on the computer executing the WinStudio application. No license is required on the client computers.

☞ WebClient licenses and ViewerClient licenses can only be used with a runtime license or an engineering and runtime license with 1.5k variables and higher.

To display the currently running execution mode, click on **Tools** ▸ **Options** ▸ **General** ▸ **Software licenses**. The dialog displays the status of the different licenses including the remaining time if you are currently in the demo mode (also refer to Options in the project settings on page 54).

4.4 Product versions

The version numbers of Rexroth WinStudio, EmbeddedView and CEView should be identical. It uses the syntax `X.Y+SPWW` (e.g. "Rexroth WinStudio v7.4" or "CEView v7.4"). The following applies:

- "X" refers to the **family version**. The family version changes only when major enhancements are added to the product technologies and concepts.
- "Y" refers to the **sub-version**: The sub-version changes when minor enhancements and/or new features are added to the product.
- "WW" refers to the **service pack**. The service pack version changes when add-on packages have to be installed to accomplish the following:
  - Upgrade files for the version previously installed
  - Fix bugs in the product (showstoppers and no-workarounds)
  - Provide minor improvements before releasing the next version of the product

  Each Service Pack release supersedes the previous service pack release. For example, SP2 includes all the contents of SP1 and all newly upgraded files, bug fixes and improvements. SP3 includes all the contents of SP2 and all new upgraded files, bug fixes, improvements and so on.

Rexroth WinStudio, EmbeddedView and CEView can execute projects built in previous versions of the product. However, older versions of Rexroth WinStudio, EmbeddedView and CEView cannot execute projects built or modified in newer versions of the product.

For example, version 6.1 projects cannot be executed using "WinStudio" version 6.0, however, version 6.0 projects can be executed with "WinStudio" version 6.1.

☞ Each license is issued for a specific "family" version and "sub-version" (X.Y) and the license is only valid for that version (including service packs).

The license is not valid for a newer "family" version or "sub-version" of the product. Therefore, if you install a new version of Rexroth WinStudio, EmbeddedView or CEView, then you have to upgrade your license to the new version being installed.

If only a service pack is installed, the license does not need to be upgraded.

4.5 WinStudio License Activation

The individual "WinStudio Runtime" and "WinStudio Engineering and Runtime" licenses are subject to additional costs and have to be activated by entering a license key.

The personal license key can be found on the license sheet.

To activate a license in IndraWorks Engineering, go to**Tools ▸ Options ▸ General Information ▸ Software Licenses** in the licensing dialog and select "WinStudio".

The activation dialog can be opened by clicking on "Licensing…" and the activation key can be entered.

After the license has been activated, it may be required to restart WinStudio Engineering or IndraWorks Operation.

Saving the license on a dongle

The activation key is only valid for one PC. To be able to use a license on several PCs, the license can be saved on a dongle. The license is valid as long as the dongle is plugged into an USB port of the computer.

Windows CE licenses (VEx) cannot be transferred to a dongle.

License activation on VEx devices

To enable a WinStudio license on a VEx device with a Windows CE operating system, open the licensing dialog on the VEx device in the Windows start menu via **Program ▸ Rexroth ▸ WinStudio ▸ License management** and enter the activation key. The licensing software is already installed on new VEx devices. On VEx devices without pre-installed licensing software, the IndraWorks visualization software has to be installed first before enabling the WinStudio license.



The IndraWorks license component for Windows CE is already installed on new devices.

If an already existing VEx device is to be updated with the new software, the runtime component has to be installed on the VEx device via the context menu of the VEx device in IndraWorks Engineering with the Install software function.

The link may not be available. In this case, start the licensing dialog via `\StorageCard\Rexroth\WinStudio\bin\IndraWorks.LicenseImport.CE.exe`.

The licensing component requires ".Net CompactFramework 2.0" so that the firmware may have to be updated.

4.6 Invalid Licenses

When opening a project with the WinStudio editor that has been created on another computer with a higher class WinStudio license, the elements in WinStudio Engineering requiring a license are not activated. This means that these elements can neither be created nor edited in this case.

When starting IndraWorks Operation, a message is displayed informing the user that the available WinStudio license is insufficient for the current project.

No WinStudio screens are displayed in case of an insufficient or missing license.



If the current project still is to be used, activate the demo license. Use the IndraWorks licensing dialog to activate the license (as described above).

5 Development environment

5.1 Development Environment, General Information

The "Rexroth WinStudio" development environment is integrated in IndraWorks Engineering. It includes the following components:

- Project Explorer: "WinStudio" folder with subfolders
- Different editors for WinStudio screens, math sheets, etc.
- Toolbars
- "WinStudio Screen Tools" tool window
- "WinStudio library" tool window
- "WinStudio LogWin" output window
- "WinStudio Cross Reference Search" output window
- "WinStudio Symbols" output window
- "WinStudio Database Spy" output window

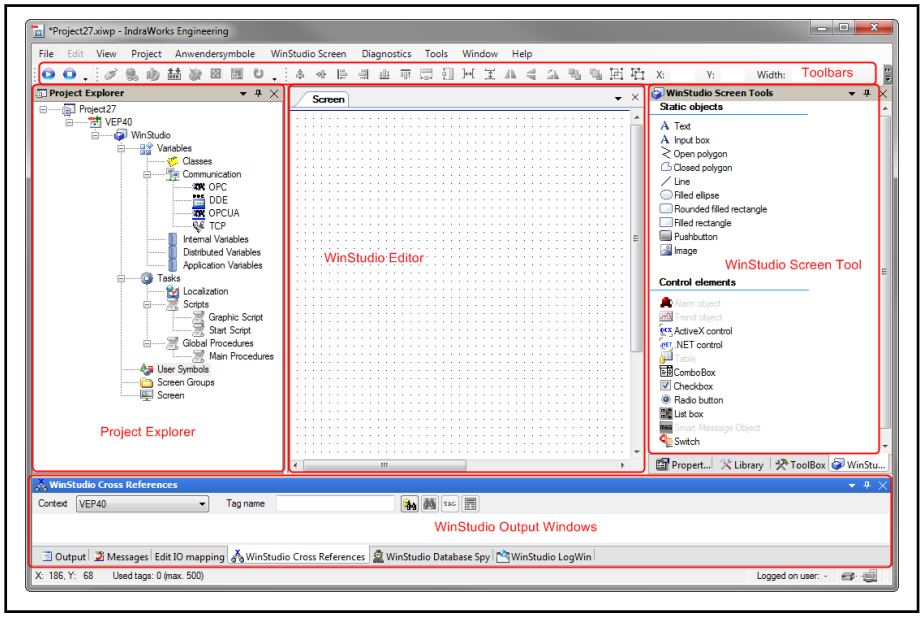


Fig. 5-1: The WinStudio development environment in IndraWorks

5.2 WinStudio toolbars

5.2.1 WinStudio Toolbars, Customizing

The creation of WinStudio applications is supported by several toolbars with numerous context-related functions. An overview of the available toolbars is provided in the IndraWorks main menu under **Tools** ▸ **Customize**. A configuration dialog is displayed to activate toolbars that are used frequently or hide toolbars that are used less frequently.

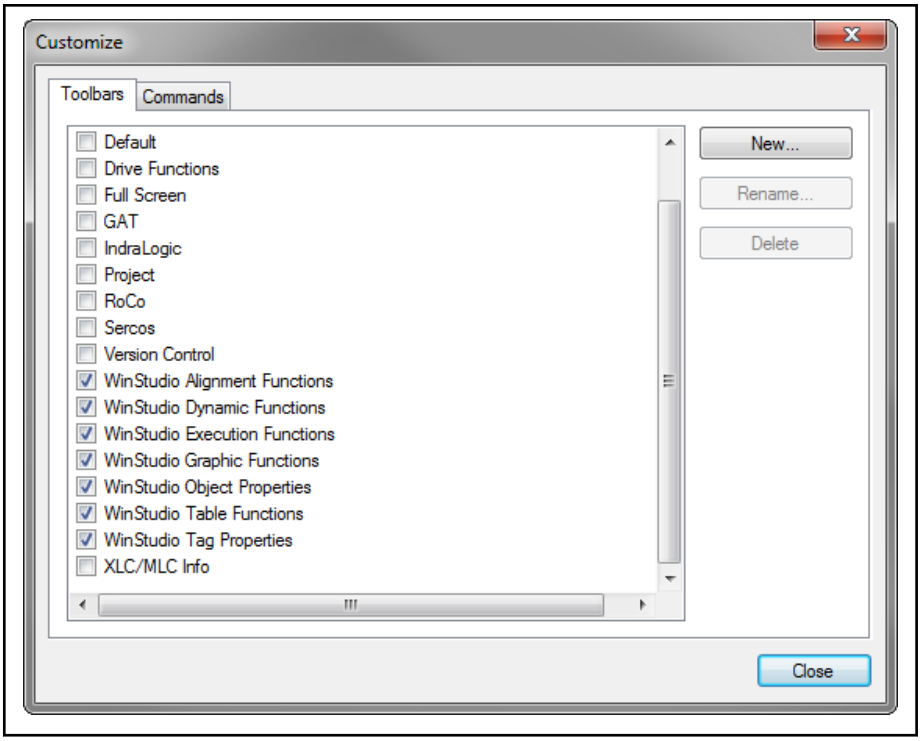


Fig. 5-2: Available WinStudio toolbars

Added toolbars are displayed at a predefined position. Toolbars can be arranged at any time by clicking on the left edge of the toolbar and keeping the mouse button pressed to draw it to the desired position.

The WinStudio functions offered in the individual toolbars depend on the context.

They can only be activated if the respective editor is opened and one or several objects are selected.

All functions are additionally available with opened WinStudio Editor in the IndraWorks main menu under menu item "WinStudio <element name>", e.g. "WinStudio Math Sheet" or "WinStudio Screen". This way, it is not necessary to show toolbars that are not frequently used.

In the following, please find a description of the individual toolbars.

5.2.2 WinStudio Execution Functions Toolbar

With the " WinStudio Execution Functions" toolbar, basic functions of WinStudio screens opened in the editor can be tested.



Fig. 5-3: WinStudio dynamic functions toolbar

Testing the screen

Activate the test mode of the currently edited WinStudio screen with the **Test screen** button.

In test mode, animations (e.g. dynamic positioning or resizing of objects) can be tested.

Screen objects can be repositioned and properties changed.

However, no tasks are started. Thus, no background scripts and functions can be tested in this test mode that require communication with external devices.

Stopping the test

With the , **Stop test** button, the test mode is deactivated and the normal editing mode activated.

5.2.3 WinStudio Alignment Functions Toolbar

The "WinStudio Alignment Functions" toolbar offers numerous functions to align, group and design screen objects.



Fig. 5-4: "Alignment Functions" toolbar

Some functions in the toolbar are only available if one or multiple objects have been selected in the WinStudio screen.

Move the cursor over an active "tool" to describe the function in the tooltip.

- **Center vertically** : Aligns all selected objects at the vertical center line of the object selected last
- **Center horizontally** : Aligns all selected objects at the horizontal center line of the object selected last
- **Left-aligned** : Aligns all selected objects at the left edge of the object selected last
- **Right-aligned** : Aligns all selected objects at the right edge of the object selected last
- **Align at bottom** : Aligns all selected objects at the bottom edge of the object selected last
- **Align at top** : Aligns all selected objects at the top edge of the object selected last
- **Same width for selected objects** : Increases/decreases the width of the object selected first to the width of the second one.
- **Same height for selected objects** : Increases/decreases the height of the object selected first to the height of the second one.
- **Evenly distribute horizontally** : Defines an equal horizontal spacing between a series of selected objects
- **Evenly distribute vertically** : Defines an equal vertical spacing between a series of selected objects
- **Mirror horizontally** : Rotates the selected object around an imaginary horizontal center line until it is a mirror image of the original object
- **Mirror vertically** : Rotates the selected object around an imaginary vertical center line until it is a mirror image of the original object
- **Rotate** : Rotates the selected object clockwise by 90 degrees

- **Move object to the background** : Moves one or several selected objects behind all other objects in the screen
- **Move object to the foreground** : Moves one or several selected objects in front of all other objects in the screen
- **Group objects** : Combines several selected objects to a single object
- **Ungroup** : Divides a grouped object into its individual components
- **Position X/Y:** Position of the highlighted object

Pixel-accurate positioning of screen objects: The position can also be changed by entering the x- and y-coordinates manually.

Use the keyboard to enter a valid numerical value either into the field "X" or the field "Y". To apply the new values, confirm them with the Enter key. The position of the screen object is immediately updated.

- **Width/Height**: Size of the highlighted object

Pixel-accurate size of screen objects: The size can also be changed by entering the width and height manually.

Use the keyboard to enter a valid numerical value either into the field "X" or the field "Y". To apply the new values, confirm them with the Enter key. The size of the screen object is immediately updated.

5.2.4 WinStudio Tag Properties Toolbar



Fig. 5-5: WinStudio tag properties toolbar

Edit the properties of the so-called WinStudio tags, the WinStudio variable objects, using the "WinStudio Tag Properties" toolbar.

- **Replace variable globally** : Enables a global exchange of a variable by an equivalent one
- **Change object properties** : Opens the object properties window of the selected object/object group

5.2.5 WinStudio Object Properties Toolbar

The "WinStudio Object Properties" toolbar offers functions to configure object properties.

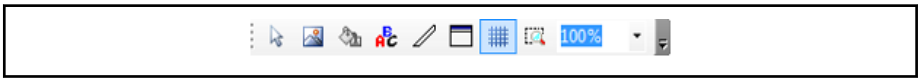


Fig. 5-6: "Object Properties" toolbar

Some functions in the toolbar are only available if one or multiple objects have been selected in the WinStudio screen.

Move the cursor over an active "tool" to display the function in plain text.

- **Select object** : Enables the tool for the object selection
- **Edit background screen** :
- **Fill color** : Opens the selection window for the default fill color and specifies it
- **Select font and text color** : Opens the selection window for the highlighted text/texts, in which fonts, style, text size, color and effects can be specified.
- **Line color** : Opens the selection window for line color of highlighted objects and defines the default color for new objects
- **Background color** : Opens the selection window for the background color of the WinStudio screen
- **Activate/deactivate grid** : Controls whether the grid in the screen editor is displayed or hidden
- **Select zoom range** 100% : Controls the zoom process for the section highlighted with the magnifier / for the entire WinStudio screen

5.2.6 WinStudio table functions toolbar



Fig. 5-7: WinStudio table functions toolbar

The "WinStudio table functions" toolbar provides the following buttons:

Create trend display : A trend display is created in the editor window

- **Create trend objects** : A trend display is created in the editor window (as an alternative to the “Trend object” control element)
- **Change security level** : Opens the security dialog for a WinStudio screen. Also refer to Security access levels on page 462
- **Find text** : Enables searching of texts in the active window
- **Replace** : Opens the dialog window to replace "Only variables", "variables and properties", "user properties". Also refer to chapter Renaming and removing tags on page 146
- **Add line** : Enables inserting lines into a worksheet
- **Select object** : Opens the dialog window to enter an object ID whose object is found in the active editor window

5.2.7 WinStudio Debug Functions Toolbar

The "WinStudio Debug Functions" toolbar supports testing and debugging of VB scripts. Scripts can be "debugged", i.e., execution can be stopped at a certain point, scripts can be executed in individual steps and values of variables can be observed.



Fig. 5-8: WinStudio Debug Functions toolbar

- **Activate/deactivate breakpoints in the Background Task**: Activate this option to debug scripts that are executed in the Background Task. These include:
  - Start script
  - Background scripts
- **Start debug mode**: Starts the WinStudio application in debug mode
- **Activate/deactivate breakpoints in the Viewer Task**: Activate this option to debug scripts that are executed in the Viewer Task. These include:
  - Graphics script
  - Screen scripts
  - Command scripts (e.g. on a button)
- **Set/remove breakpoint**: Inserts a breakpoint into the currently opened script at the current curser position or removes it

---

Breakpoints cannot be inserted at every position of scripts (e.g. blank lines, comment lines or lines with variable declarations). If the cursor is positioned in one of these lines and a breakpoint is set, it is automatically moved to the next admissible line.

- **Continue execution**: Leaves the single step mode and continues execution
- **Interrupt execution**: Interrupts the execution and switches to single step mode
- **Stepping**: Changes into the subfunction if the cursor is in single step mode on a line with function call
- **Next command**: Executes the next command
- **Return**: Returns from a subfunction to the superordinate function call
- **Stop debug mode**: Stops the debug mode

5.2.8 WinStudio Dynamic Functions Toolbar

With the "WinStudio Dynamic Functions" toolbar, properties of screen objects can by changed dynamically, i.e. animated.

To do so, select a screen object by clicking it and press one or several buttons of the toolbar. Press again to remove the dynamic property.



Fig. 5-9: WinStudio dynamic functions toolbar

- **Command**: Adds interfaces for commands that are to be executed by clicking on the screen object.
- **Hyperlink**: Adds the "Hyperlink" property. Afterwards, a link (URL) can be configured that is opened when clicking on the screen object.
- **Bar graph**: Adds the "Bar graph" property for filled rectangles, ellipses and polygons to allow the parameterization of a variable-controlled, continuous color gradient for the mentioned elements
- **Text input/output**: Inserts the "Text input/output" property to dynamically change the text displayed in a text object.
- **Color change**: Adds the "Color change" property to change the background color according to a variable value.
- **Position**: Adds the "Position" property to change the position of the screen object based on a variable value.
- **Resizing**: Adds the "Resizing" property to change the size of the screen object according to a variable value.
- **Line rotation**: Adds the "Line rotation" property to lines and polygons to rotate the object according to a variable value.

5.3 Project Explorer

5.3.1 Project Explorer, General Information

The "Project Explorer" organizes all of the screens, worksheets and other elements that make up the project and displays them in tree structure.

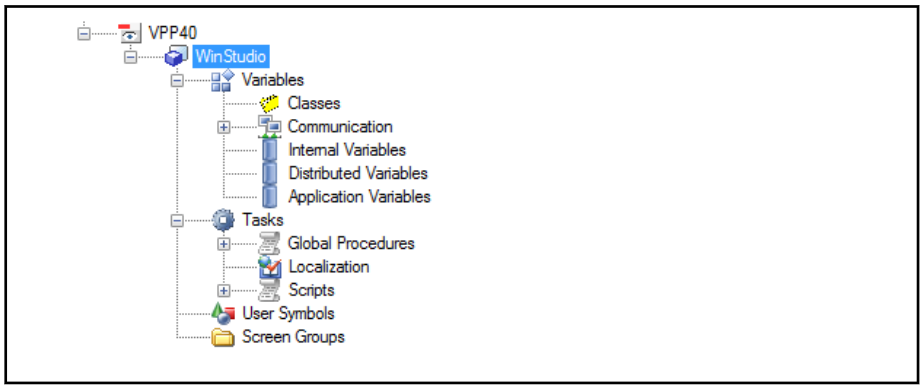


Fig. 5-10: The WinStudio folder structure

Click on the expand ▷ icon or double-click the folder to view the contents of the folder.

Click on the reduce ◢ icon or double-click the folder to close it.

All WinStudio elements in the project tree have a context menu. This menu can be opened by right-clicking on the element. The context menu contains

general functions as well as functions that are only available for the respective element type.

The individual sections of the WinStudio project tree include different WinStudio functionalities.

5.3.2 WinStudio folder

The folder **WinStudio** includes all screens, screen groups and symbols in the project.

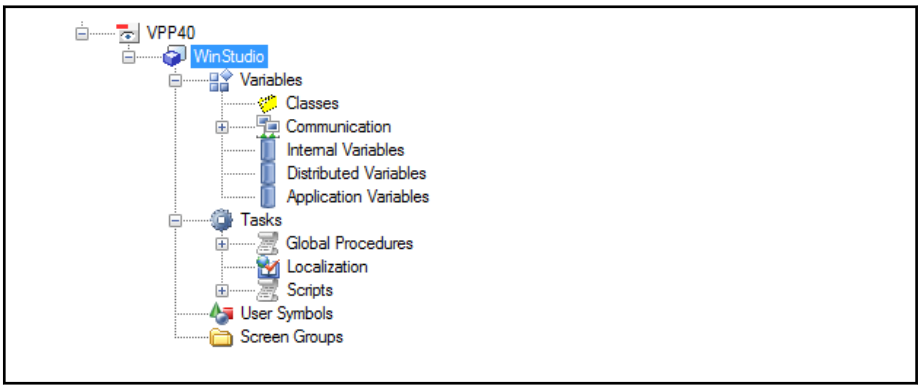


Fig. 5-11: The "WinStudio" folder

- **WinStudio** contains all screens on page 170 created for the current project.
- **Variables** contains all functions and editors for editing variable types as well as configuration of process integration of variables into external devices.
- **Tasks** contains different editors and functions required to create more complex applications ("math sheets", "alarms", "recipes","reports","trends","scheduler", database connections)
- **Screen groups** contains all screen groups on page 175 (individual WinStudio screens combined into manageable groups) created for the current project.
- **User symbols** contains all user-defined symbols on page 223, which can be groups of screens and/or text. Customized symbols for the project can be created and saved to this folder.
- **Graphics script** contains predefined functions that are executed when certain screen actions occur on page 930, such as when the thin client is started on a remote station.

5.3.3 Variables folder

The **Variables** folder contains all functions and editors required to manage WinStudio tags, to create user-defined data structures ("Classes") as well as for communication connection to a control system by means of one of the different available drivers.

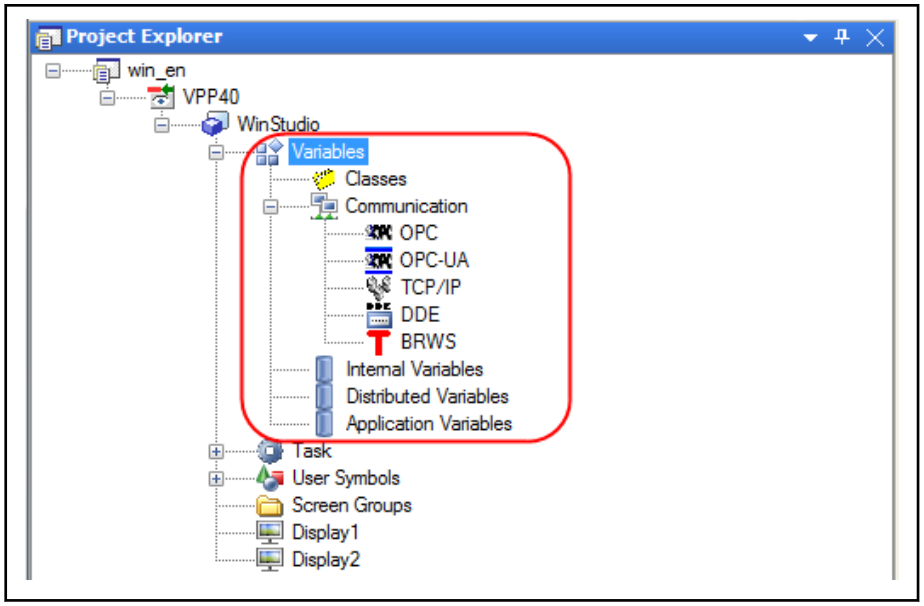


Fig. 5-12: "Variables" folder

This folder contains the following elements:

- Classes on page 106 contains compound tags, referred to as "classes" tags, created to link a set of values (rather than a single value) with an object
- System tags (internal variables) on page 108 contains predefined tags with particular functions that are used by the project for specific monitoring tasks (for example, "Date" tags contain the current date in string format). All internal tags are "read-only", which means they cannot be added, edited or removed from the database
- The Shared Database (distributed variables) on page 107 contains tags automatically created for process integration
- Project tags (application variables) on page 104 contains tags created during project development (such as screen tags or tags that read from/write to connected devices)

5.3.4 Tasks folder

The **Tasks** folder organizes the worksheets that are executed as background tasks during project runtime.

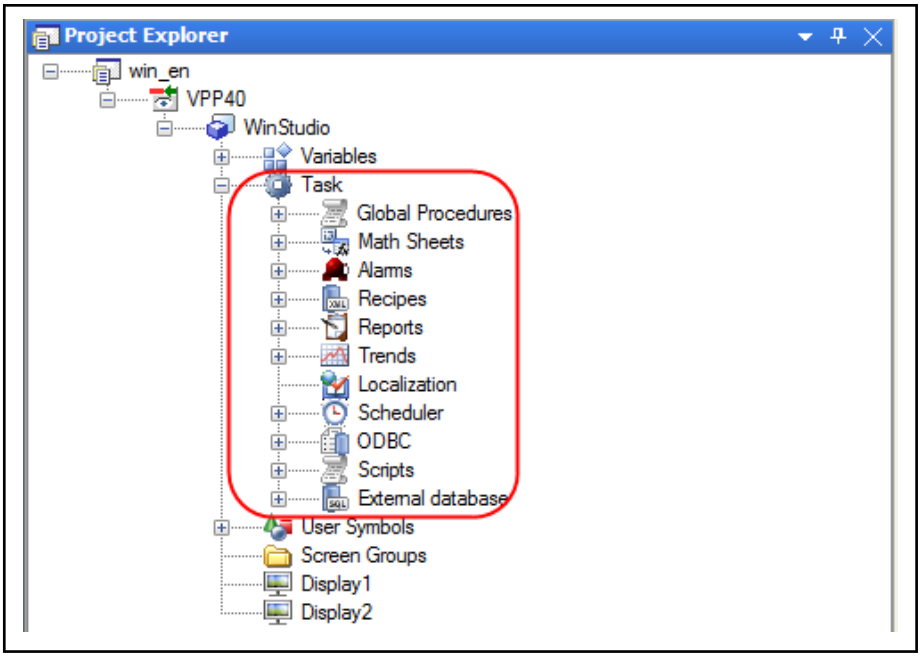


Fig. 5-13: "Tasks" folder

The folders on the “Tasks” tab are described on the following pages:

- **Global procedures** contains the **Main procedures** element by default. In this file, user-defined functions can be declared that can be called in a script. To structure these functions, add further scripts to declare additional functions

  Procedures on page 927 contains VBScript functions and subroutines that can be called by any other script in the project
- **Math sheets** contains the math worksheets on page 400 used to configure and create additional routines that work with different tasks. WinStudio executes "math" worksheets as background tasks during runtime. Configure "math" worksheets to provide sequences and mathematical calculations for the project
- **Alarms** contains Alarm worksheets on page 313 to specify alarm messages that are displayed during runtime if an assigned tag reaches or exceeds a certain value. The pending alarm messages at runtime can be logged automatically into a file and displayed in a WinStudio screen by means of the alarm control
- **Recipes** contains Recipe worksheets on page 395 to configure the variables the data of which is written to an ASCII or XML file or read from a file and applied to the project database at runtime
- **Reports** contains Report worksheets on page 396 to create text documents that are created at runtime and sent to a printer or file. Not only fixed texts but also current values of variables can be put out in reports. This function is particularly useful for documentation of system data
- **Trends** contains Trend worksheets on page 340. Trend worksheets facilitate the recording of chronological sequences of variable values. These sequences of values can be used to display them in a WinStudio screen with the trend control. Declare those variables in Trend worksheets that are to be saved to what is referred to as a binary history file (**host**)

- **Localization** opens the translation editor. Here, all texts that are used in the application can be translated to different languages. At runtime, the texts can be switched between various languages

  Translation on page 487 contains the translation worksheet that defines how the project's user interface is to be translated into other languages
- **Scheduler** contains Scheduler worksheets on page 403 to execute a function on various events. Events can be value changes of variables, reaching a value or defined times or dates
- **ODBC** contains ODBC worksheets on page 397 to configure ODBC interfaces in a network environment. ODBC worksheets are used to exchange data between projects and any database that supports the ODBC interface
- **Scripts** contains the Startup script on page 402 as well as the **Graphics script**. Script groups on page 401. These predefined scripts can be used to execute functions on start of the application or during execution of the Viewer. Additionally, user-defined scripts can be created in this folder to program sequences in VB Script
- **External database** contains Database worksheets on page 404 that communicate with external databases via the standard ADO.NET interface (as an alternative to ODBC)

5.3.5 Communication folder

Create worksheets in the subfolders of the **Communication** folder to establish the communication with other devices or software by means of available protocols.

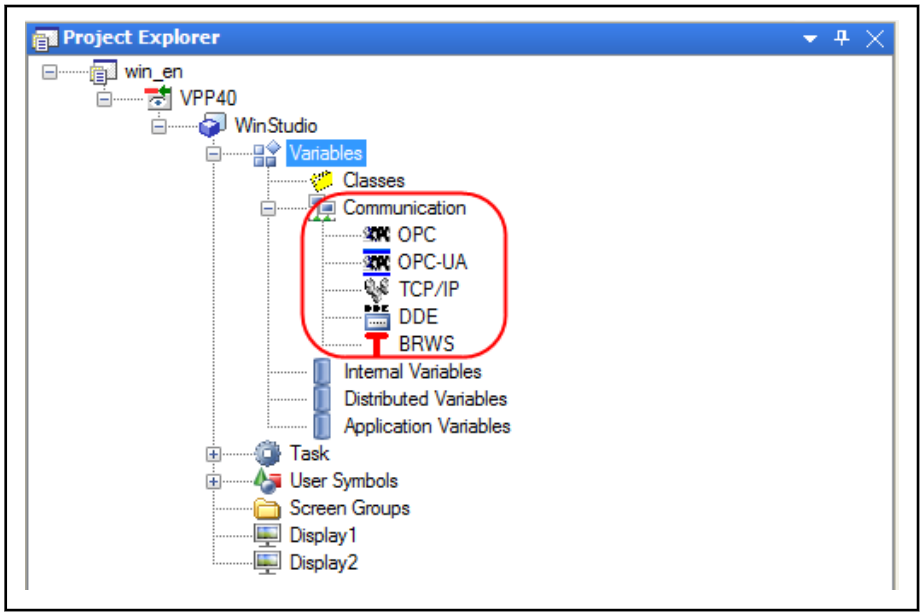


Fig. 5-14: "Communication" folder

The “Communication” folder contains the following subfolders and functions.

- **OPC** contains OPC worksheets on page 427 to configure communications between a project (which acts as an OPC Client) and any other device that implements the classic OPC Standard (OPC-DA)
- **OPC UA** contains OPC UA worksheets on page 433 that are used to establish a connection to OPC Servers via the new "OPC Unified Architecture" protocol

- **TCP/IP** contains TCP/IP worksheets on page 453 to configure TCP/IP communication to other "WinStudio" stations

  By means of "WinStudio" TCP/IP client and server modules, databases of two or more projects can be synchronized
- **DDE** contains DDE worksheets on page 455 to enable WinStudio to exchange data via DDE protocol (Dynamic Data Exchange) with a Windows application (e.g. Microsoft Excel or similar Windows program supporting this interface). DDE communication is an interaction between server and client programs. "WinStudio" provides interfaces that run as clients or as servers
- **BR_WS** contains Driver worksheets on page 413 to enable WinStudio to exchange data with a PLC control via the BR_WS communication interface. A "communication driver" is a `.DLL` file that contains specific information on the remote equipment and implements the communication protocol
- **Additional drivers:** If data is to be exchanged with a device that does not support any of the protocols described above, respective additional drivers can be added. If a driver file (`.DLL`) is available for the device, copy it to the `..\WinStudio\Drv` folder. Subsequently, open the driver configuration dialog in the context menu of the "Communication" folder via the menu **Add/Remove driver**. All available drivers are listed under this item. After adding a driver, the respective subfolder is created in which the associated worksheets can be created

5.4 Configuring and testing an application

5.4.1 Verifying an application

Use the **Verify project** function to check your project (**WinStudio ▸ Verify project**).

The function checks Math worksheets on page 400, the screen logic, WinStudio tags, BRWS and OPC driver sheets and updates HTML files. The latter has to be configured in the Web tab of the Project settings dialog on page 71.

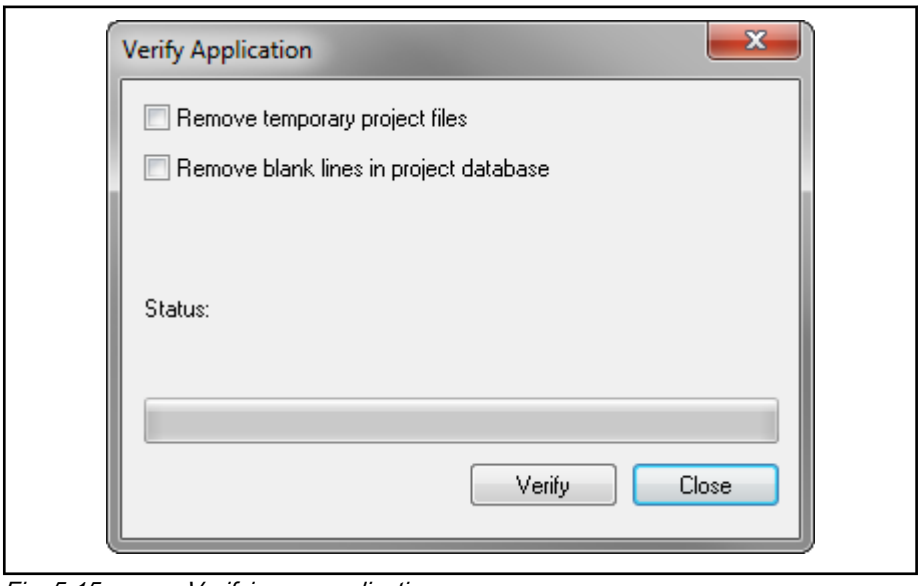


Fig. 5-15: Verifying an application

If variables are for example used in the application that are not defined in the database, error messages are generated in the message window of the output area. The position of the error opens when clicking on the message in the editor. Additionally, the function detects unused variables. These can be removed by the project by clicking on the message.



When saving screens or worksheets, WinStudio references to the current database version. When executing a project, WinStudio compares the database version of the runtime with the current project database and recompiles the expressions if they do not match.

To avoid this compilation during the project runtime, it is recommended to execute the **Verify Application** function before downloading a project to the target device.

This function is also to be used when a project was converted to a new WinStudio version.

Select **Verify application** to delete all temporary files (e.g., `*.txt`, `*.mac` and `*.tag` files) from the project folder and to clean the project before loading it to the runtime environment.

However, WinStudio does "not" automatically upload the project to a target device, but has to use the function Send project to target on page 549 to transfer the project to the runtime environment of the target device.

5.4.2 Starting the application

Use the **Start application** function (**WinStudio ▸ Start application**) to execute the runtime modules which are marked as **Automatic** in the Execution tasks on page 91 dialog ("Execution tasks tab").

Start the "Viewer" module to open the screen(s) currently being edited.

If no "automatic" tasks are specified, "WinStudio" automatically starts the "Viewer" and "Background" tasks when clicking on **Start application**.

If no screens are opened in the editor, the "Viewer" module opens the configured start screen (refer to the **Start screen** field on the "Viewer" tab in the Project Settings on page 53 dialog).

Start application refers to the project on the target system (configured in the Remote management on page 52 dialog).



Before clicking **Start application**, make sure that the correct target system is configured (local or remote).

5.4.3 Stopping the application

Use the **Close application** function (**WinStudio ▸ Close application**) to close all runtime tasks.

Close application refers to the project of the selected target system (refer to the Remote management on page 52 dialog).



Before clicking **Close application**, make sure that the correct target system is configured (local or remote).

5.4.4 Debug application

The development application provides additional tools for troubleshooting in your VBScript code (**WinStudio ▸ Debug application**)

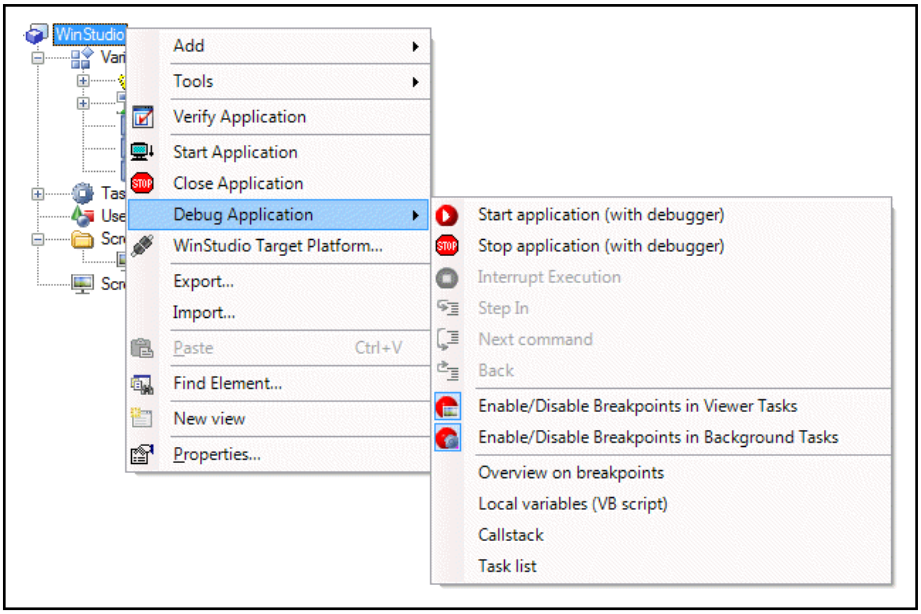


Fig. 5-16: Debugging tools

See also

- Troubleshooting VBScript on page 966
- WinStudio Debug Functions Toolbar on page 44

5.4.5 WinStudio target platform

Select this function to open the "Remote management" dialog to manage remote stations: Uploading and downloading files and executing/stopping remote projects (**WinStudio ▸ WinStudio target platform**).

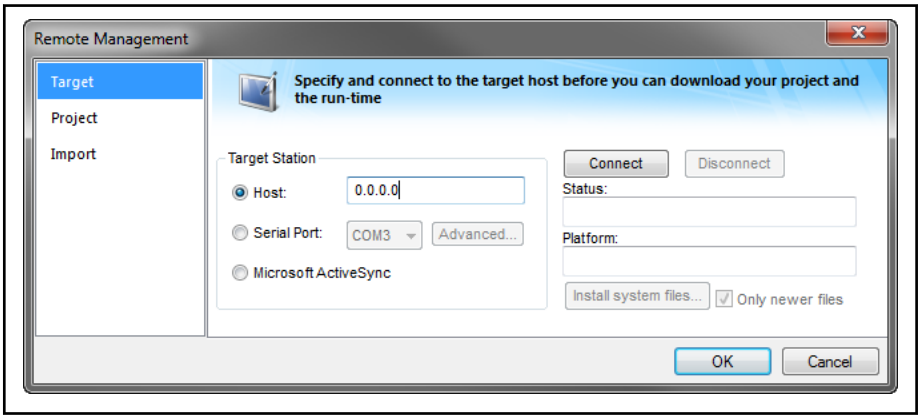


Fig. 5-17: "Remote management" dialog

This dialog contains the following tabs:

- **Target**: Use this tab to specify the target system, including:
  - **Network IP**, **Serial port**, or **Microsoft ActiveSync** (Microsoft ActiveSync currently only works with Windows Embedded Compact and Windows Mobile 5.0 and later.)
  - **Connect**/ **Disconnect** the target system
  - Review station **Status**

  - Returns the **Platform** (operating system + processor type) of the target system after connecting to it
  - **Install system files** (or **Only newer files**)
- **Project**: Use this tab to specify the project, including:
  - **Local** and **Target** directory paths
  - Specify whether to **Send** (the project) **to target system**
    - **Only newer files**: If this option is enabled, only newer application files are sent to the device
    - **Retain user files**: If this option is enabled, any user files in the project folder are retained
  - **Send file**
  - **Run/Stop** the project
  - Review project **status**
- **Import**: Use this tab to import a project, this includes:
  - Specify the **target** and **source** project path
  - **Get from target**
  - **Get log file**
  - Review project **status**

For further information on connecting to Windows Embedded devices, refer to Installing EmbeddedView or CEView on a target device on page 27.

5.4.6 Project settings

Information Tab

Select the entry **Properties** in the context menu of the "WinStudio" folder to open the "Project Settings" dialog (**WinStudio ▸ Properties**).

It controls settings concerning the overall project.

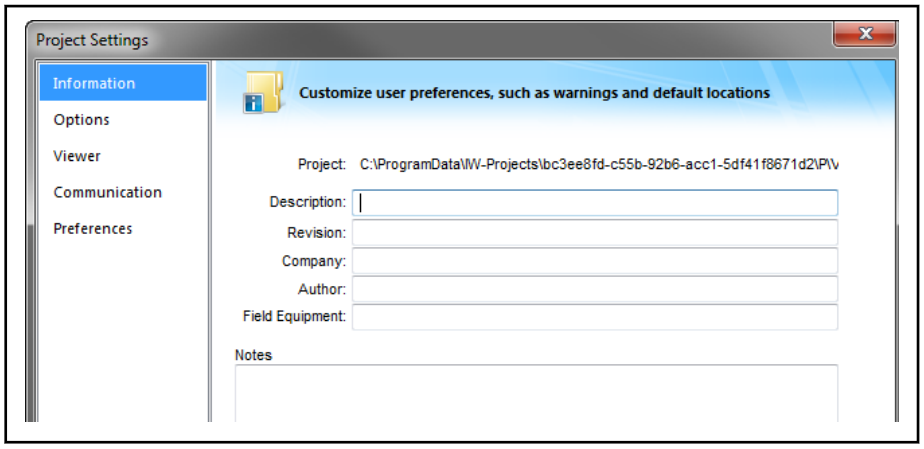


Fig. 5-18: Project Settings: Information tab

Identify the project with the following parameters (for documentation purposes only).

- **Description**
- **Revision**
- **Company**
- **Author**
- **Field Equipment**

- Notes

Options tab

Options tab, general information

Use this tab to specify parameters relating to the project in general.

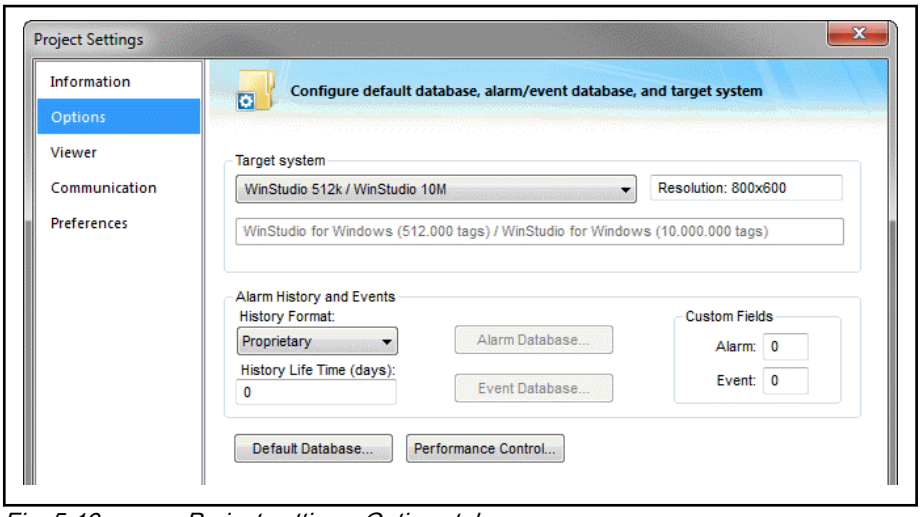


Fig. 5-19: Project settings: Options tab

These parameters are described in the following:

- **Target system**: Use the selection list to specify the target system for the current project. The target system sets the project restrictions (such as number of tags supported) and has to match the active license. The description of the main license restrictions for each target system is displayed below the selection list where it is specified.

  NOTICE **If a Target system level is specified that does not match the current license level on the target system, the project may not run properly.**

- **Resolution**: Displays the screen resolution of the project.
- **Alarm history and events**: Enter a value into the "History life time (days)" field to specify how long alarm and event history files are retained. After the specified number of days, the project automatically deletes existing alarm/event history files that are older than the period specified. Enter zero into the field to configure that the project does not delete any history files automatically. In this case, create an external procedure to delete the old history files. Otherwise, the free disk space in the computer decreases.
- **History format**: Select the format of the alarm/event history as follows:

Format	Description
Proprietary	Saves the history data in the "Alarm" subfolder of the project folder (by default) in text files using the proprietary format
Database	Saves the history data in the user-defined SQL relational database using the built-in ADO interface
Binary	Saves the history data in the "Alarm" subfolder of the project folder (by default) in binary files using the proprietary format

For more information, refer to Saving the alarm history/event log to an external database on page 322.

- **Default database**: Use this item to configure a default database that can be used by different tasks and objects. For further information, refer to Configuring a default database for all task histories on page 567.

The PC-based control has a customized interface that requires information on the PC-based control application in order to share tags with the "WinStudio" project.

Saving the alarm history to an external database

By default, the alarm history and event log are saved to proprietary-format text files in the “Alarms” folder of the project. However, the project settings can be modified to save them to an external SQL database instead.



If your project was created with Rexroth WinStudio v7.3+SP2 or earlier and then upgraded to the current version, you should consider starting over with new database tables.

Beginning with Rexroth WinStudio v7.3+SP3, new database tables are automatically indexed by event time in order to improve runtime performance. Existing database tables cannot be indexed in this way, so if you can afford to discard that data, you should change your database configuration to create new tables.

If you do this, you must also manually edit your project file (projectname.APP) to add the following setting:

```
[Alarm]
AddEventTimeColumn=1
```

The setting exists for backward compatibility, and it defaults to 0 for projects that were upgraded from earlier versions.

1. Open the **Options** tab in the "Project settings" dialog.

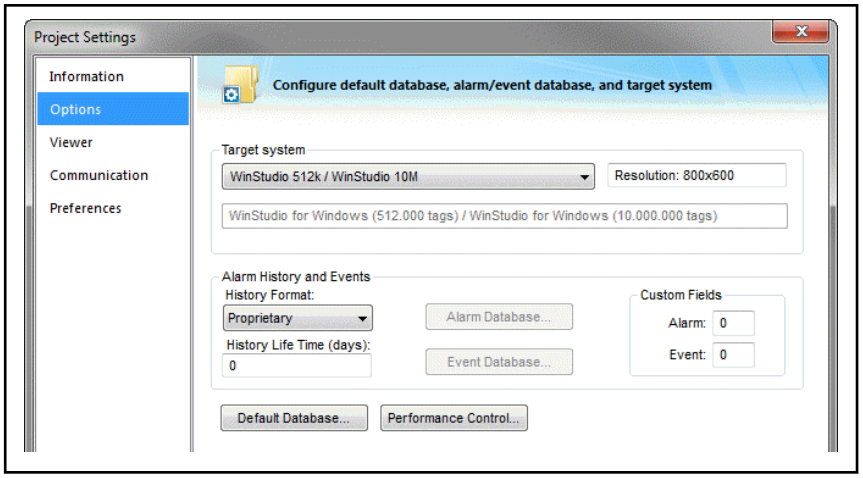


Fig. 5-20: Project settings: Options

2. In the **Alarm history and events** pane, in the **History life time (days)** input field, enter the number of days of history that are to be saved. As the history exceeds the specified number of days, it is automatically deleted in a first-in, first-out manner. If no number is specified - that is, if it is left blank or set to 0, - the history is never deleted. There is no limit on how much history can be saved, however, the more is saved, the more disk space it takes.

3. From the **History format** list, select **Database**.
4. To configure a single, default database to be used for both the alarm history and the event log (as well as all other runtime tasks), click **Configure** in the **Default database** pane. The "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For further information, refer to Configuring a default database for all task histories on page 567.
5. To configure a separate database, either for the event log or the alarm history, click **Event database** or **Alarm database**. In either case, a "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For more information, refer to Default database configuration on page 563.
6. Click on **OK**.

Configuring a default database for all task histories

A default database can be configured to save the historical data of all tasks in a project. After configuring a default database, either use the default database or configure a new database for this special worksheet when creating a new task worksheet.

Configuration of the connection settings for the default database:

1. Open the properties dialog of the "WinStudio" folder and select the "Options" tab
2. In the "Default database" pane, click **Configure**. The "Default database configuration" dialog is displayed

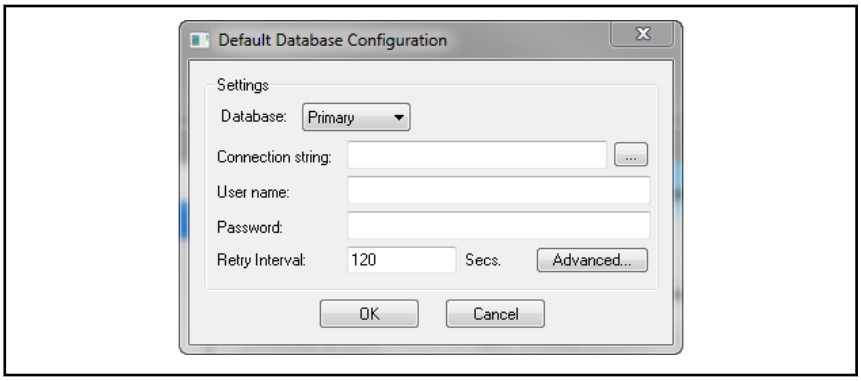


Fig. 5-21: "Default database configuration" dialog

Please refer to Default database configuration dialog on page 563 for help on the completion the fields in this window.

Database configuration

Database connection configuration

The data exchange with an external data base file is carried out analogously to the device drivers. First, configure a new data base connection. Afterward, worksheets can be created in which the created data base connection can be selected and WinStudio tags can be linked to individual data sets of the external database.

To configure a new data base connection, open the context menu of the **WinStudio** ▸ **Tasks** ▸ **External database** folder and select the "Default database configuration" entry.



The "External database" folder is only available if "WinStudio 1.5k" or higher is selected as target system in the WinStudio properties and if the device version in the VPx/Vex properties is "13V06" or higher.

Open the configuration dialog for a new database connection using the "Add" button in the overview dialog. Enter a name in the "Name" field that can later be selected in the worksheets.

The “Database configuration” dialog allows to configure the necessary settings to link WinStudio to an external database file.

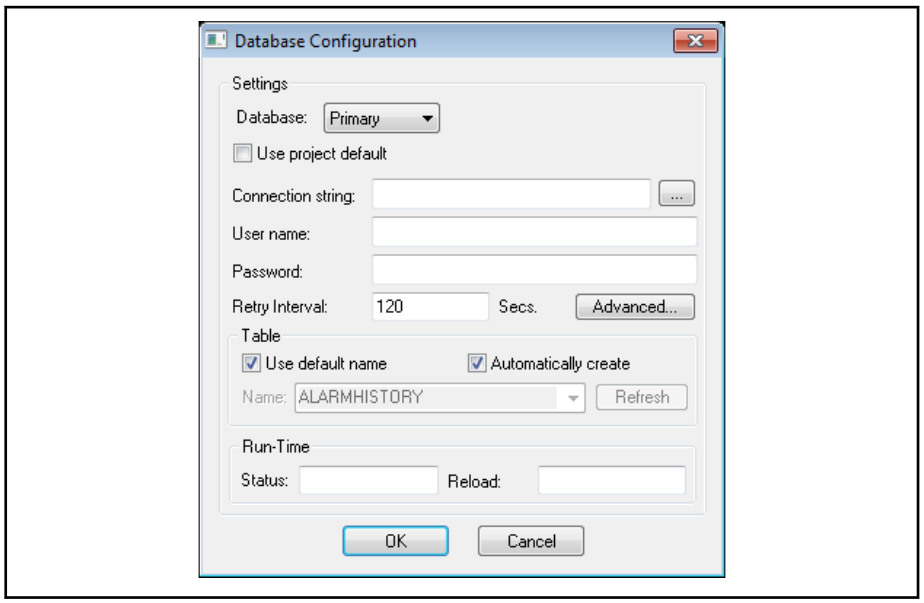


Fig. 5-22: “Default database configuration” dialog

- **Database** combo box: To select either “Primary” or “Secondary”. If "Primary" is selected, all settings of the default database configuration apply to the primary database interface. Otherwise, they apply to the secondary database interface. The secondary database can be configured in the following modes:
  - **Disabled**: In this mode, "WinStudio" saves data in the primary database only. If the primary database is unavailable for any reason, the data is not saved anywhere else. This option may cause loss of data if the primary database is not available.
  - **Redundant**: In this mode, "WinStudio" saves data in both primary and secondary databases. If one of these databases is unavailable, "WinStudio" keeps saving data only in the database that is available. If the database that was unavailable becomes available again, "WinStudio" synchronizes both databases automatically.
  - **Store and forward**: In this mode, "WinStudio" saves data in the primary database only. If the primary database becomes unavailable, "WinStudio" saves the data in the secondary database. If the primary database becomes available again, "WinStudio" moves the data from the secondary database into the primary database.



The primary and secondary databases can be different types of databases. However, they have to have the same fields.

Use the secondary database to increase the reliability of the system and to use the secondary database as a backup when the primary database is not available. This architecture is particularly useful when the primary database is located in the remote station. In this case, a secondary database can be configured in the local station in such a way that data is saved temporarily if the primary database is not available (during a network failure, for instance).

- **Use project default**: If this option is checked, "WinStudio" uses the settings in the default database for the configured task (connection string, user name, password, retry interval and advanced). If this option is disabled, these settings can be configured individually for the current task.
- **Connection string**: This field defines the database where "WinStudio" writes and reads out values as well as the main parameters used when connecting to the database. Instead of entering the connection string manually, the “Browse” button (…) can be clicked to select the database type from the **Data link properties** window.

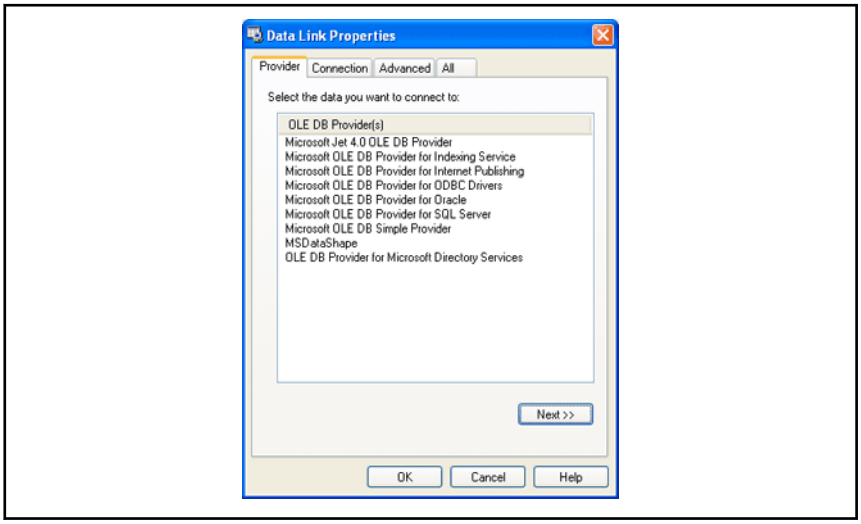


Fig. 5-23: “Data link properties” dialog

The list of database providers shown in the “Data link properties” window depends on the providers actually installed and available on the computer where "WinStudio" is running. Consult the operating system documentation (or the database documentation) for further information on the settings of the database provider for the used database.

- **User name**: User name used to connect to the database. The user name entered in this field has to match the user name configured in the database.
- **Password**: Password used to connect to the database. The password configured in this field has to match the password configured in the database.
- **Retry interval**: If "WinStudio" is unable to connect to the database for any reason, it automatically retries to connect to the database after the seconds configured in this field have passed.
- **Advanced**: Pressing this button provides access to the configuration of advanced settings. For most projects, the default value of these settings does not need to be modified and is to be retained.

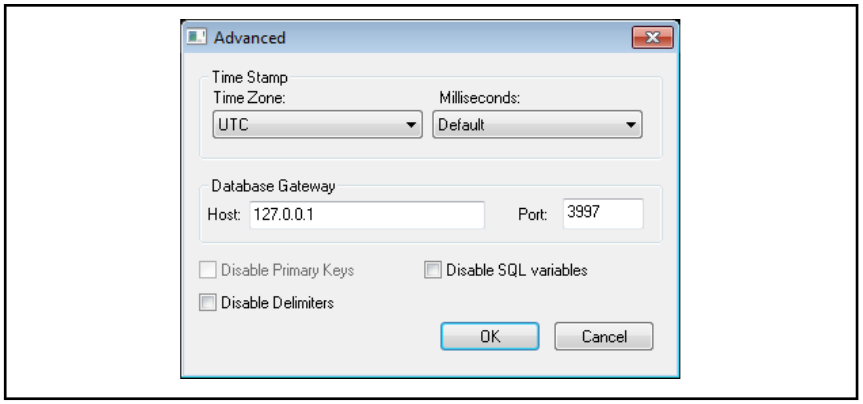


Fig. 5-24: Database configuration: “Advanced” dialog

– **Time zone**
  – **Local time + time difference**: Save the local time on your computer plus the difference (offset) between the local timezone and the Coordinated Universal Time.
  – **Local time**: Only save the local time without offset. This is not recommended.
  – **UTC**: Only save the UTC time. This is the default setting and is highly recommended in most cases.
– **Milliseconds**

  Use this item to set how the milliseconds are saved when saving the date in the database. Different databases save the date in different formats. For example, some databases do not support milliseconds in the **Date field**. The following options are available:
  – **Standard**: Uses the pre-defined format for the current database. The databases that were already tested by "Bosch Rexroth" are preconfigured with the most suitable option. When selecting "Default", "WinStudio" uses the preconfigured setting for the current database type. If a database is used that has not been preconfigured, it is tried to save the milliseconds in a separate field if the “Default” option is selected.



The default option for each database is pre-configured in the `StADOSvr.ini` file, stored in the `\BIN` sub-directory of "WinStudio". For further information on configuration of the `StADOSvr.ini` file, refer to Studio Database Gateway on page 556.

  – **Disable**: Does not save the milliseconds at all when saving the date in the database.
  – **Enable**: Saves the milliseconds in the same field where the date is saved.
  – **Separate column**: Saves the milliseconds in a separate column. In this case, the date is saved in one field (without milliseconds) and the number of milliseconds is saved in a different column. This option is displayed if time stamps are to be saved with milliseconds precision and the used database does not support milliseconds for **date fields**.
– **Database gateway**: Enter the host name/IP address of the network node where the "WinStudio" Database Gateway is running. The

TCP port number can also be specified, however, if the default is not used, the "WinStudio" database gateway has to be configured with the same TCP port. For further information on configuration of the advanced settings of the "WinStudio" ADO gateway, refer to Studio database gateway on page 556.

- **Disable primary keys**: For some modules, "WinStudio" tries to define a primary key to the table in order to speed up the queries. Enable this option if a database is used that does not support primary keys (e.g. Microsoft Excel).
- **Disable delimiters**: Select this error handling option to disable the delimiters that are used to format communications with the database. Delimiters can cause problems when a trend control or grid creates a query that includes aggregates such as min. and max.
- **Disable SQL variables**: Select this error handling option to disable SQL variables, such as "@Value1" and "?", that are often used in SQL commands and requests. Some specific database providers do not support these variables.



If this option is selected, the language or culture to be used to format the values to SQL statements may have to be specified. For further information, refer to “Advanced settings” in the Studio database gateway, page 556.

Table pane

This pane enables configuration of the settings of the table where the data is saved. All tasks can share the same database. However, each task (Alarm, Events and Trend worksheets) has to be linked to its own table. "WinStudio" does not check this field for invalid configuration. Thus, ensure that the respective setting is suitable for the used database.

- **Use default name**: If this option is checked (default), "WinStudio" saves and/or loads the data in the table with the default name written to the **Name** field.
- **Create automatically**: If this option is enabled (default), "WinStudio" automatically creates a table with the name displayed in the **Name** field. If this option is disabled, "WinStudio" does not automatically create the table. Thus, data can only be saved in the database if the database already contains a table with the name that had been preconfigured manually in the field **Name**.
- **Name**: Specifies the name of the table from the database where the history data are saved. To specify a worksheet in a Microsoft Excel spreadsheet file, use the following syntax:

  ```
  [worksheetname$]
  ```

- **Refresh**: If the set database is currently available, press the **Refresh** button to fill the **Name** combo box with the name of the tables currently available in the database. This way, it is possible to select the table where the history data is to be saved instead of manually entering the table name in the **Name** field.

Runtime pane

In this pane, the runtime values can be defined. The following fields are available:

- **Status** (output): The tag in this field receives one of the following values:

Value	Description
0	Disconnected from the database. The database is not available, its configuration is incorrect or an incorrect operation is executed
1	The database is successfully connected
2	The database is being synchronized

- **Reload** (output): Specify a "reload" tag when using curly brackets in the configuration field. To reconnect to the database using the updated values of the tags, set the tag in this field to 1. "WinStudio" updates the configuration when trying to perform an action in the database, setting the tag back to 0 when it is finished.

For further information, refer to:

Configuring a default database for all task histories on page 567.

Viewer tab

Viewer tab, General information

Use this tab to configure the project viewer and to change certain behaviors at runtime.

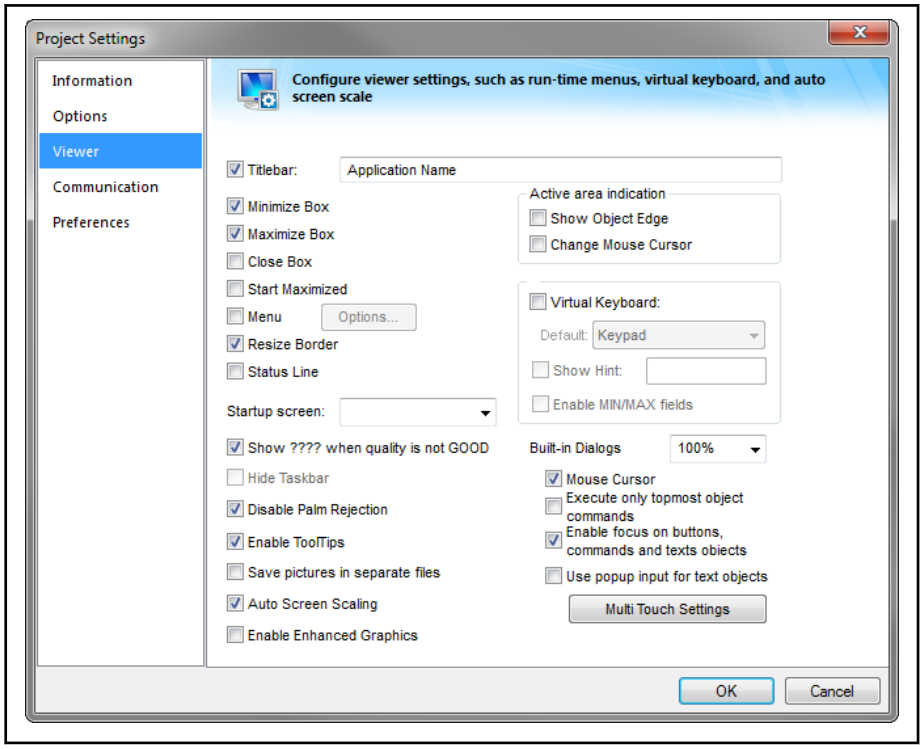


Fig. 5-25: Project settings: Runtime Desktop tab

- **Title bar**: Click (enable) this box and enter a new name into the field provided to specify or change the default title bar text for the Viewer window
- **Minimize box**, **Maximize box**, and **Close box**: Click these checkboxes to display (enable) or hide (disable) these buttons in the Viewer window
- **Start Maximized**: Click (enable) this checkbox to maximize the Viewer window automatically when running the project

- **Menu** and **Options…** button: Click (enable) the checkbox and the **Options…** button to specify which menu options are available at runtime. If the "Runtime menu options" dialog is displayed (as follows), click the checkboxes to display (enable) or hide (disable) the respective menu options.

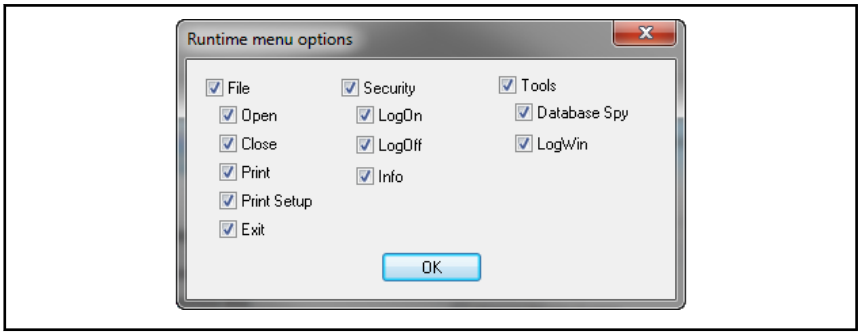


Fig. 5-26: Runtime menu options dialog

- **Resize border**: Click (enable) this checkbox to be able to resize the Viewer window at runtime
- **Status line**: Click (enable) this checkbox to display the status line in the runtime project
- **Startup screen**: Click on the selection list and select a WinStudio screen or a screen group that is to be displayed automatically on opening the runtime project

---

☞ Another way to specify a screen or screen group as startup screen is to right-click on the element in the "Project Explorer" element and to choose **Set as startup screen**.

- **Show ???? when quality is not GOOD**: Click (enable) this box to display question marks (???) instead of the tag value if the quality is not good
- **Hide taskbar**: Click (enable) this box to hide the Windows taskbar by default
- **Disable Palm Rejection**: Select this option to disable “Palm rejection” at project runtime. “Palm Rejection” is a feature of Windows 7 touchscreen devices that detects and rejects accidental touches from the operator's palm. This may, however, considerably reduce the responsiveness of the touchscreen. For this reason, disabling this function can improve the performance of the project runtime object.

☞ “Palm Rejection” is only available on Windows 7 touchscreen devices, so selecting **Disable palm rejection** has no effect on projects running on other target systems.

Windows 7 touchscreen devices also have an option that makes it possible to press and hold on the screen to perform a right click ("press and hold"). This function is useful on final consumer devices like tablets and smartphones, however, it is possibly not useful when working on "WinStudio" projects as it is necessary to press and hold buttons and other screen objects. This option is enabled by default. To deactivate this option on the device, proceed as follows:

1. Click on the Start button and afterwards on Control Panel in the Start menu. The Control Panel window is displayed.
2. Double-click on “Stylus and Touch”. The “Stylus and touch” dialog is displayed.
3. Click the Touch tab.
4. From the Touch actions list, select "Press and hold" and click Settings. The “Press and hold settings” dialog is displayed.
5. Disable "Enable press and hold for right click".
6. Click OK to close the “Press and hold settings” dialog.
7. Click OK to close the "Stylus and touch" dialog.

- **Enable ToolTips**: Click (enable) this checkbox to see Windows tooltips when running the project. Tooltips can be configured in the **Hint** field of the "Object properties" dialog of each object.
- Check box **Save pictures in separate files**: If this option is selected, graphics which are inserted in a screen are automatically converted to graphics links and saved as separate BMP graphics files in your project folder under `projectname\Web\Resources`. It is then possible to reuse the graphics without increasing the size of your project since each instance of the graphic is linked to the same file. This should improve the runtime performance.

  If this option is not selected, graphics which are inserted in a screen remain Bitmap objects and are embedded in the screen file. Each instance of a graphic is embedded. If the same graphic is used more than once, this may considerably increase the size of the screen file.

The effect of this option is permanent: If an inserted graphic is converted into a graphics link, it does not become a Bitmap object again if the option is deactivated.

The Resources subfolder is automatically created by the development application and is only used for Bitmap graphics which are inserted and converted. Do not save any other graphics files in this folder since they may be moved or deleted. Save graphics to be embedded in the Web subfolder instead.

- **Auto Screen scaling**: Click (enable) this box to automatically scale project screens when you resize the Viewer window. This function is available for local projects running on Windows PC or in the thin client. This parameter is not available for projects running on Windows Embedded target systems.
- Check box **Enable enhanced graphics**: If this option is selected, your project screens have enhanced graphics functions such as anti-aliasing and opacity.

Anti-aliasing smooths the lines, edges, captions and gradient fills of shapes (i.e., line objects et al.), text objects and button objects (standard and OS styles). It is not supported for conical gradient fills.

Using the opacity (0% to 100%), a screen object can be made transparent so that other objects behind it become visible. The objects can then be arranged to create specific visual effects. To set the opacity of an objects, use the Visibility/Position animation on page 255.

Activating the enhanced graphics functions may reduce the runtime performance of some clients. Check this thoroughly before implementing your project. In addition, enhanced graphics functions are not supported on Windows Embedded devices. The target system of your project has to be a Windows computer.

- **Virtual keyboard**: If this option is enabled, the virtual keyboard on page 66 is enabled for the project. (This option is "not" available if the project is running on a Thin Client.

  For more information, refer to Project settings: Web on page 71.) With this keyboard it is possible to enter data during runtime on touch screens without entering data on a physical keyboard. For example, a text object on page 192 the Text data link animation on page 251 was applied to and the **input enabled** option was enabled. A default configuration for the virtual keyboard can be set:

  – **Default**: Select the default keyboard type to be used in the project if no keyboard type is specified by the calling object or function.

  – **Scale:** This option enables shrinking or enlarging the keyboard to fit the size of the target screen. A value of 100% represents the default size of each keyboard type.

  – **Show hint**: If this option is enabled, a hint is displayed in the title bar of the keyboard. For individual objects, the hint is configured in the Object properties dialog on page 178. Otherwise, enter a string value into the "Show hint" field to serve as a default hint.

  – **Enable MIN/MAX fields**: If this option is enabled, the permissible minimum and maximum values are displayed at the bottom of the keyboard. For objects, these values are configured in the Object properties dialog on page 178. For other cases, the min. and max. properties on page 122 of the associated tag are used by default.

☞ The **Min** and **Max** fields are displayed only on the keypad keyboard type, and only if the associated tag is defined as Integer or Real. If the min. value exceeds the max, value, the input is disabled. If the min./max. values configured on the object differ from the min./max. values configured in the tag properties, the project attempts to convert the input accordingly.

- **Built-in dialogs** combo box: Select the type of scaling for the display of built-in dialogs (i.e. logon, E-sign and virtual keyboard) during runtime. In this way, the dialogs are easier to see and use on project runtime clients with small displays with high resolution. Enter a scaling between 100% (native resolution) and 400% or select `Auto` which means that the client automatically choses the best scaling for the display.
- **Mouse cursor**: Click (enable) this checkbox to display the mouse cursor in the runtime project.

The mouse cursor option is not supported in Windows CE running on Armv4I processors.

- **Execute only topmost object commands**: This option controls the project behavior if it is clicked in a pane where two or more screen objects overlap. If this option is enabled, only the commands of the topmost object are executed. If this option is disabled, the commands of all overlapping objects are executed.

The topmost object is the one with the highest ID number. (The ID number of an object is displayed in the Status bar on page 84 at the bottom of the development environment.) The Move to back / move to front on page 269 and Move backward / move forward on page 269 tools can be used to change the order in which objects are stacked.

- **Enable focus on buttons, commands and text objects**: This option is enabled by default. If this option is enabled, clicking on a command button or text input during runtime puts the focus on that object, as shown in the illustration below:

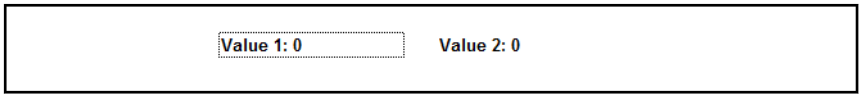


Fig. 5-27: Focus on text input (left), no focus on text input (right)

After that, the **Tab** key can be used to navigate to all focusable objects in the screen or the **Enter** key can be used to activate the currently focused object (i.e., clicking on a command button, entering text in a text field), similar to a standard Windows application. This is useful if the client workstation is equipped with a physical keyboard and if many objects are to be processed quickly.

To force the end-user to always use the mouse or touchscreen to activate screen objects, delete this option.

- **Use popup input for text objects**: Select this option to display a small popup for text inputs, as shown in the following illustration:

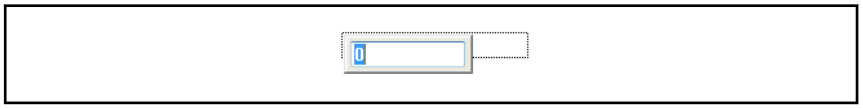


Fig. 5-28: Popup for text input

This is an alternative to typing directly into the text object and to displaying a virtual keyboard for input (which requires using a mouse or touchscreen).

If the **virtual keyboard** option (above) is selected it overrides this option.

if the **Enable focus on buttons, commands and text objects** option (above) is "disabled", this option is automatically enabled. This is to ensure that there is an indication in the screen of which text input is currently activated.

- **Multi touch settings**: Configuring the standard multi-touch settings for all screens in your project. For further information, refer to Multi-touch settings for project screens on page 286.

Keyboard

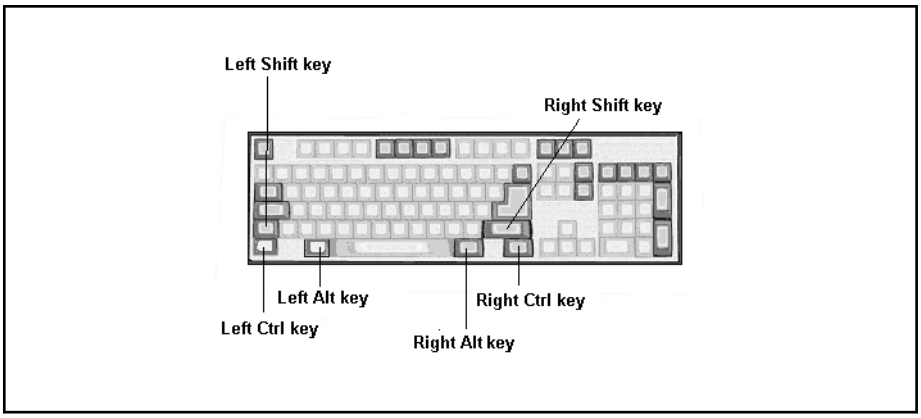


Fig. 5-29: Keyboard

Virtual keyboard

The virtual keyboard (VK) enables entering data (text or numbers) during runtime using a touchscreen device instead of a physical keyboard.

The virtual keyboard can be enabled for projects on the Runtime Desktop dialog on page 61. A default keyboard type as well as a scaling (adjustment factor) can be specified for all virtual keyboards displayed at runtime.

When configuring objects and animations that support data input, a virtual keyboard can be assigned to the object by the virtual keyboard combo box (either the default or a specific one). The following default virtual keyboards are available:

Name	Description	
Alphanumeric	This virtual keyboard is used to enter alphanumeric values (characters and/or numbers).	
Enhanced...	This virtual keyboard is used to enter alphanumeric values (characters and/or numbers) on devices that are only equipped with a small display (e.g. PDAs). The "->" button enables scrolling though different panel labelings to find the required character	
Keypad	This virtual keyboard is used to enter numeric values (numbers only)	

There are two ways to start a virtual keyboard during runtime:

- Clicking on the object where a new value (new data) can be entered (e.g. the text with text data link animation configured with released input).
- Executing the build-in function Keypad() on page 700.

The language of the virtual keyboard can be changed by the built-in SetKeyboardLanguage() on page 802 function.

- New languages can be created for any virtual keyboard by editing the `VKlanguage.INI` file from the `\BIN` subfolder of "WinStudio", where **language** is the description of the language for the VK (e.g. `VKEN.INI` is virtual keyboards in English, `VKGE.INI` is virtual keyboards in German, etc.).

  For further information on editing and configuring this file, contact the software vendor.
- By default, the VK is started close to the object it is connected with. However, the following settings in the project file can be configured (`project_name.app`) so that the virtual keyboard is always opened at a fixed position:

  ```
  [Keypad] PosX=0 //Coordinate (in pixels) of the TOP coordinate where the virtual keyboard has to be started.
  PosY=0 //Coordinate (in pixels) of the LEFT coordinate where the virtual keyboard has to be started.
  ```

Communication tab

This tab is used to specify communication parameters relating to the project in general.

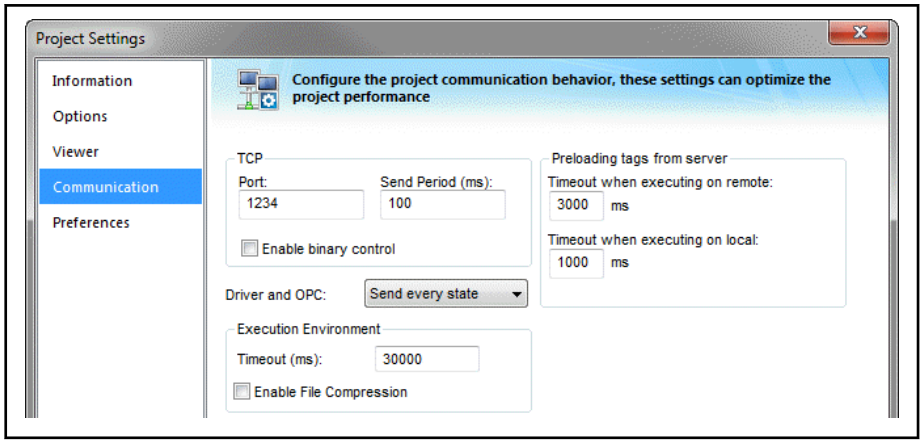


Fig. 5-30: Project settings: Communication tab

- **TCP** area: In this pane, configure the communication settings for the TCP/IP client and server modules on page 453:
  - **Port**: TCP Port used by the TCP/IP Client and TCP/IP Server modules running on the current processor. When changing this value in the local project, be sure to change the same value in the remote project that is communicating with the local one.

    This setting is required for all projects and cannot be left blank. Make sure that the network configuration allows connections to this port on the station that is to be the project server.



If the port is changed to a different number than 1234, Mobile Access and Mobile Access Tabular are disabled.

  - **Send period (ms)**: Period (in milliseconds) between two consecutive messages sent by the TCP/IP client or TCP/IP server modules to update tag values in the remote station. Typically, a lower value means better performance and higher traffic in the network.
  - **Enable binary control**: Select this option to enable binary control when the TCP/IP client or the TCP/IP server module sends messages to the remote station. Binary control increases the safety of the system; however, it decreases the efficiency (performance) of the communication. When connecting two stations via the TCP/IP client and TCP/IP server module, make sure this setting is either enabled or disabled in both projects.

- **Driver and OPC**: Select the method used by all communication drivers on page 413 and OPC Client worksheets on page 427 configured in the current project when writing values to the remote PLC/device:
  - **Send every state**: When the communication task is configured to write values upon a change of a tag value, all changes in the tag value are buffered in a queue and sent to the device when the communication task (driver or OPC) is executed.



There is a size limit for the buffer regarding tag value changes to prevent accumulated changes from reducing the runtime performance. If the buffer size is exceeded, only the most recent changes are retained until the task is executed again and the changes are sent. In addition, a warning message is saved in the dump file.

To adjust the buffer size, manually change the following setting in your project file (`projectname.APP`):

```
[Options]
DriverAndOpcBufferSize=5
```

  - **Send last state**: When the communication task is configured to write values upon a change of tag value, only the current (last) value of the tag is sent to the device when the communication task (driver or OPC) is executed. When this method is selected and the tag value changed more than once while the communication task was executed, the transiently valid values of the tag are not sent to the device. This is the desired behavior for most projects.

- **Preloading tags from server** pane: To improve performance, the runtime viewer loads all of the server tags of a screen to memory before it displays that screen. Configure the timeout settings for both remote and local viewers:
  - **Timeout when executing on remote** field: Specifies the time (in milliseconds) that a Secure Viewer or Thin Client running on a remote station waits to load the tags.
  - **Timeout when executing on local** field: Specifies the time (in milliseconds) that the viewer running on the local station (i.e. the server) waits to load the tags.
- **Execution environment** pane: Use this item to configure the communication settings for the remote management operation on page 52 to send the project files to a target system:

– **Timeout (ms)**: Specifies the time (in milliseconds) that the project waits to communicate with the target system.
– **Enable file compression**: Select this option to compress the system and project files before sending them to the target system. This may reduce the download time if there is a slow connection between server and target system. (If there is a fast connection, however, then selecting this option may "decrease" performance because each compressed file has to be decompressed on the target system before the next file is sent. Select this option only if there is an extremely slow connection, such as dial-up connection.) Data compression is disabled by default.

Preferences tab

Use this tab to configure personal settings.

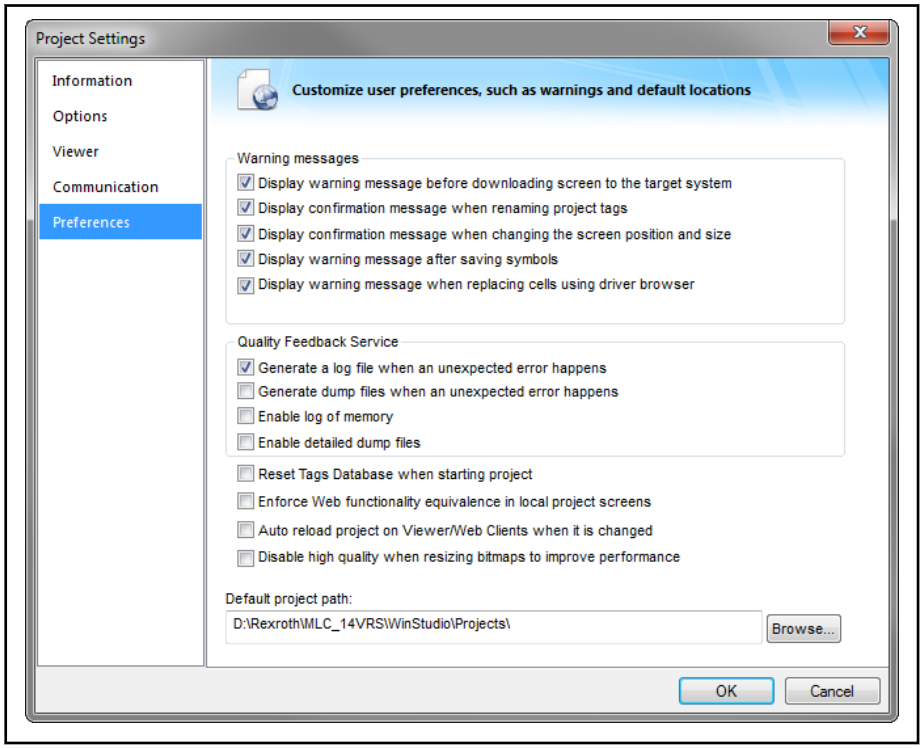


Fig. 5-31: Project Settings: Preferences tab

Warning messages

- **Display warning message before downloading screen to the target system**: If this option is selected and "WinStudio" is connected to a remote station (Remote Management on page 52 dialog), a prompt is displayed directly after saving a screen in the screen editor to download the updated screen to the remote station. If this option is disabled, the screen is downloaded automatically, regardless of any confirmation.
- **Display confirmation message when renaming project tags**: If this option is selected and the name of any tag is modified in the Project Tags on page 104 worksheet, a prompt is displayed to replace the old tag name in the entire project by the new tag name. If this option is enabled, "WinStudio" executes the "global replace" command to replace the old tag name with the new tag name in all documents of the project (screens and worksheets).
- **Display confirmation message when changing the screen position and size**: If this option is selected, you are prompted to update the screen at-

tributes (width, height, top and left) after modifying them on the Layout on page 177 interface.

- **Display warning message after saving symbols**: If this option is enabled, a warning message is shown after saving symbols on page 223.

Quality feedback service

This pane enables configuration of projects to generate log files and/or dump files that can be used to diagnose hardware and software problems, such as memory leaks and unexpected errors. These files are saved in the Dump subfolder of the current project (`[…]\Documents\WinStudio v7.4 Projects\projectname\Web\Dump`).

- **Generate a log file when an unexpected error happens**: If this option is enabled, the runtime modules append the log file (`\Web\Dump \Dump.txt`) whenever an internal exception (error) occurs. These exceptions may not necessarily crash the runtime modules, but they can affect the stability of the system and are to be investigated.

☞ The log file is continually appended until it reaches its maximum size of 2MB. After it reaches its maximum size, the existing file is deleted and a new file is created.

- **Generate dump files when an unexpected error happens**: If this option is enabled, the runtime modules generate a new dump file (`*.dmp`) with useful information on the conditions of the error. This is a binary file that can only be read by the software vendor.

☞ Dump files are named `WinXXX.dmp` - where "XXX" is an identifying number (in hexadecimal format) automatically generated by the system - in order to prevent an existing file from being overwritten when a new error occurs. Therefore, if more than one error occurs, multiple dump files are saved in the directory. The log file indicates the name of the dump file associated with each error.

- **Enable log of memory**: If this option is enabled, runtime modules append the log file every 15 minutes with information on the current memory allocation. (The first log entry is written 15 minutes after the runtime module is started). This information can be used to identify memory leaks.
- **Enable detailed dump files**: If this option is selected, the dump files contain more detailed information than usual. Please note that these files require much more space on the hard disk. Only select this option when looking for a specific error with your support representative.

Even if none of these quality feedback options are enabled, a post-mortem dump file (`\Web\Dump\WinDump.dmp`) is always generated to analyze the crash when the runtime module is terminated by a fatal error. However, for debugging purposes, it is highly recommend to enable all options in this section and send the log file and all dump files to the software vendor.

Other preferences

- **Reset tags database when starting project**: If this option is selected, the project tags are reset automatically to the initial values whenever the project is run. For further information, refer to Reset tags database on page 149.
- **Enforce web functionality equivalence in local project screens**: If this option is selected, the development environment generates a warning if

you try to use functions which are not supported by the project runtime server or the Thin Clients (i.e., Secure Viewer and Web Thin Client). This ensures that your project always behaves in the same way, no matter where it is displayed. If this option is selected, for example, the Open on page 666 function cannot be called from the Global procedures on page 927 or the Script worksheets on page 401 since the function is only effective if it is executed on stations which display project screens.



This option is disabled by default in order to maintain compatibility with previous versions of Rexroth WinStudio. If the compatibility does not have to be maintained, select this option.

- **Auto reload project on Viewer/web clients when it is changed**: If this option is enabled, remote stations (i.e. thin clients and Secure Viewers) check the server to see if they have the most recent version of the project. If this is not the case, the new version is downloaded automatically from the server.
- **Disable high quality when resizing bitmaps to improve performance** option: If this option is selected, the size of Bitmaps in project screens is changed if the quality is poor. Usually, Bitmaps need to be read in again if they are not displayed in their actual size to maintain high quality. If this reading process needs to be performed often or for many Bitmaps on the same screen, this may reduce the runtime performance.
- **Default project path** field: The default memory location of new projects you created. If the Rexroth WinStudio software is installed, it is automatically set to `C:\Users\username\Documents\WinStudio v7.4 Projects\`. It can, however, be changed to any location on your computer or in your network.



If the Cooperation function was selected during the installation of the Rexroth WinStudio software, it is not possible to change the set default project path even if you are not actually connected to a Collaboration Server. To change the location where your projects are saved, your workspace needs to be re-assigned in the Collaboration Server settings.

For further information, refer to Cooperation.

Web tab

For the “Web” tab, go to **WinStudio** ▸ **Tools** ▸ **Web Settings**.

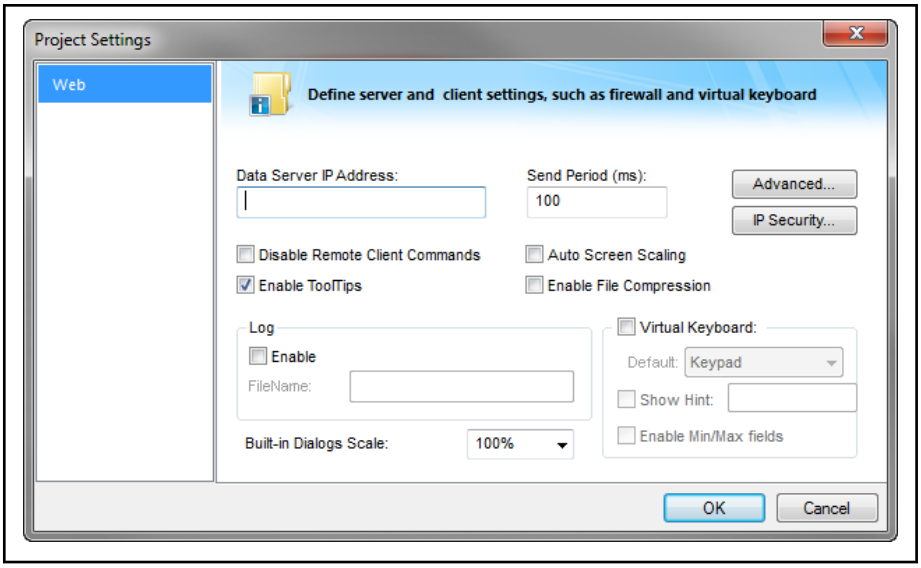


Fig. 5-32: Project settings: Web tab

If any settings are changed, check the project before sending it back to the target device (Verifying an application on page 50).

The parameters on this tab are configured as follows:

- **Data Server IP address**: Enter the IP address (or host name) of the data server station here. The data server station is the computer or device on which the TCP/IP server module of "WinStudio" is running. If this field is left blank, the Thin Client assumes that the web server (i.e. the address entered into the browser) is also the data server.

The IP address "127.0.0.1" (localhost) can be used to access the TCP/IP server on the local computer, regardless of its actual address in the network. This option is useful for local tests; however, the data server cannot be accessed from remote computers with this configuration.

- **Send period (ms)**: Enter a value to specify the interval (in milliseconds) used to exchange data between the server and the thin client stations. It means that the thin client sends a package with the new tag values to the server every "n" millisecond(s). The **Send period** of the server is configured in the project settings (**Communication** tab in the project properties). The default value is "1000" (milliseconds). A lower value can be set in this field to increase the update rate between server and thin clients. This may result in higher data traffic in the network (the network is accessed more frequently) if the tags are changing continuously (faster than once per second).
- **Disable remote client commands**: Enabled this checkbox to prevent that remote clients commands are sent from thin clients to the server. If this option is enabled, the thin client can read data from the server but cannot send data (tag values, command values) to the data server. In this case, the thin client station becomes a "read only" station.
- **Enable ToolTips**: Enable this checkbox to display the "ToolTips" configured for the screen objects when displaying them on a thin client (web browser).

- **Auto Screen scaling**: Enable this checkbox, to automatically adjust the size of the screens displayed in a web browser window. Using this option, the screen adjusts to the size of the web browser window, regardless of its resolution.

☞ The **Auto screen scaling** option is not valid for web browsers running under Windows CE operating systems.

- **Enable file compression**: Enable this checkbox to compress the files stored in the "Web" subfolder of the project folder. This option is useful for reducing load time, particularly in case of a slow connection between server and Thin Client.
- **Log** (**Enable** and **FileName**): Enable this checkbox and enter a file name into the text field to generate a log file on the Thin Client station. This log file can be used for debugging.
- **Virtual keyboard**: If this option is checked, the virtual keyboard on page 66 is enabled for projects running with Thin Clients. With this keyboard it is possible to enter data during runtime on touch screens without entering data on a physical keyboard. For example, a text object on page 192 the Text data link animation on page 251 was applied to and the **input enabled** option was enabled.

  A default configuration for the virtual keyboard can be set:

  – **Default**: Select the default keyboard type to be used in the project if no keyboard type is specified by the calling object or function.

  – **Scale**: This option enables shrinking or enlarging the keyboard to fit the size of the target screen. A value of 100% represents the default size of each keyboard type.

  – **Show hint**: If this option is enabled, a hint is displayed in the title bar of the keyboard. For objects, the hint is configured in the Object properties dialog on page 178. Otherwise, enter a string or string tag in the **Show hint** field as a default hint.

  – **Enable min/max fields**: If this option is enabled, the permissible minimum and maximum values are displayed at the bottom of the keyboard. For objects, these values are configured in the Object properties dialog on page 178. For other cases, the min. and max. properties on page 150 of the associated tag are used by default.

☞ The **Min** and **Max** fields are displayed only on the **Keypad** keyboard type, and only if the associated tag is defined as Integer or Real. If the min. value exceeds the max, value, the input is disabled. If Min/Max configured on the object is different from min./max. configured in the tag properties, the project attempts to convert the input accordingly.

IP security settings

IP security: Click this button to open the “IP security” dialog.

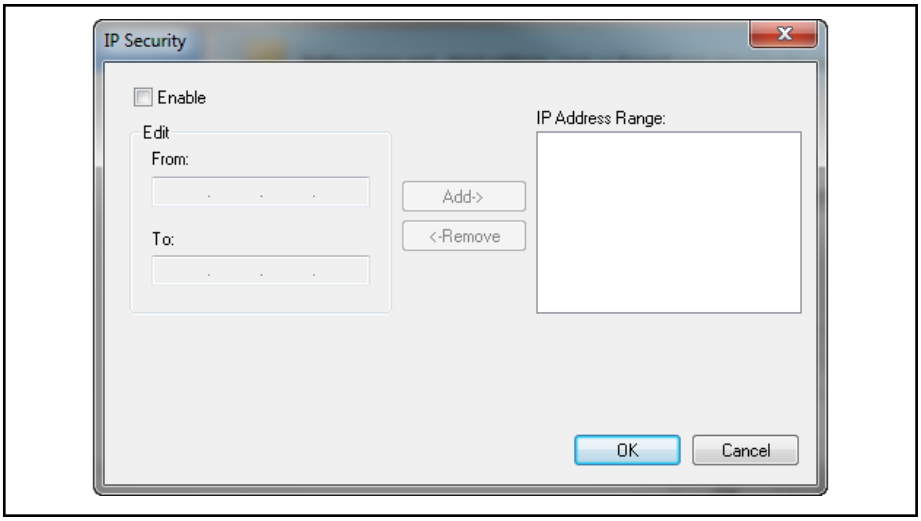


Fig. 5-33: "IP security" dialog

The parameters of this dialog are used to specify the range of IP addresses for the computers that are allowed to access the project as thin clients. This option is useful if the IP address of the thin client computers that are allowed to connect to the web server can be controlled.

Click the **Enable** checkbox and enter the IP addresses in the **From** and **To** fields in the active **Edit** pane to specify the IP address range. Use the **Add** and **Remove** buttons to move the IP addresses into the **IP address range** list. If a thin client attempts to connect to the server, it checks for an IP address for the Thin Client station that is within any range configured in the "IP security "dialog. If none is found, the server refuses the attempt to connect to the thin client station.

☞ By default, IP security applies only to thin clients connecting to the data server. IP security can also be implemented for database synchronization between projects running on different stations on page 453. To do so, insert the following parameter into the project file (**project_name.app**):

[TCP]

UseWebIPSecurity=1

Extended settings

Advanced: Click this button to open a dialog to edit the Advanced web settings. In most cases, these settings do not need to be modified. However, depending on the project architecture, it is possible to configure advanced settings.

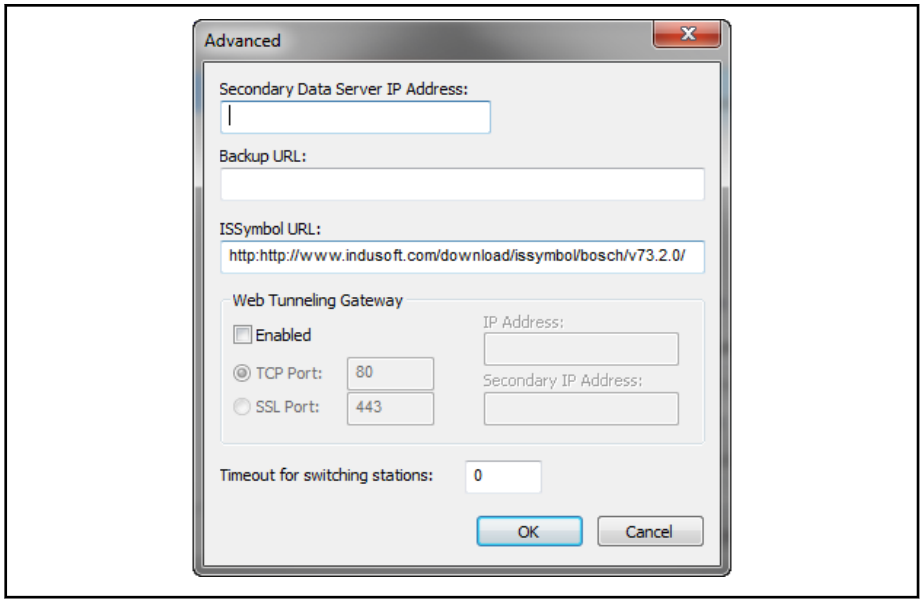


Fig. 5-34: "Advanced" dialog

- **Secondary data server IP address** field: Enter the IP address (or host name) of the secondary data server station. The data server station is the computer or device on which the TCP/IP server module of "WinStudio" is running. This field has to be filled if data servers are to be used that are redundant from the thin clients. If the primary data server fails, the thin client attempts to connect to the secondary data server automatically.
- **Backup URL** field: Enter the URL where the web files are stored (files from the "Web" subfolder of the project folder). This URL is used to download the files from the secondary web server when the primary web server is not available.
- **ISSymbol URL** field: If the thin client connects to the server, it attempts to load the

  Control ISSymbol on page 516. If ISSymbol is not registered on the local computer (thin client), the browser attempts to download it from the URL specified in this field. The default URL is a web site where "Bosch Rexroth" keeps the latest version of ISSymbol available for download. A different location may need to be configured, especially if the thin client computer is not connected to the internet. The **ISSymbolVM.cab** (stored in the **\BIN** subfolder of "WinStudio") has to be available for the URL configured in this field.

☞ If the thin client stations do not have access to the internet, it is recommended that the **ISSymbolVM.cab** file is made available to the web server station and that the respective URL is configured in this field.

☞ Web browsers running under the Windows CE operating system cannot automatically download the Control ISSymbol (**ISSymbolCE.ocx**) from a remote location. **ISSymbolCE.ocx** has to be registered manually in the Windows CE device to be used as a Thin Client.

- **Web tunneling gateway**: Enable this option to activate the web tunneling gateway. Depending on the architecture on page 519 of the project, the web tunneling gateway may be required to route the Thin Client computers to the data server.
- **TCP port**: Select this option when using HTTP with the web server of Microsoft IIS. Specify the TCP port used by the HTTP web server here ("80" is the default TCP port for HTTP protocols).
- **SSL port**: Select this option when using SSL (Secure Socket Layer) with the Microsoft IIS web server. The TCP port used by the HTTPS web server can be specified here ("443" is the default TCP port for HTTPS protocols).
- **IP address**: IP address of the web server computer where the web tunneling gateway is running. This has to be the IP address of the web server accessible from the thin client station(s).
- **Secondary IP address**: IP address of the Web server computer where the secondary web tunneling gateway is running. This has to be the IP address of the secondary web server accessible from the thin client station(s). This field has to be configured if redundant web servers are to be used.



Most of the web settings can be modified dynamically by the SetWebConfig on page 804 function. It is especially useful if the project is to be created just once and to make sure that the web settings are automatically configured when running the project on different stations with different IP addresses.

5.5 WinStudio tools

WinStudio offers various tools to support the creation, configuration and analysis of WinStudio applications (**WinStudio ▸ Tools ▸ ...**).



Some functions are only available in a certain context. Please note the instructions for use in the descriptions of the individual tools.

- **WinStudio library**: Opens the "WinStudio symbols" tool window. Here, numerous preconfigured symbols can be found that can be used in WinStudio screens. Refer to Symbols on page 222.
- **Reset variables**: Resets all variables to their original values. Refer to Tags database reset on page 149
- **Variable import wizard**: Opens the "Variable import wizard" to transfer WinStudio variables from other projects to your current application. Refer to Importing from Other Project Databases on page 154
- **Database spy**: Opens the "Database Spy" tool window. This tool used to display and modify variables in the tags database. Refer to Database spy on page 78
- **LogWin**: Opens the output window in which the messages of the WinStudio runtime module of the local station can be observed. Refer to LogWin output window on page 79
- **Cross references**: Opens the "Cross references" tool window to search for the application of WinStudio variables in the current project. Refer to Cross reference list on page 82

- **Add user**: Opens a configuration dialog to create users. Refer to Creating and configuring users on page 478
- This function is only available for the application type "WinStudio".
- **Add group**: Opens a configuration dialog to create user groups. Refer to Creating and configuring groups on page 474

  This function is only available for the application type "WinStudio".
- **Security settings**: Opens the "Security settings" dialog. In this dialog, the WinStudio user management can be activated and configured. Refer to Using the security system configuration wizard on page 463

  This function is only available for the application type "WinStudio".
- **Log in / Log out user**: Opens the login dialog. Refer to Logging On/Off on page 482
- **Remote database spy**: Opens the input dialog to connect to other devices to display and modify their WinStudio variables. Refer to Remote database spy on page 502
- **Remote LogWin**: Opens the input dialog to connect to other devices to display their messages of the WinStudio runtime environment. Refer to Remote LogWin on page 503
- **Status**: Opens the "Execution tasks" dialog. This dialog displays an overview of all tasks and their starting mode and current execution status.
- **Event logger**: Opens the "Event logger" dialog. Use this dialog to activate and configure logging of WinStudio messages. Refer to Events and event logs on page 326
- **Load project**: Opens the Windows file browser. Already available WinStudio projects can be selected and applied to the current project

NOTICE	**If a different WinStudio project is loaded, all files of the current WinStudio project is deleted.**

  In case of doubt whether to delete all existing files, cancel the process with the "Cancel" button.
- **Web settings**: Opens the "Web settings" configuration dialog. If the project is to be executed as a web application, configure the web server in this dialog. Refer to Web tab on page 71

  This function is only available for the application type "WinStudio".
- **Save all documents as HTML**: Creates or updates a HTML file for all WinStudio elements. Refer to Creating a simple thin client program on page 507
- **Change password for all documents**: Changes the password for all protected files. Refer to Password protection for screens, symbols and worksheets on page 484.

5.6 Screen/Worksheet Editor

Create or edit any kind of screens and worksheets for projects using the screen editor. Information can be input using the mouse and keyboard and control data can be output to processes and screens can be updated and animated based on the data input from processes.

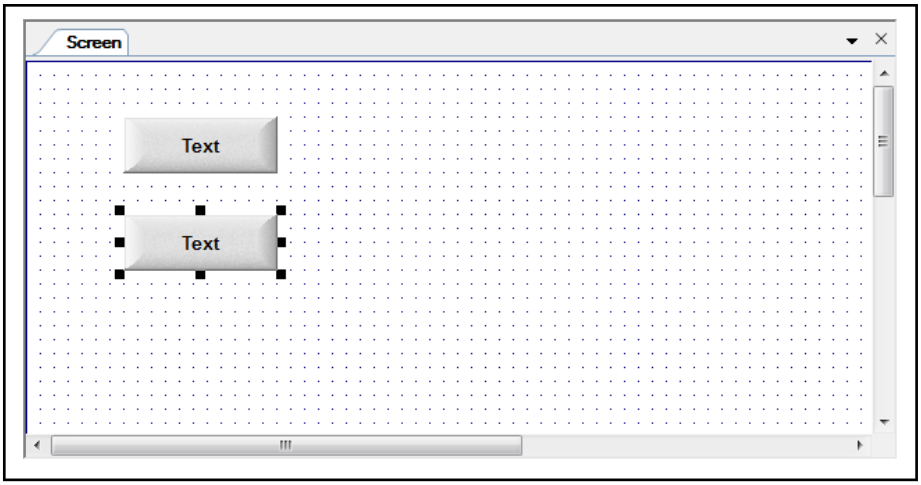


Fig. 5-35: Screen/worksheet editor

Other screen editor features include:

- Point-and-click, drag-and-drop support for easy screen creation
- The option to group objects to keep their arrangement
- To edit group elements without ungrouping them
- Handling bitmap objects and background bitmaps

5.7 Database spy

The "Database spy" window is a tool for error analysis to display and change tag values and execute and test functions and mathematical expressions.

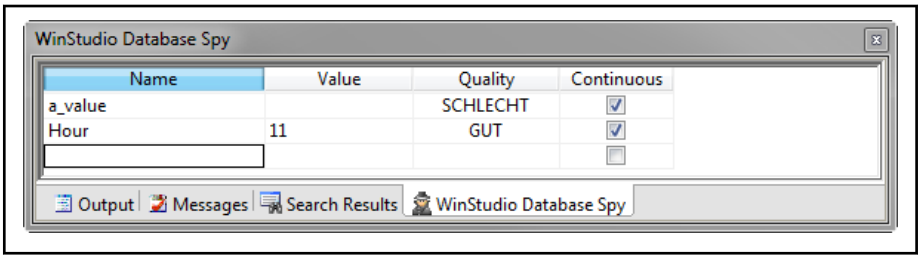


Fig. 5-36: Example of database spy window

The window displays the following information:

- For each desired date, the following is displayed at runtime:
  - **Name**: Indicate an application variable, an internal variable or an expression to be monitored
  - **Value**: Displays the value returned by the tag/expression
  - **Quality**: Displays the quality (GOOD or BAD) of the value returned by the tag/expression
  - **Continuous**: Select this option to have the project continuously evaluate the tag/expression

In order to monitor a variable, enter its name into the column or click on an empty field in the "Name" column and open the context menu. By means of the "Insert variable or expression" function, the browser dialog can be opened alternatively and a variable can be selected.

The database spy is dockable, i.e. it can be moved to another location in the development environment.

Click on the title bar and drag the window to a new location.

Release the mouse button to attach or dock the window to its new location.

5.8 LogWin output window

Use the "WinStudio log" window to display error analysis messages generated during project runtime. The window displays OPC, DDE, and TCP/IP transactions, module activations, trace messages, and so on.

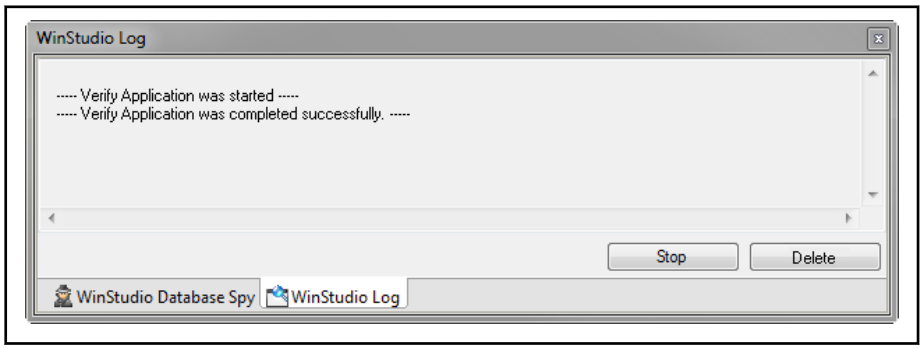


Fig. 5-37: Example of output window

The "WinStudio log" window is dockable, i.e. it can be moved to another location in the development environment. To do so, click on the title bar and drag the window to a new location. Release the mouse button to attach or "dock" the window to its new location.

Configuring the log to show additional information

By default, the log shows only error analysis and error messages — that is, messages indicating that the project is not running properly. If the log displays **all** messages created by "WinStudio", the output quickly becomes confusing.

To configure the messages displayed in the log, proceed as follows:

1. Right-click anywhere in the "WinStudio log" window and then click **Settings** in the context menu. The "Log settings" dialog is displayed:
2. On the "Log options" tab of the dialog, select the specific types of messages that are to be displayed in the log.

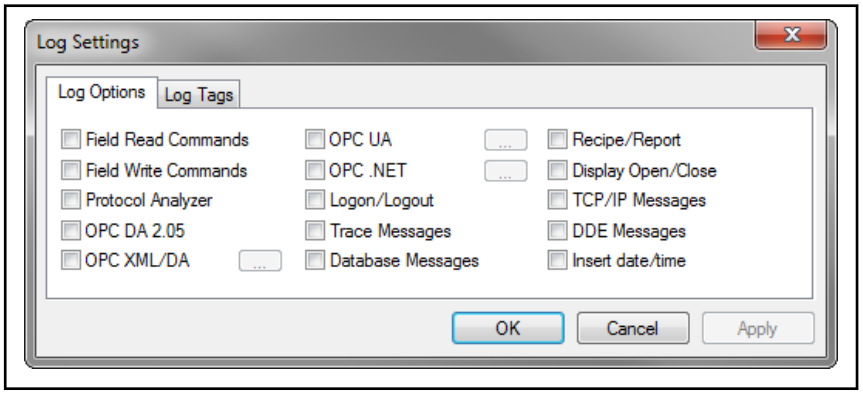


Fig. 5-38: Log settings - Options tab

Options	Description
Field read commands	Displays all field read commands that are sent to connected devices
Field write commands	Displays all field write commands that are sent to connected devices.
Protocol analyzer	Displays messages generated by configured device drivers on page 413.
OPC DA 2.05	Displays messages generated by the OPC DA 2.05 Client runtime task, page 427.
OPC XML/DA	Displays messages generated by the OPC XML/DA Client runtime task.
OPC UA	Displays messages generated by the OPC UA Client runtime task, page 433.
OPC .NET	Displays messages generated by the OPC.Net Client runtime task.
Logon/logout	Displays a message whenever a user logs on or logs out (For further information, refer to Security on page 461.)
Trace messages	Shows messages generated by the **Trace()** function. This function is used to generate customized messages from the project to trace execution of the application
Database messages	Displays messages generated by ODBC on page 397 and ADO.NET on page 404 database interfaces
Recipe/report	Displays messages of the tasks for recipe on page 395 and reports on page 396.
Displaying Open/close	Displays detailed information whenever a screen on page 170 is opened or closed: **Disk load time:** Time to load the screen file from the disk into memory **Open time:** Time to load the WinStudio screen, including initialization tags used in the screen and running any "OnOpen" scripts or functions. **Total load time:** Total time to load the screen (includes disk load time and open time above). **"First draw" time:** Time for the first drawing of screen objects **"First OnWhile" time:** Time for running any "OnWhile" scripts or functions for the first time **Total open time:** Total time to open the screen (includes first draw time and first OnWhile time as described above). **Close time:** Time to close the screen, including finalizing tags used in the screen and running any "OnClose" scripts or functions. **Total close time:** Total time to close the screen, including the time to write the screen file on the disk This information can be used to analyze runtime performance on low-end end systems. If a particular step of opening or closing takes an unusually long time, it can be identified and redesigned
TCP/IP Messages	Displays messages generated by TCP/IP communications on page 453.
DDE messages	Displays messages generated by DDE communications on page 455.
Insert date/time	Add a timestamp to each message

Tab. 5-1: Log settings - Options tab

3. If **OPC XML/DA**, **OPC UA**, **OPC .NET** was selected, specify precisely which messages are to be displayed in the output window.

   Required steps:

   Click the Browse button to the right of the option. An additional Log settings dialog is displayed.

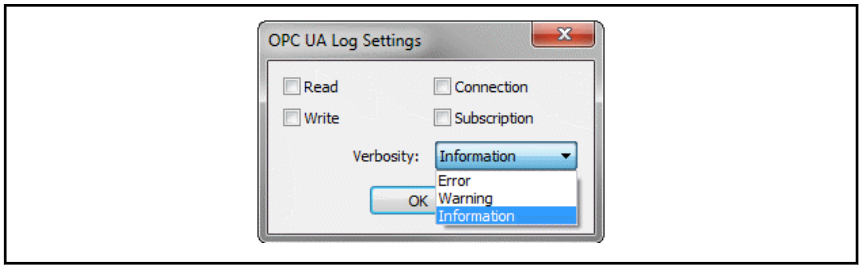


Fig. 5-39: OPC UA log settings dialog

Each task has its own settings but the descriptions of the settings are the same for all three tasks.

Select the type(s) of messages to be displayed

Option	Description

Connection	Activate trace messages for changes of the connection status between the OPC client (i.e. your project) and the OPC server.
	Example of messages:
	`OPC UA: Connection established with server "Connection1"`
	`OPC UA: Error to connect to server "Connection2" - Time out waiting for server response`

Subscription	Activate trace messages for subscriptions to server elements such as the creation of new subscriptions and changes of the data type. Example of messages: `OPC UA: Group 1 => Subscription created`

Read	Activate trace messages for reading processes.
	Example of messages:
	`OPC UA: Read Group 1 Started - OK`
	`OPC UA: Read Group 1 Completed - OK`
	`OPC UA: Read Group 25 Started - Error, asynchronous reading pending for the current group`

Write	Activate trace messages for writing processes.
	Example of messages:
	`OPC UA: Write Group 1 Started [Line 1 = 10.25, Line 42 = 20.45] - Status OK`
	`OPC UA: Write Group 1 Completed [All Items] - Status OK`

Select the verbosity level from the Verbosity list.

Option	Description
Error	Critical problems which have lead to the breakdown of the communication. These problems have to be remedied before the communication can be resumed.
Warning	Non-critical problems which may lead to the breakdown of the communication under different conditions. These problems are to be remedied as soon as possible. Includes the above-mentioned **errors**.
Information	All trace messages for communication. This is the default setting. Please note that these messages may be extremely detailed depending on which type(s) of messages were selected for display and how many member subscriptions were created. Includes the above mentioned points **Error** and **Warning**.

Click on **OK** to close the dialog.

4. In the "Log Tags" tab of the dialog, click **Add** to browse for project tags on page 104.

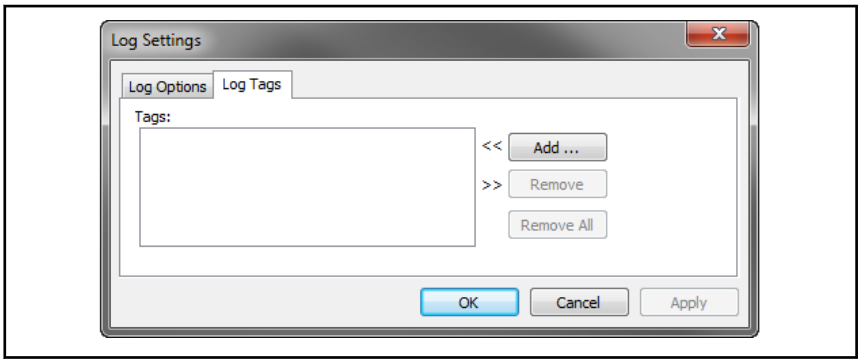


Fig. 5-40: Log settings — Tags tab

The log pane of the LogWin window displays a message if the value of a selected tag changes.

5. Click **OK** to save the settings and close the "Log settings" dialog.

5.9 Cross reference list

To find out in which worksheets a variable is used, open the tool window **WinStudio ▸ Tools ▸ Cross references**.

To search all worksheets, enter the name of the variable to be searched for in the **Tag name** text field. As an alternative, open the variable browser via the  button to select a variable. The search is started after confirmation of the variable name with Enter. Afterwards, all hits are listed in the log window (also refer to output on page 79).

By double-clicking on a hit with the mouse, the respective worksheet is opened in the editor and the element using the searched variable is selected.

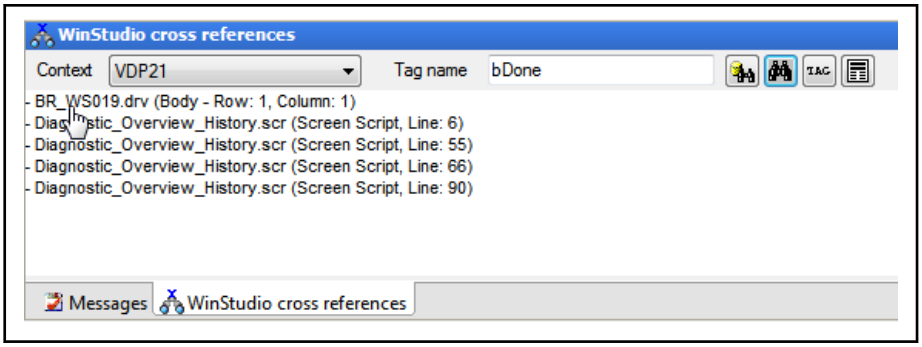


Fig. 5-41: Cross reference search results

If a project consists of several visualization devices, select "Context" in the selection list to specify in which visualization device the WinStudio search is to be carried out.

5.10 Text search

When searching for a specific text in the WinStudio project, use the search function. All WinStudio documents are browsed for the text. This way, the user can also search text in commented out script codes. In case of a targeted search for the location of a WinStudio variable, use the cross reference list.

Start the WinStudio text search via the context menu of the "WinStudio" folder using the "Search..." entry. Any text can be entered in the search dialog. Subsequently, specify under "Options" if whole words only are to be searched and if the search is to be case sensitive.

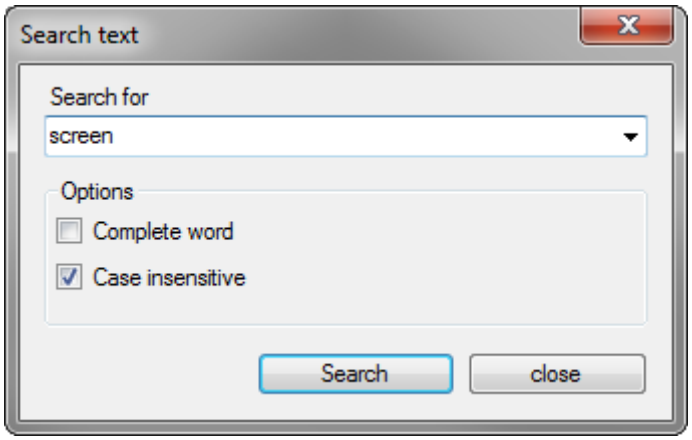


Fig. 5-42: Input dialog for WinStudio text search

Start the search by pressing "Search". The found hits are displayed in the "Search results" tool window. If a search text is available multiple times in an element, an individual entry is added to the hit list for each hit. For each hit, name and location of the element are displayed. Additionally, the complete text position in the "Text" column as well as the exact position within the element is displayed.

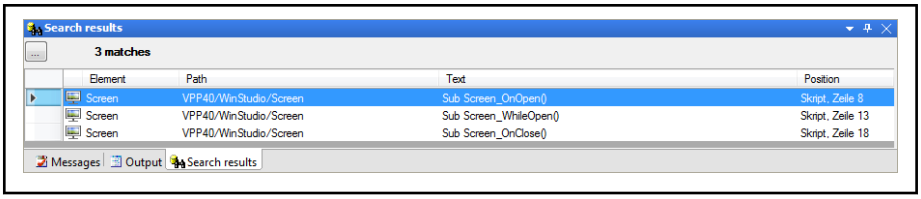


Fig. 5-43: Hit display of a WinStudio text search

If you want to analyze your hits in detail or if you want to change the text at the found position, double-click on the line in the result window to open the editor. The hit list is not automatically refreshed during editing. Open the search dialog again in the event window using the "..." interface and repeat the search.

The WinStudio search always refers to the WinStudio project in which the search has been started. Searching across all WinStudio projects within an IndraWorks project is not possible.

The WinStudio search exclusively finds text in saved elements. Save the element first if you want to edit it in the editor when searching for a previously added text.

5.11 Status Bar

The status bar can be found at the bottom of the development environment. It displays information on the active screen, e.g. the number of tags already used.

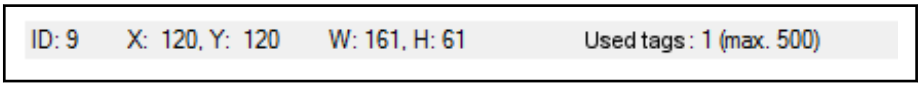


Fig. 5-44: Example of status bar

The status bar fields (from left to right) are described in the following table:

Field	Description
Object ID	ID number of a selected screen object
Cursor position	The position of the cursor within the active screen or the active worksheet For screens, the position of the mouse cursor is given as X/Y coordinates, where X is the number of pixels from the left edge of the screen and Y is the number of pixels from the top edge of the screen to the cursor For worksheets, the position of the text cursor is given as line and column
Object size	The size (in pixels) of a selected screen object, where W is the width and H is the height
Used tags	Number of the tags already used in the project and the maximum possible number of variables for the current project type

Tab. 5-2: Status bar fields

6 WinStudio projects

IT security

To operate systems and machines, a holistic IT security concept, corresponding to state-of-the-art technology is required. Bosch Rexroth products are part of the holistic IT security concept. Bosch Rexroth product properties have to be taken into consideration in a holistic IT security concept. The properties to be taken into consideration are documented in the IT security guideline.

See Rexroth media directory:

DE: DOK-IWORKS-SECURITY***-PRRS-DE-P

EN: DOK-IWORKS-SECURITY***-PRRS-EN-P

Additionally to IT security measures, it is explicitly recommended to use the WinStudio project security system. The project security sytem manages how users and user groups can access projects during development and at runtime.

The WinStudio security system is only available if the **application type** "WinStudio application" is selected in the properties of the visualization device. For other application types, functions are protected during runtime by the IndraWorks function "Access protection".

For more information, refer to:

- chapter 12 "Project security" on page 461
- chapter 12.7 "Security system dialog" on page 480

6.1 Creating a new project

6.1.1 Creating a new project, general information

WinStudio projects are always assigned to a visualization device in an IndraWorks project. To add a new visualization device to an open IndraWorks project, proceed as follows:

1. Add a visualization device to the IndraWorks project. To do so, the following options are available:
   - Open the **Library** window (in the "View" menu) and select the appropriate visualization type in the **Visualization** tab by clicking on it. Afterwards, keep the mouse button pressed and drag the selected visualization type to the project folder in the Project Explorer and release the mouse button.
   - Open the context menu of the project folder and select the visualization type of the visualization device in the **Add** submenu.

   In both cases, the HMI device wizard opens.
2. In the HMI wizard, select the appropriate entry from the "Device type" selection list.
3. In the "Name" input field, enter a text under which the device is to be displayed in the Project Explorer.
4. In the "IP Address" field, enter the computer name or the IP address of the device.
5. Select one of the following entries from the "Application type" selection list:

- **WinStudio application**: Select this application type if an independent application is to be created that does not require additional visualization objects like F, M and OP panels or various standard applications (diagnostics, operating screens, etc.). The use of WinStudio-internal user management as well as support of Web clients is only available for this application type.

  With this application type, the project is executed in the WinStudio runtime environment.
- **HMI standard application**: Select this application type if panels and standard applications are to be used in addition to WinStudio objects. WinStudio is executed within the IndraWorks runtime.
- **MTX standard application**: Select this application type if panels and MTX control-related standard applications are to be used in addition to WinStudio objects. WinStudio is executed within the IndraWorks runtime.

6. Under "Device version", select the version of the runtime environment that is installed on the visualization device.
7. Use the "Finish" button to confirm the settings. The HMI wizard creates a new folder for the visualization device in the Project Explorer. The WinStudio folder can be found under the Visualization folder:
   - With type **WinStudio application**, under the Visualization folder.
   - With type **HMI standard application** and **MTX standard application** under the "Screen application" folder.

When creating a visualization device, the following settings are adjusted according to the device and application type:

- **Target platform**: For VEx devices with Windows Embedded operating system, "WinStudio CE Lite" is selected and "WinStudio Lite" for all other devices.
- **Resolution**: With WinStudio applications, the resolution corresponds to the size of the application window at runtime. This is usually the screen resolution of the target device (e.g. 1024x768). At HMI and MTX applications, this configuration shows deviations from the screen resolution as panels and headers are additionally displayed by the runtime environment. Thus, the effective range for WinStudio screens is lower.
- **Viewer task starting mode**: For WinStudio applications, the value is set to "Automatic". For HMI and MTX applications, the viewer task is started by the IndraWorks runtime environment, i.e. the value has to be set to "Manual".

6.1.2 Target platforms and product types

Target platform and product type determine important factors about the project, such as how many tags the project supports and which features can be used at runtime.

Target platform The target platform is generally either Windows or Windows Embedded.

Windows A computer that runs one of the following operating systems:

- Microsoft Windows XP Service Pack 3 or later
- Microsoft Windows Vista Service Pack 2
- Microsoft Windows 7 Service Pack 1 or higher
- Microsoft Windows Server 2003 Service Pack 2 or later
- Microsoft Windows Server 2008, R2 Service Pack 1

- Microsoft Windows Server 2012
- Microsoft Windows XP Embedded Service Pack 3
- Microsoft Windows Embedded Standard 7 (2009)

The computer requires the installed full "Rexroth WinStudio" software, even if it is not used for project development, because the full software includes the modules that are needed to host the project runtime on Windows. The computer also requires an appropriate runtime license key. For further information, refer to Installation on page 21.

If the target platform is **Windows**, all project features are available.

Windows Embedded

An embedded device that runs one of the following operating systems:

- Microsoft Windows XP Embedded Service Pack 3
- Microsoft Windows Embedded Standard 7 (2009)
- Microsoft Windows Mobile or Windows CE 5.x/6.x
- Microsoft Windows Embedded Compact 7

The device requires either EmbeddedView (for Windows XP Embedded and Windows Embedded Standard) or CEView (for Windows Embedded Compact) to be installed. The device does also require an appropriate runtime license key. For further information, refer to Installation on page 21.

Please note: If the target platform is **Windows Embedded**, certain project features are not available:

- All DDE features, including the DDE Server task and the DDE Client task and worksheets.
- ODBC runtime task, worksheets and functions. (ADO database interface is still available.)
- OPC HDA server task.
- OPC .Net client task and worksheets.
- OPC XML/DA client task and worksheets.
- Minor graphic features such as support for image formats other than BMP or JPG and fill effects for shapes other than rectangle.
- Miscellaneous other built-in functions, as described in the documentation for the built-in scripting language.



In most cases, the following applies: If the project runtime is to be hosted on a Windows XP Embedded or Windows Embedded Standard device and project development is not to be performed on that device, only EmbeddedView is to be installed on the device and **WinStudio Embedded** selected as target platform. EmbeddedView has a smaller footprint than the full "Rexroth WinStudio" software and can be installed and managed remotely.

However, if any of the features that are not available when the **WinStudio Embedded** target platform is to be used, the full "Rexroth WinStudio" software has to be installed on the device and **WinStudio** has to be selected as target platform.

The full "Rexroth WinStudio" software cannot be installed on a Windows Embedded Compact device under any circumstances.

Product type The product type primarily determines how many tags can be created in the project, including tags shared or imported from other systems. The "WinStudio" project type supports considerably more tags than "WinStudio Embedded".

Note that the computer or the embedded device that will host your project runtime object, has to have a license key that matches or exceeds the selected product type. To verify the license key, run Protection Manager (**Start** ▸ **All Programs** ▸ **Rexroth** ▸ **WinStudio**) on that computer or embedded device. Although, the product type and the license key can subsequently be changed, we recommend to verify the license key and to select the correct product type now in order not to waste time by developing a project using more tags than allowed by the license.

Also, if you plan to run your project on multiple stations with different license keys, then we recommend that you develop for the lowest common product type.

Target system The target platform and the product type specify the target system which can subsequently be changed using the target system command on page 88.

6.1.3 Changing the target system of an existing project

On adding a visualization device, the target system is preconfigured according to the respective device type. The default settings have to be changed under the following circumstances:

- More than 500 tags are required for the application
- At least one of the following functions is to be used: Trends, recipes, reports, external databases
- The WinStudio-integrated user management or web functionality are to be used with the "WinStudio application" application type

If this does not apply to the project, the target system has to be changed to WinStudio 1.5k (for 1500 tags) or higher. All target systems above and including "WinStudio 1.5k" enable using the above mentioned functions.

To select "WinStudio 1.5k" or higher as target system, a respective WinStudio Engineering license is required. Otherwise, this target system cannot be selected.

To change the target system with a suitable WinStudio Engineering license, proceed as follows:

1. Open the WinStudio properties dialog.
2. Click on the "Options" tab.
3. Open the selection list in the **Target system** pane. A list of available product types is displayed. Select the new product type of the project.

   For further information, please refer to target platforms and product types on page 86.
4. Close the dialog with "OK".

The project is converted to the selected product type.



By changing from a WinStudio target platform to a WinStudio Embedded target platform, some features become unavailable.

6.2 E-Mail Configuration

Some features, such as alarms and certain functions, can be used to send E-Mails to designated recipients. To use these features, the project E-Mail settings have to be configured.

The E-Mail settings can be configured at any point at runtime by means of the "CnfEmail" function. However, default settings can be configured that are automatically used if the project is run for the first time and then restored as required at runtime, overwriting any changes made by calling the "CnfEmail" function.

1. Open the context menu of the "WinStudio" folder and select the entry **Tools ▸ E-Mail and FTP configuration**. The "E-Mail and FTP configuration" dialog is displayed.
2. Click the **E-Mail Settings** tab.

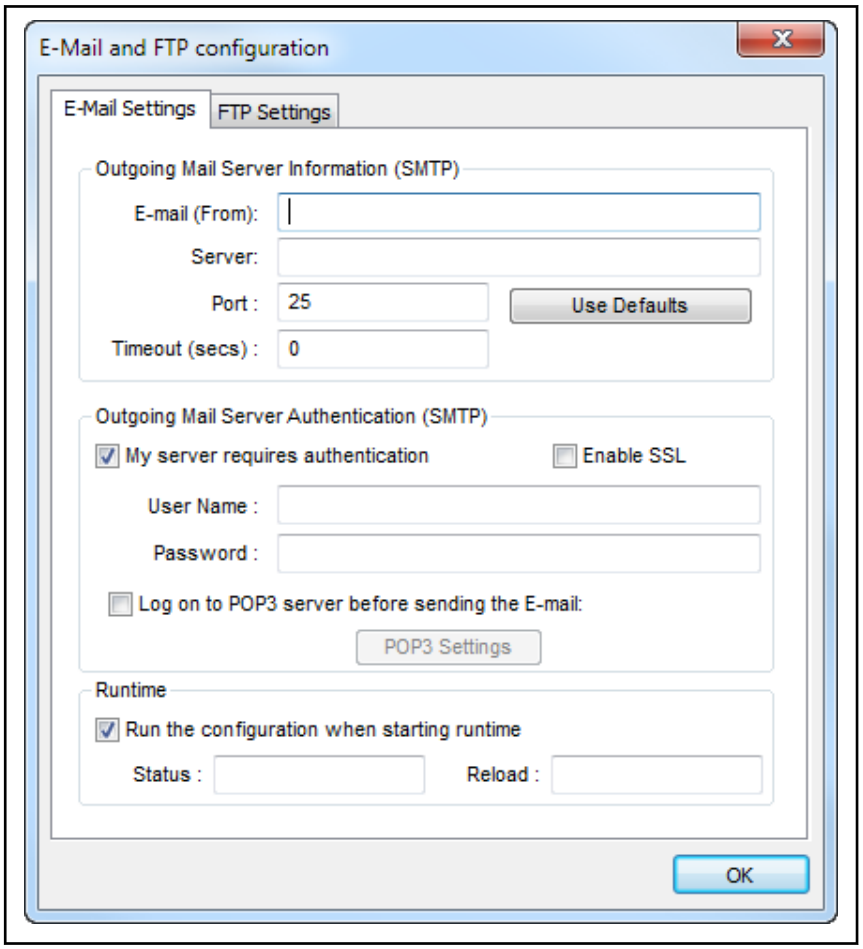


Fig. 6-1: E-Mail Settings tab

3. In the **E-mail (From)** field, enter the E-mail address of the sender.
4. In the **Server** and **Port** fields, enter the server address and port number of the outgoing mail server. The default port for SMTP is 25, however, it may vary according to the server and network configuration. Please contact your email provider.
5. If the outgoing mail server requires authentication, select **My server requires authentication**. If the authentication has to be encrypted additionally, select **Enable SSL**. Enter the access data into the **User Name** and **Password** fields.

   Most outgoing mail servers require authentication to prevent spamming and other abuse from unauthorized users.

Encryption via SSL is not supported in projects running on Windows Embedded target systems.

6. In the **Status** field, type the name of a tag (Integer type) that is to receive status codes when the project sends an E-Mail.
7. In the **Reload** field, enter a tag/expression. If the value of this tag/expression changes, the project reloads these default E-Mail settings.
8. Click **OK** to apply the changes and close the dialog.

6.3 FTP Configuration

WinStudio functions enable transmission of files between computers and FTPs. To enable these features, the project FTP settings have to be configured.

The FTP settings can be configured at any point at runtime by calling the CnfFTP on page 895 function. However, default settings can also be configured that are automatically used if the project is run for the first time and then restored as required at runtime, overwriting any changes made by calling the **CnfFTP** function.

1. Open the context menu of the "WinStudio" folder and select the entry **Tools ▸ E-Mail and FTP configuration**. The "E-Mail and FTP configuration" dialog is displayed.
2. Click the **FTP Settings** tab.

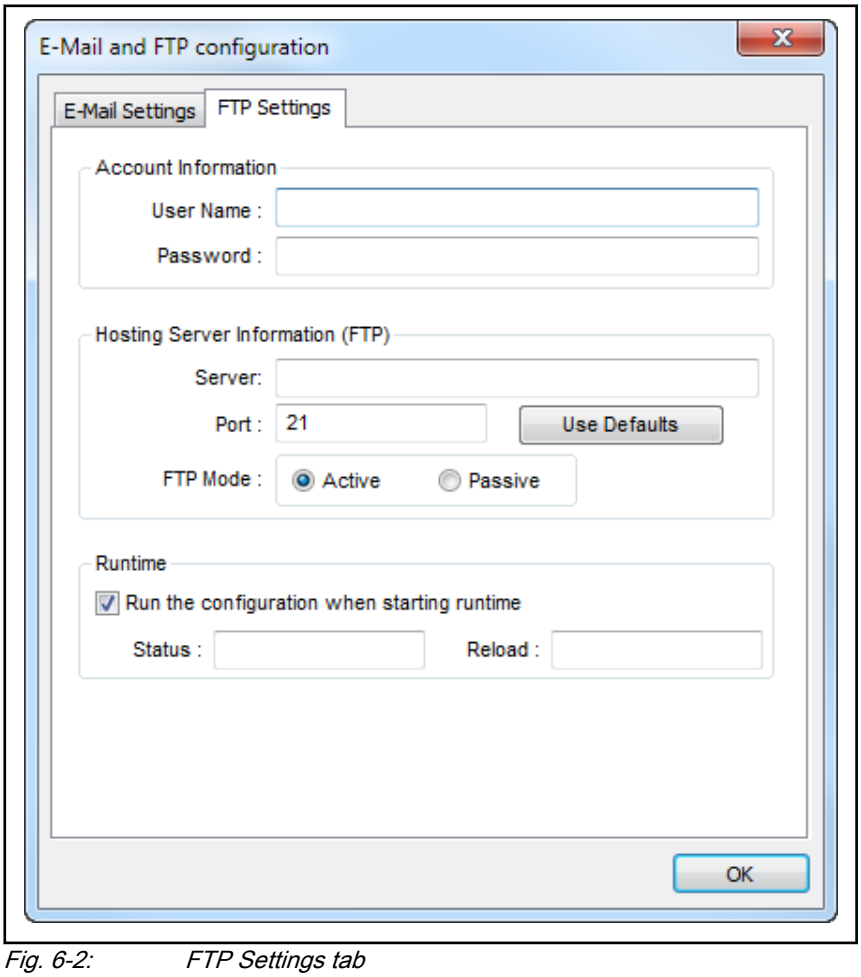


Fig. 6-2: FTP Settings tab

3. Enter the access data for the FTP server into the **User Name** and **Password** fields.
4. In the **Server** and **Port** fields, enter the server address and port number. The default port for FTP is 21, however, it may vary according to the server and network configuration. Please contact the server administrator.
5. Select **Active** or **Passive** mode, depending on the configuration of the server. Use the passive FTP mode to bypass some network firewalls. Again, please contact the server administrator.
6. In the **Status** field, enter the name of a tag (Integer type) that is to receive status codes if the project transfers a file.
7. In the **Reload** field, enter a tag/expression. If the value of this tag/expression changes, the project reloads these default FTP settings.
8. Click **OK** to apply the changes and close the dialog.

6.4 Execution tasks dialog

The "Execution tasks" dialog is used to configure which tasks have to be started automatically when the project is run, as well as to manually start/stop tasks at runtime.

The "Execution tasks" tab displays the list of available tasks for the current project. Their status and startup modes (**Automatic** or **Manual**) are also displayed.

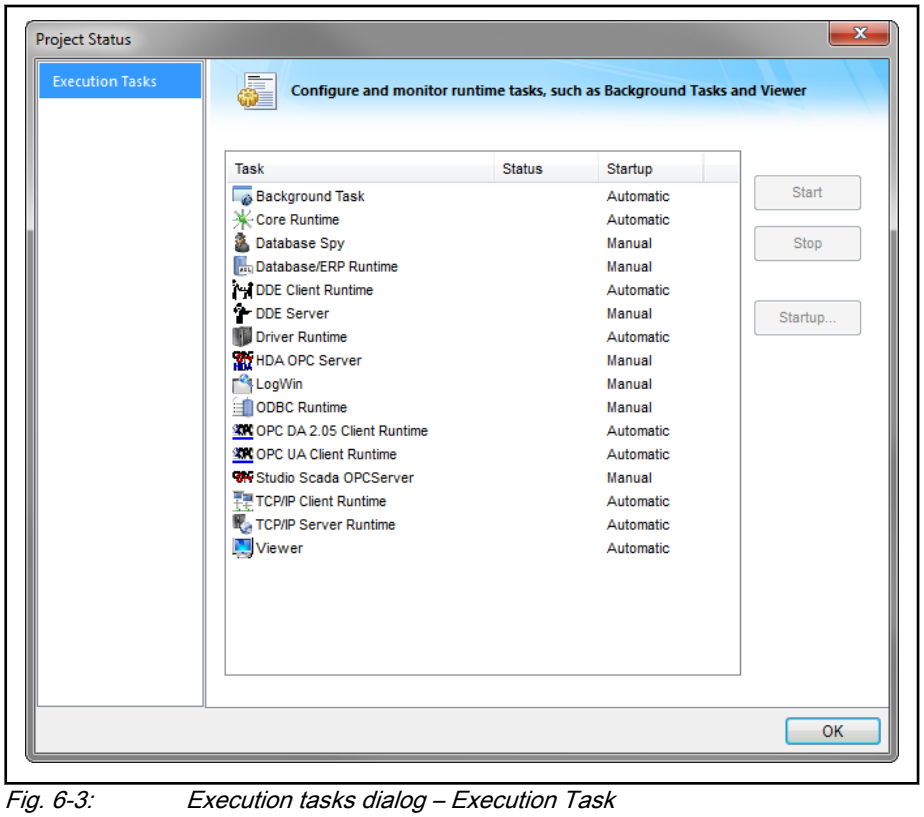


Fig. 6-3: Execution tasks dialog – Execution Task

The following table lists the tasks that are not available for projects running on Windows Embedded target systems:

Task	Available for Windows Embedded
Background task	Yes
Database client runtime program	Yes
Database spy	No
DDE client runtime program	No
DDE server	No
HDA OPC server	No
Driver runtime	Yes
LogWin	No
ODBC runtime	No
OPC client runtime	Yes
Studio Scada OPC server	Yes
TCP/IP client runtime	Yes
TCP/IP server	Yes
Viewer	Yes

To execute the task Studio Scada OPC server in EmbeddedView on a Windows-Embedded Standard computer, download the OPC core components to this computer and install them first. Download the required software from the OPC foundation website:

www.opcfoundation.org/Download-File,aspxCM=3&RI=820&CN=KEY&CI=274&CU=7

No actions are required for Rexroth WinStudio under Windows or CEView under Windows Embedded Compact, as the required software is automatically installed.

You can configure tasks for automatic execution when the project is run by selecting the task in the list, clicking Startup and then selecting Automatic.

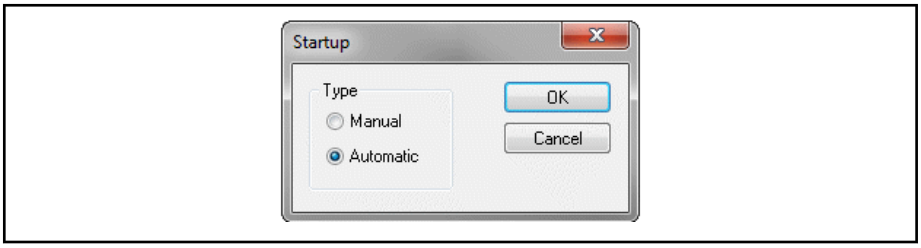


Fig. 6-4: Startup dialog of the execution tasks

The tasks configured with Startup = Automatic are automatically processed when executing the project.

Tasks with startup = Not manually. You can also Start/Stop each runtime task by clicking the name and then pressing the **Start** or **Stop** button.

You can also start/stop each task at runtime by using the StartTask page 721 and EndTask page 697 built-in functions. You can also use the IsTaskRunning page 699 function to check if each task is running at runtime.

6.5 Project execution under Windows services

"WinStudio" projects can be configured to run under "Windows services". Microsoft Windows services, formerly known as NT services, enable the creation of long-running programs that run in their own Windows sessions. These sessions can be automatically started when the computer boots, can be paused and restarted and are not displayed on any user interface. These features make services ideal for use on a server or whenever long-running functionalities are required that do not interfere with other users who are working on the same computer. Services can also be executed in the security context of a specific user account that is different from the logged-on user or the default computer account. For further information on services, please refer to the Microsoft Developer Network (MSDN) Library.

Why are projects to be executed under Windows services?

- To ensure that the project always runs with whatever system privileges it needs, regardless of the privileges of the user that is currently logged on to Windows
- To prevent the user from interfering with the project while it is running
- To let the project keep running if there is no user logged on at all

Creating and configuring the Windows service



All of the procedures described below were tested using Microsoft Windows XP Service Pack 3. To perform these actions, the logged in user requires administrative privileges and should know how to use the "Computer management" console. (To access the console, right-click the **My Computer** icon and choose **Manage** from the shortcut menu.)

There are two ways to create and configure the Windows service for the project: The "Service configuration" tool in WinStudio itself or the command-line utility program.

Tool service configuration

New services can be configured and executed in the development environment by selecting **Service configuration** in the WinStudio context menu. The "Service configuration" dialog is opened:

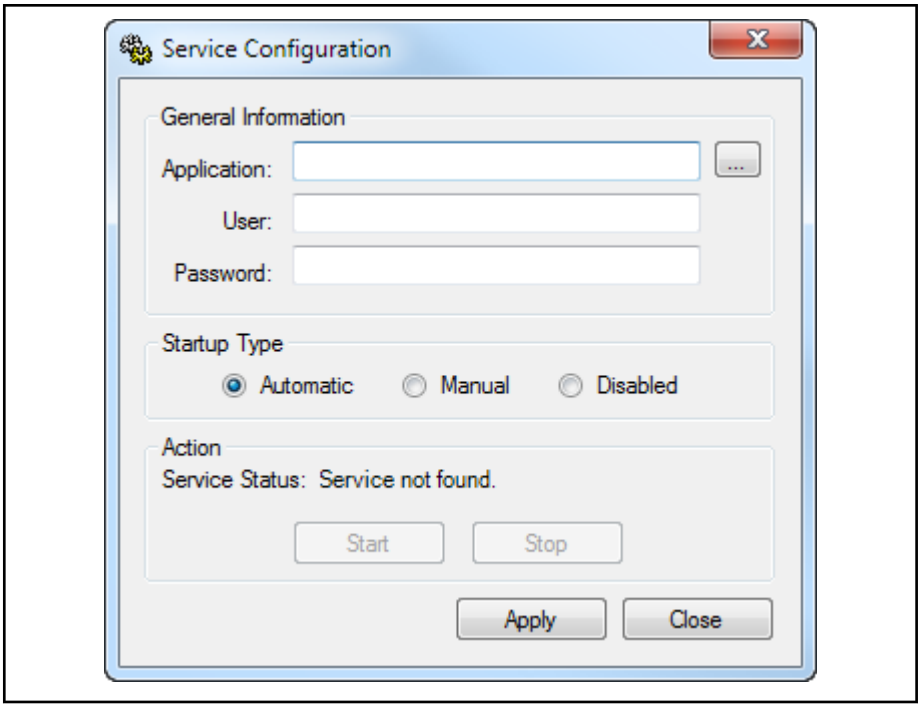


Fig. 6-5: "Service configuration" dialog

Application field

The location of the project file (**project_name.app**) that the service loads and runs when it is started. A complete file path has to be entered.

User field

The Windows user account under which the service is to be executed. This setting is optional. If it is not used, the service runs under "Local system".

Try to avoid running the service under "Local system". This account has too many rights for the file system and too few rights to properly execute OPC client and server modules on page 427. The best alternative is to create a user solely to run "WinStudio" and configure its rights to fit the project requirements. For more detailed information, refer to "Configuring user rights" below.

Password field

The password for the specified user account. This setting is optional. It is not required if no user is specified or if the specified user does not have a password.

Startup type field

How the Windows service starts. The following options are available:

- **Automatic**: The service starts automatically when the computer boots.
- **Manual**: The service can be started manually in the "Computer Management" console or by clicking **Start**, as described below.
- **Disabled**: The service is created and then disabled. It cannot be executed before an administrative user enables it in the "Computer Management" console.

Action pane

Starting or stopping the service. Please note: These buttons are not enabled before the service is actually created.

Creating a new service:

1. Next to the **Project** field, click **…** to open a standard Windows file browser. Use the browser to select the project file
2. In the **User** and **Password** fields, enter the username and password (if any) for the Windows user account under which the service is to run
3. Select a **Startup type**
4. Click **Apply**. The service is created with the specified settings.

After the service has been created, it will appear in the "Services" console (**Start ▸ Control Panel ▸ Administrative Tools ▸ Services**) under the name "Rexroth WinStudio". Use this console to quickly stop and restart the service if the "WinStudio" development environment is not to be executed

Command-line utility

The service can also be configured by using the command-line utility **StdSvcInst.exe**. It offers a few more options than the service configuration tool described above – such as specifying a name and description for the service – and it can be used without launching the "WinStudio" development environment. The utility is located in the **\Bin** folder of the "WinStudio" program directory. To run the utility, open a command prompt, navigate to the `\Bin` folder and enter the command with the desired options.

The utility has the following command syntax:

`StdSvcInst { -create -app filepath -startup { auto | manual | disabled } -user` username `-password` password `-name` displayname `-descr` description `| -start | -stop | -delete }`

-create

Creates the Windows service.

-app filepath

Specifies the project file (`project_name.app`) that is loaded and executed if the service is started. (This is the same as the **Project** field in the "Service configuration" dialog.) The complete file path has to be specified and enclosed in quotation marks.

This command line parameter is "required" to create a new service.

-startup { auto | manual | disabled }

Specifies how the service starts. (Corresponds to the **Startup type** field in the "Service configuration" dialog.) This is an optional command line parameter. If it is not used, the default behavior for a new service is "manual".

-user username

Specifies the Windows user account under which the service runs. (Corresponds to the **User** field in the "Service configuration" dialog.) This is an optional command line parameter. If it is not used, the service runs under “Local system”.

-password password

Specifies the password for the given user account. (Corresponds to the **Password** field in the "Service configuration" dialog described above.) This parameter is optional. It is not needed if no user is specified or if the specified user does not have a password.

-name displayname

Specifies the service name that is displayed in the "Computer management" console. The name has to be enclosed in quotation marks. This is an optional command line parameter, the default name is "WinStudio".

-descr description

Specifies the service description that is displayed in the "Computer management" console. The description has to be enclosed in quotation marks. This is an optional command line parameter.

-start

Starts the service. This corresponds to starting the service using the "Computer management" console or by clicking **Start** in the "Service Configuration" dialog.

-stop

Stops the service. This corresponds to stopping the service using the "Computer Management" console or by clicking **Stop** in the "Service configuration" dialog.

-delete

Deletes the service.

Example: Creating the service

In this example, a new Windows service is to be created with the following options:

"WinStudio" Project File	`C:\Users\username\My Documents\WinStudio v7.4 Projects\project_name\project_name.app`
Startup mode	Automatic
User	"WinStudio"

Password	"WinStudio"
Service name	"Rexroth WinStudio"
Service description	"Starts a WinStudio project"



Please note: The system requires a "WinStudio" user account with password "WinStudio".

To create the service with the required options, proceed as follows:

1. Ensure to be logged in as a user with administrative rights.
2. Open a command prompt (**Start ▸ All Programs ▸ Accessories ▸ Command Prompt**)
3. Navigate to the Bin folder: `cd Bin`
4. Enter the following command:

```
StdSvcInst –create –app "C:\Users\username\My Documents\WinStudio v7.4 Projects\ project_name/project_name.app" -startup auto –user WinStudio -password WinStudio –name "WinStudio" –descr "Starts a WinStudio project"
```

If the procedure is successful, the system displays the message "Service created". Otherwise, an error message is displayed.

Example: Changing the project file

After creating the service, the "WinStudio" project file that it runs may need to be changed. To do so, use the `-app` command line parameter:

1. Ensure to be logged in as a user with administrative rights.
2. Stop the service if it is running.
3. Open a command prompt.
4. Navigate to the Bin directory.
5. Enter the command, for example, to set NTDemo as the project file:

```
StdSvcInst –app "C:\Program Files (x86)\Rexroth\WinStudio\Demos\ NTDemo\NTDemo.app"
```

Example: Deleting the service

To delete the service, proceed as follows:

1. Ensure to be logged in as a user with administrative rights.
2. Stop the service if it is running.
3. Open a command prompt.
4. Navigate to the Bin directory.
5. Enter the following command:

```
StdSvcInst –delete
```

Configuring user rights

The service runs under the rights of the user account specified in the **User** field of the "Service configuration" tool (or by the `-user` switch of the command-line utility). If "WinStudio" requires a system resource for which no user rights are available, it fails. Thus, the account needs to be configured to have the necessary rights.

The following actions can only be performed by a user with full administrator rights.

Enabling "Log on as a service" for a user account

Before anything else, the specified user account has to be enabled to log on to the computer as a service. To enable the account:

1. Open the "Local security settings" console (**Start** ▸ **Control Panel** ▸ **Administrative Tools** ▸ **Local Security Policy**).
2. In the console window, select the folder "Security Settings\Local Policies\User Rights Assignment".
3. In the list of available policies, double-click **Log on as a service**.

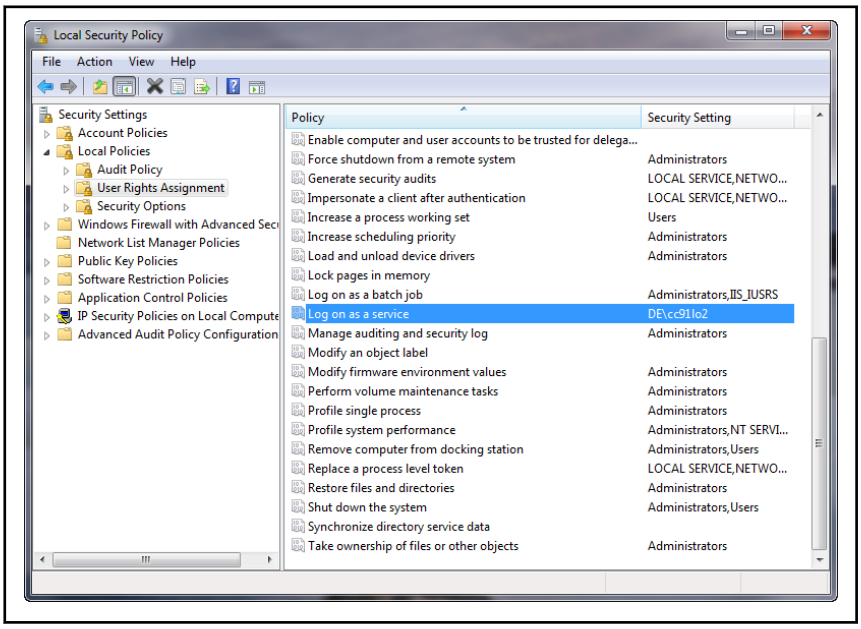


Fig. 6-6: Selecting the "Log on as a service" policy

The "Log on as a service" properties page is displayed.

4. Click **Add user or group**. The "Select users or groups" dialog is displayed.
5. Enter the name of the user account under which the service is to run.
6. Click on **OK**.

Giving the user account full control over the program directory

For the "WinStudio" project to run properly, the specified user account requires full control over the program directory and all of the files in it. To give the account those privileges, proceed as follows:

1. Open "Windows Explorer" (**Start** ▸ **All Programs** ▸ **Accessories** ▸ **Windows-Explorer**).
2. Select **Tools** ▸ **Folder options**. The "Folder options" dialog is displayed.
3. Click the **View** tab.
4. Ensure that the option **Use sharing wizard** is selected.

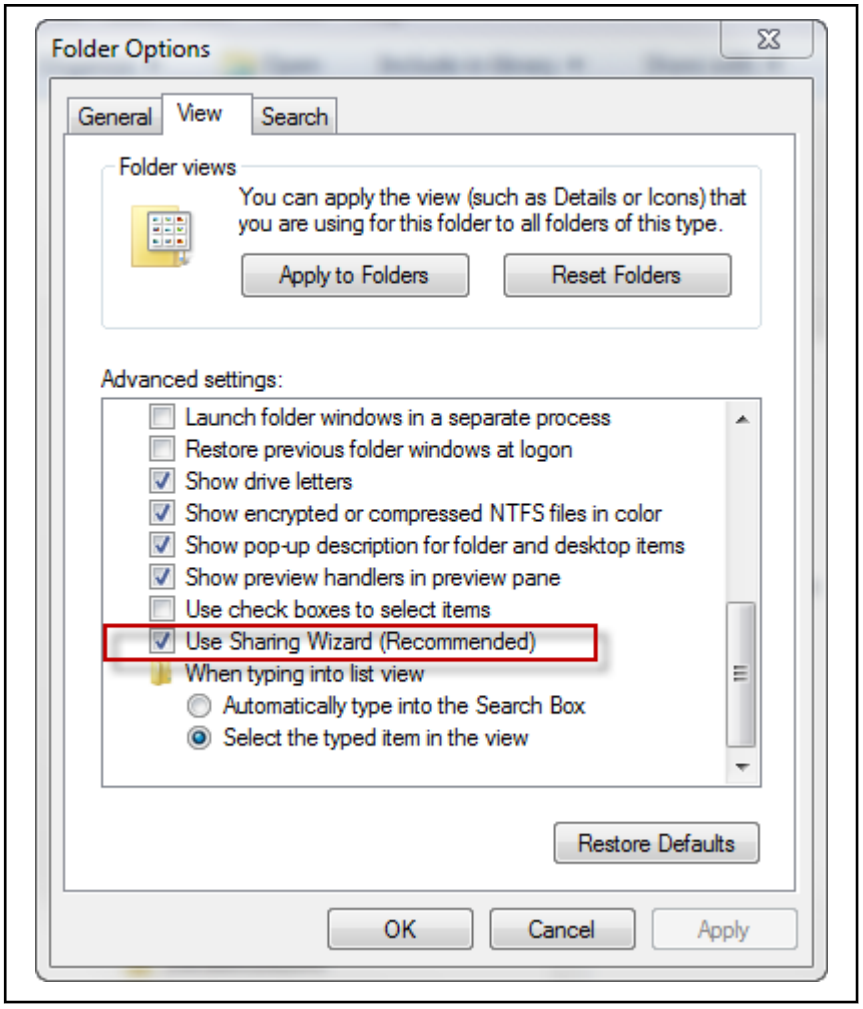


Fig. 6-7: Ensure that the option "Use sharing wizard" is enabled

5. Click **OK** to close the "Folder options" dialog.
6. Still in "Windows Explorer", locate and select the "WinStudio" program directory, i.e., the folder that contains the file `project_name.APP`.

   In this example, the folder is "NTDemo".
7. Right-click the folder and choose **Properties** from the shortcut menu. The “Properties” dialog of the directory is displayed.
8. Click on the **Security** tab.

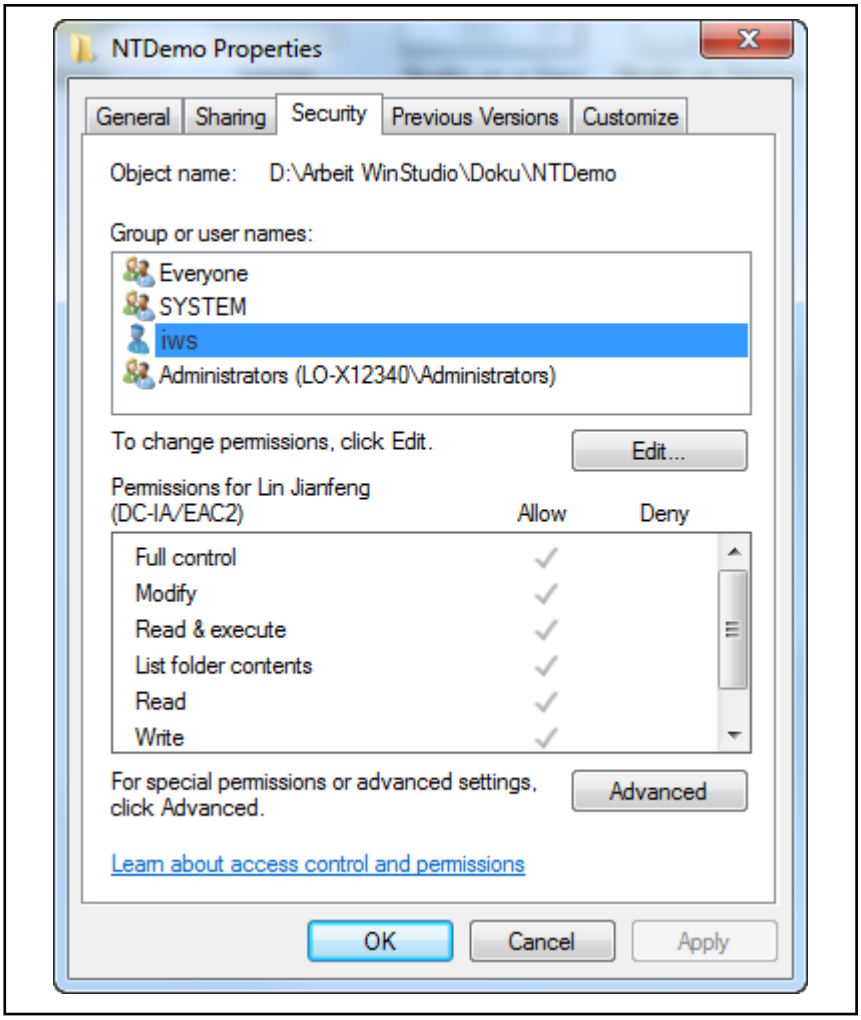


Fig. 6-8: "Properties" dialog for the program directory

9. Click **Add** and then add the user account that was specified when creating the service. In this example, the user is "iws".
10. Select this user and then set its "Full control" permission to **Allow**.
11. Click **OK** to apply the changes and close the dialog.
12. Close the "Windows Explorer".

Allowing the user account to run the OPC client/server module

As mentioned previously, normal users have too few rights to properly execute the OPC client/server module. Therefore, the user account has to be configured to have these rights:

1. Open the "Component services" console (**Start ▸ Control Panel ▸ Administrative Tools ▸ Component Services**).
2. In the console window, select the folder `Console Root\Component Services\Computers\My Computer\DCOM Config`.

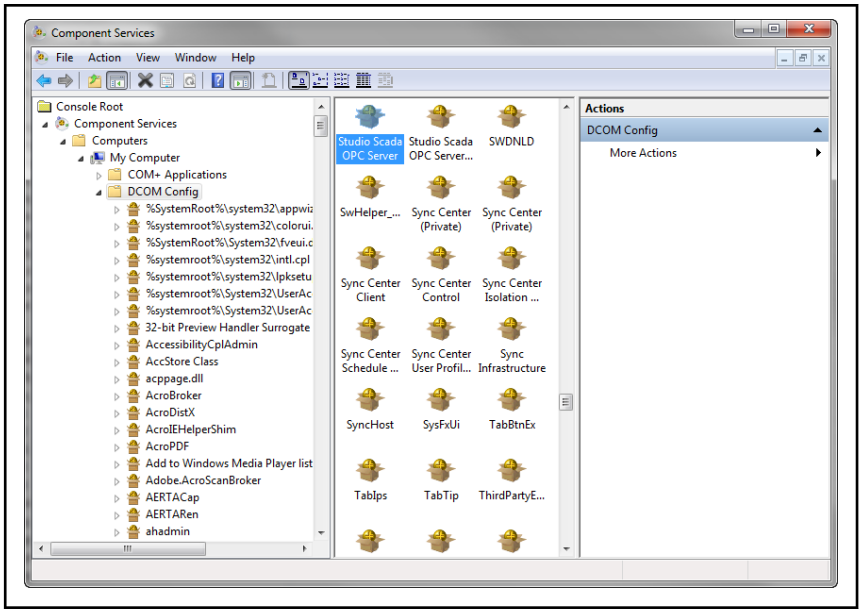


Fig. 6-9: Selecting the OPC Server module in the “Component services” console

3. In the DCOM Config pane, right-click the **WinStudio Scada OPC Server** icon and then select **Properties** from the context menu. The "WinStudio SCADA OPC server properties" sheet is displayed.
4. Click the **Identity** tab.
5. Click **This user** and then complete the fields with the same user and password that were specified when the service was created.

   In this example, the user is "iws" and the password is also "iws".

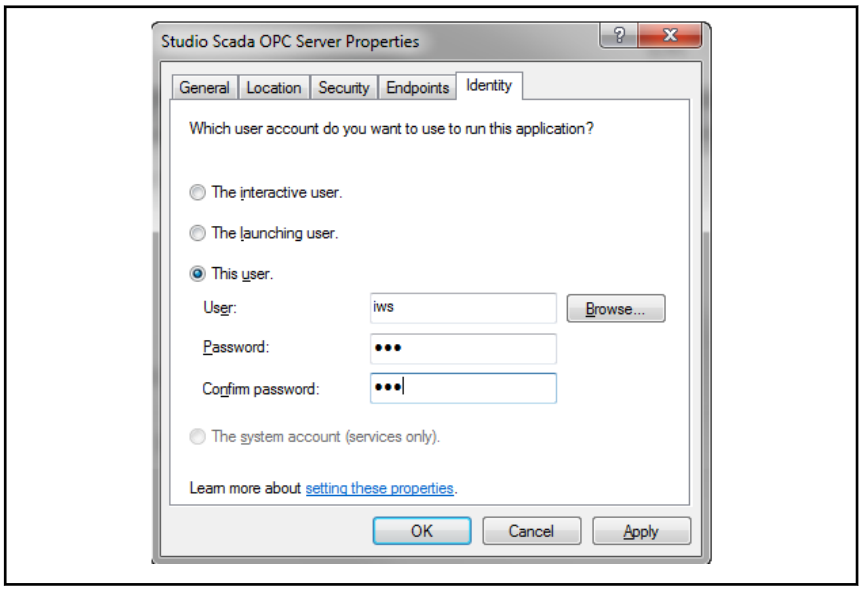


Fig. 6-10: Entering user name and password

6. Click **OK** to apply the changes and close the property sheet.
7. Close the "Component services" console.

Debugging

If "WinStudio" is running under Windows services, it has no user interface. Thus, occurring errors are only logged as Windows application events. Check the messages using the "Event Viewer" (**Start ▸ Control Panel ▸ System and Security ▸ Administrative Tools ▸ Event Viewer**).

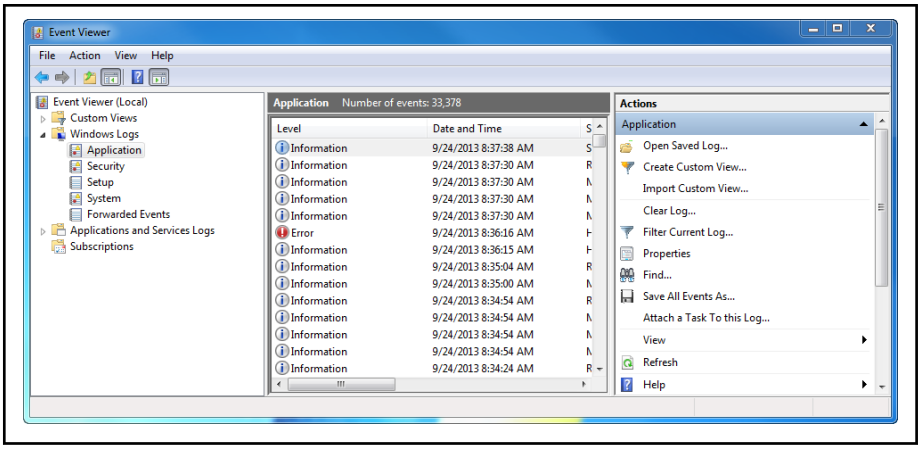


Fig. 6-11: WinStudio error messages in the event viewer

7 Working with tags

7.1 Tags and project database

7.1.1 Tags and the project database, general information

Tags are a core component of any "WinStudio" project. In simple terms, tags are variables used by "WinStudio" to receive and store data obtained from communication with plant floor devices, from the results of calculations and functions, and from user input. Additionally, tags can be used to display information on screens (and Web pages), to manipulate screen objects, and to control Runtime Tasks on page 391.

However, tags are more than simple variables. "WinStudio" includes a real-time database manager that provides a number of sophisticated functions such as time-stamping of any value change, checking tag values against run-time minimum and maximum values, comparing tag values to alarm limits, and so on. "WinStudio" tags have both, a value and various properties that can be accessed – some during development and others only during runtime.

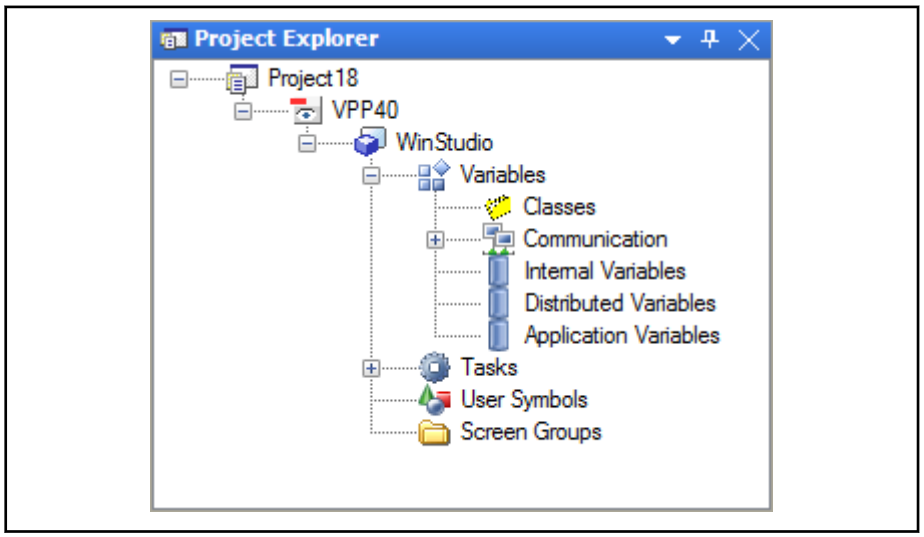


Fig. 7-1: WinStudio tags in the Project Explorer

All WinStudio tags are assigned to one of the following categories that are respectively illustrated as elements in the "Variables" folder:

- Application variables on page 104: are tags created during project development. For example, application variables are used at the following positions:
  - Screen tags
  - Tags reading values of devices or writing into a device
  - Control tags
  - Auxiliary tags used to perform mathematical calculations
- Distributed variables (Shared database folder) on page 107: Tags are created in a PC-based control program and then imported into the "WinStudio" Tags database.

  Shared tags cannot be modified within "WinStudio" – the tags have to be modified in the original PC-based control program and then re-imported into the Tags database.

- Internal Variables (System tags folder) on page 108: are predefined tags with defined functions that are used for "WinStudio" supervisory tasks. Example:
  - Date tags hold the current date in string format
  - Time tags hold the current time in string format

  Most internal variables are "read-only". This means that these tags cannot be added to, edited or removed from the database.

  To display a list of internal variables, open the "Internal Variables" editor in the "Variables" folder. The preceding figure illustrates a partial list of internal variables.

After creating a tag, it can be used anywhere within the project. The same tag can be used for more than one object or attribute.

7.1.2 Application variables

Application variables, general information

The "Application Variables" editor displays all created tags of the current project. Application Variables can be applied to display values, to read and write at external devices, to control and to perform mathematical calculations, etc.



Prior to deleting a tag, it is highly recommended to use the **Cross reference search** tool to ensure that the tag is not used at a different position in the project (screens, math worksheets, etc.). If a tag is deleted from the project database that is being used in another part of the project, this causes a compiling error and the project to function improperly.

To create a new tag, right-click on the "Application Variables" element and select **Add variable** from the context menu.

The "New Tag" dialog is displayed, as illustrated in the following figure:

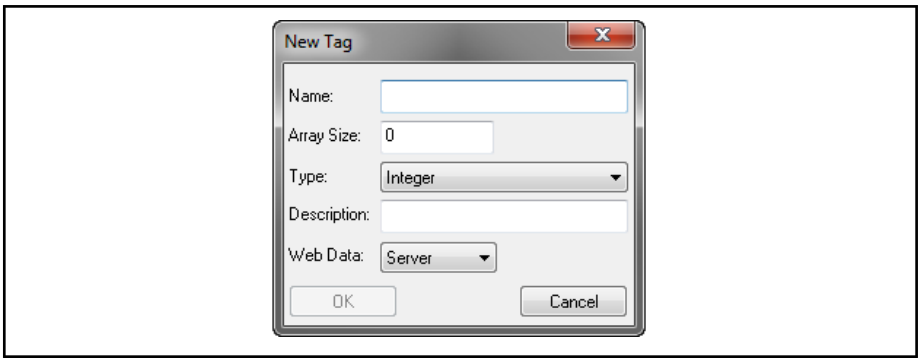


Fig. 7-2: New Tag dialog

Use this dialog to specify the following parameters.

- **Name**: Enter a name for the new tag. The first character has to be a letter and the name can contain up to 255 characters.
- **Array Size**: Enter a value to specify the size of the tag. Any size greater than 0 implies that the tag is an Array on page 112.
- **Type**: Select a standard Tag Type on page 111 from the list (**Boolean**, **Integer**, **Real** or **String**). Also new types, i.e., Classes on page 106 can be defined from standard types.
- **Description**: Enter a tag description for documentation purposes.
- **Scope**: Click to select one of the following options:

- **Local**: The information in this tag is not to be shared over the Internet.
- **Server**: The information in this tag is to be shared over the Internet.

These options have no effect on projects that do not have Web capabilities. If the **Scope** option for a project with Web capabilities is selected, any object property using "Local" will not work properly over the Web.

Tag names have to be unique. Tags that use the name of an existing tag cannot be created.

Tag properties can be displayed or edited in on of the following dialogs:

- "Tag Properties" dialog: Open the "Cross References" tool window, enter the tag name and press the **Tag Properties** button.
- "Application Variables" editor: In the **Variables** folder, click on the **Application Variables** element.

The "Application Variables" datasheet contains five columns (refer to the following sample dialog).

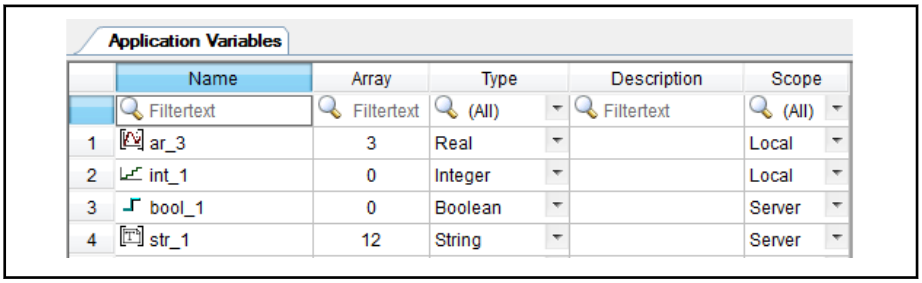


Fig. 7-3: Application Variables editor

With this editor, tag properties can be created, modified or deleted. Any tag and its properties can be used by means of the standard Windows commands cut (<Ctrl>+<X>), copy (<Ctrl>+<C>) or paste (<Ctrl>+<V>). It is also possible to undo the last modification to a field (<Ctrl>+<Z>).

Right-click and select the respective option from the context menu to sort data on the "Application Variables" sheet and/or insert/remove additional columns to/from the sheet.

Changes in application variables are immediately effective. It is not required to save or close the editor to apply the changes.

Extending the Application Variables Datasheet

The Application Variables worksheet can be extended to up to 65,488 rows, if necessary.

The datasheet is normally limited to a maximum of 32,721 rows. (This is separate from the maximum size of the project database as a whole, as well as the runtime limit that is set when selecting a target platform for a new project.)

To extend the worksheet, edit the project file (**project_name.app**) to include the following entry:

```
[Options]
EnableExtendedTagCount=1
```

However, this brings the following restrictions:

- Project tags in rows `32,722-65,488` of the worksheet cannot be used as indices in expressions. I.e., an expression like `Abs(numArray[indexTag]), indexTag` cannot be included in these lines. (This restriction does not apply to the VBScript interface.)
- In a Class worksheet, only the first 32 class members can have alarms. For any class members after the first 32, alarms do not work.

Generally speaking, extending the "Application Variables" datasheet stretches the capabilities of "WinStudio" and should be done only if it is absolutely necessary.

7.1.3 Classes folder

The "Classes" folder contains all of the project "Classes" and their respective class elements. Classes are compound structures consisting of the default data Tag Types on page 111 ("Integer", "Real", "Boolean" and "String"). Classes allow for high-level data encapsulation in the project database. A class-type tag provides a set of values for its class elements.

To define a class, "class elements" and their types have to be defined. Class elements are variables that hold values for an object with particular characteristics. Thus, defining a class can be very useful for projects with a repeatedly occurring group of variables.



Class variables are displayed in the **Application Variables** editor in the Project Tags on page 104 folder with the class symbol .

To access the elements of a class, use the following syntax with a period (".") as separator: `TagName.MemberName`. For example: `tk.LEV` or `tk.TMP`.

If the `tk` tag is an Array on page 112, the following syntax is to be used: `ArrayTagName[ArrayIndex].MemberName`

For example: `tk[1].LEV` or `tk[n].TMP`

A class tag is a set of values (instead of a single value) associated with the class. Class tags are created by grouping simple tags, which become the class elements. The maximum number of class elements for any class depends on the product specification. Class elements can include the standard types "Integer", "Real", "Boolean" and "String" as mentioned previously.

To create a new class, open the context menu of the "Class" folder and select the "Insert Class" entry.

When the "Insert Class" dialog is displayed, enter a class name in the **Name** field and click **OK** to close the dialog.

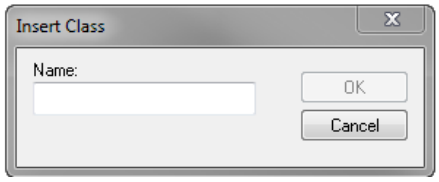


Fig. 7-4: Insert Class dialog



The class name has to be unique, i.e., it is not possible to create two classes with the same name.

"Distributed Variables" and "Internal Variables" cannot be configured as classes.

By opening the datasheet of a "Class" in the editor, class elements can be created, changed or deleted.

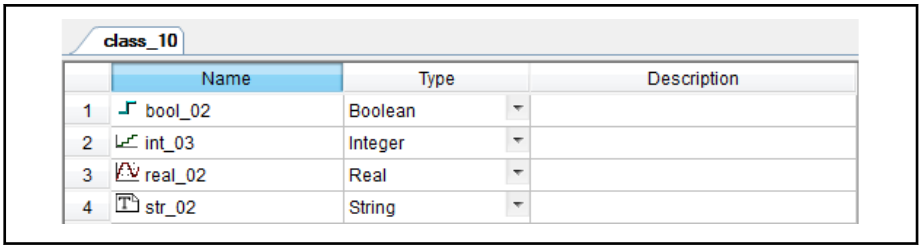


Fig. 7-5: Class datasheet

The maximum number of elements for each class is 512 as long as it does not exceed the maximum number of tags supported by the license (Product Type) selected for the project.

If a tag is created from a class type, each element from the class counts as one tag for licensing restrictions, because each class element has an independent value.

To edit a class element or property, right-click on the item and use the standard Windows commands to cut (<Ctrl>+<X>), copy (<Ctrl>+<C>), or paste (<Ctrl>+<V>). It is also possible to undo the last modification to a field (<Ctrl>+<Z>).

To edit class elements proceed as follows:

- **Name**: Enter a name for the class element or the element property. The first character has to be a letter and the name can contain up to 255 characters.
- **Type**: Select a data type ("Boolean", "Integer", "Real" or "String").
- **Description**: Enter a description of the class element for documentation purposes.

- Elements of a class cannot be of another class type.
- The class name has to be unique, i.e., it is not possible to re-use the name of an existing class.
- However: Elements with the same name can be created in different classes.

To delete a class and all its class elements, right-click on a class folder and select delete.

"WinStudio" disables the "Delete" option if any runtime tasks are executed.

Classes cannot be deleted as long as there is one variable of the class to be deleted.

7.1.4 Distributed variables (Shared database folder)

The "Distributed Variables" folder contains all tags shared between "WinStudio" and the selected PC-based Control device. These tags have to be created and edited in the PC-based Control software and are then imported automatically into "WinStudio" under the following conditions:

- When starting "WinStudio" or
- When right-clicking on the "Shared Tags" folder to refresh (update) the database.

Each PC-based Control software package has its own individual interface characteristics and conditions that allow "WinStudio" to import its tags. For example, the PC-based Control software application might have to be running to import database tags into "WinStudio".

If there are no PC-based Control software products associated with the application, the "Datasheet View" and "Tag List" subfolders (located in the "Distributed Variables" folder) are empty.

Distributed Variables cannot be edited in the "WinStudio" environment, however, they can be modified by means of the PC-based Control software.

However, Distributed Variables can be edited in any "WinStudio" task like any other tag.

Distributed Variables are read-only and viewable on the "Tag Properties" dialog and the "Distributed Variables" datasheet.

Right-click on the "Distributed Variables" folder (or click **Datasheet View**) and select the **Refresh** option to update the last "Version" of the PC-based Control software's tags database.

To edit the Shared Tags database (create a new tag, delete tags or edit tag properties) this command to update the "WinStudio" shared database has to be activated.

To view the "Tag Properties" dialog, proceed as follows:

- Click on the **Tag Properties** button in the "Cross References" tool window (the tag name has to be in the **Tag Name** field) or
- Double-click on the tag name in the Project Tags on page 104 editor

If the "Distributed Variables" datasheet is displayed, it contains four columns ("Name", "Size", "Type" and "Description"). This datasheet is read-only and can only be used to view distributed variables.

7.1.5 Internal Variables (System tags folder)

The "Internal Variables" folder contains predefined tags with specific functions (time, date, acknowledge alarms, storage of the logged user, etc.). These tags cannot be edited or deleted, however, their values can be accessed from any "WinStudio" task, copied and used elsewhere.

The internal variables database is updated by right-clicking on the "Internal Variables" element. For a list of the available internal variables including their properties and descriptions, refer to List of system tags on page 949:

The properties of internal variables can be displayed using the "Internal Variables" datasheet that contains four columns ("Name", "Size", "Type" and "Description").

Most internal variables are read-only.

For example, to change the time, the mathematical function SetSystemTime on page 658 has to be used instead of writing to the internal variable.

7.2 Creating tags

7.2.1 Tag Name Syntax

When naming tags, the following guidelines have to be observed:

- Tag names **have to be unique**, i.e., same name cannot be specified for two different tags (or functions).

  If an existing tag name is entered, "WinStudio" recognizes that the name already exists and does not create the new tag.
- Each tag name has to begin with a "letter". Additionally, letters, numbers and the underscore character ("_") can be used in tag names.
- The following symbols **cannot** be used tag names:

  ` ~ ! @ # $ % ^ & * ( ) - = / + \ [ ] { } < > ?
- Tag names or class elements may not exceed 255 characters.

  Upper and lower case letters can be used. Tag names are **not** case sensitive.



As "WinStudio" does not differentiate between upper case and lower case characters, both can be used to make tag names more readable.

"TankLevel" instead of "tanklevel".

- Tag names have to differ from internal variables and math functions.



Use the `@` character at the beginning of a tag name to indicate that the tag is used in the project as an Indirect Tag on page 117.

Some valid tag examples include:

- Temperature
- pressure1
- Count
- x

7.2.2 Choosing the tag type

"WinStudio" allows to create the following types of tags:

- **Basic tags** hold a single value
- **Array tags** are a set of tags that use the same name with unique indices
- **Class tags** are a set of compound tags that consist of user-defined data types (Boolean, Integer, Real or String) or data-type structures
- **Indirect tags** are pointers that provide indirect access to another tag type, including class tags

The following section describes these tag types in detail.

Basic tags

"Basic" tags receive single values. Typically, most tags defined for a project are basic tags.

Some examples of a basic tag include:

- `TankID` (to identify different tanks in the project)
- `Temperature` (to identify the current temperature of an object)
- `Status` (to identify whether an object is open or closed)

Array tags

An "array" tag consists of a set of tags that all have the same name but use unique indices (a matrix of "n" lines and one column) to differentiate between each tag. An "Array index" can be a fixed value, another tag or an expression. Maximum array sizes are determined by product specifications.

Array tags can be used for the following purposes:

- Simple configurations
- Enable multiplexing in screens, recipes and communication interfaces
- Saving development time during tag declaration

Array tags can be specified in two formats:

- For a simple array tag, enter:

  `ArrayTagName[ArrayIndex]`
- For a complex array tag (where the array index is an expression consisting of a tag and an arithmetic operation), enter:

  `ArrayTagName[ArrayIndex + c]`

  The following apply here:

  – `ArrayTagName` is the tag name
  – `[ArrayIndex]` is the unique index (fixed value or another tag)
  – "+" is an arithmetic operation; and
  – **c** is a numerical constant



- For each array tag, a maximum index has to be specified by entering a value ("n") in the Array size column of the "Application variables" datasheet or in the Array size field on a "New variable" dialog (see Application variables on page 104).

  When creating an Array Tag with **n** positions, "WinStudio" actually creates **n+1** positions (from 0 to "n"). For example, if `ArrayTag[15]` is specified, the array has 16 elements, where 0 is the start position and 15 is the end position.
- Array tags may not contain any spaces.

  If "WinStudio" reads a tag, it begins with the first character and continues until it finds the first space or zero. This way, the system does not recognize any characters following the space as part of the array tag.

  For example, if `a[second + 1]` is entered, "WinStudio" recognizes `a[second` as the tag and considers it invalid because "WinStudio" does not find (recognize) the closing bracket.

  However, `a[second+1]` is recognized as a valid array tag.

Array tags can be specified wherever a variable name can be used. Also, as array tags greatly simplify configuration tasks and can save development time, it is recommended using them whenever possible.

As an example, temperature monitoring of four tanks is observed. The conventional configuration method is:

- temperature1 – high temperature on tank 1
- temperature2 – high temperature on tank 2
- temperature3 – high temperature on tank 3

- temperature4 – high temperature on tank 4

Array tags can be used to simplify this task as follows (where **[n]** represents the tank number):

- Temperature[n] – high temperature on tank "n"

The following table provides some additional examples of array tags:

Array tag examples

Array tag examples	Description
`Tank[1], Tank[2], Tank[500]`	Simple arrays, where array indices ("1", "2" and "500") are numerical constants. For example, tank numbers.
`Tank[tk]`	A simple array, where the array index ("tk") is a tag. For example, a tag representing the tank number.
`Tank[tk+1]`	An array, where the index (`tk+1`) is an expression. For example, the value of `tk` (tank number) plus 1.



If using another tag to reference the index of an array and if the value of the tag is outside the size of the array, the following results are given:

- If `IndexTag` is greater than the size of the array, `MyArray[IndexTag]` points to the end position of the array; and
- If `IndexTag` is less than 0, then `MyArray[IndexTag]` points to the start position of the array (i.e., `MyArray[0]`).

Indirect tags

Indirect tags "point" to other database tags (including class-type tags). Using indirect tags can save development time as no duplicate tags (and the logic built into them) have to be created.

Indirect tags can be created from any String-type tag simply by typing the "@" symbol in front of the tag name, i.e. `@TagName`.

- To reference a simple tag, it is assumed that the `strX` tag, (a String tag) contains the value "Tank", i.e. the name of a different tag. In this case, reading or writing to `@strX` provides access to the tag value `Tank`.
- To reference a class-type tag and member, simply create a string tag that points to the class tag and the member.

  For example, if a tag `strX` (a string tag) holds the value "Tank.Level", which is the name of the class tag, then reading from or writing to `@strX` provides access to the tag value "Tank.Level".
- It can also be pointed directly to a class-type tag member by identifying a class-type that points to a class member.

  Example: To access the `Tank.Level` member of the class, the "Tank" value has to be stored within the `strX` tag and the `@strX.Level` syntax has to be used.

7.2.3 Choosing the Tag Data Type

Another consideration when designing a tag is what type of data the tag receives. "WinStudio" recognizes the following, standard tag "data types":

- **Boolean** ("one bit"): Simple Boolean value with the possible values of 0 (false) and 1 (true). Equivalent to the "bool" data type in C++. Typically used for turning objects off and on or for closing and opening objects.
- **Integer** ("four bytes"): Integer number (positive, negative or zero) internally stored as a signed 32-bit.

Equivalent to the "signed long int" data type in C++. Typically used for counting whole numbers or setting whole number values.

Examples: `0, 5, -200`.

- **Real** ("floating point, eight bytes"): Real number that is stored internally as a signed 64-bit.

  Equivalent to the "double" data type in C++. Typically used for measurements or for decimal or fractional values.

- **String** ("alphanumeric data, up to 1024 characters"): Character string up to 1024 characters that holds letters, numbers or special characters.

  Supports both ASCII and UNICODE characters.

  Examples: `Recipe product X123, 01/01/90, *** On ***`.

Tags can also be turned into a compound tag by assigning it a Class on page 115.

A Class is a template consisting of two or more tag definitions, each with its own data type.

Classes can be used in projects that have items (e.g., tanks of liquid) with multiple attributes (e.g., fill level, temperature, pressure) to be monitored or controlled.

For further information, refer to : Tag properties on page 122

Changing how Boolean tags receive numeric values

When writing any numeric value not equal 0 (i.e., ≠0) to a Boolean tag, the tag assumes a default value of 1.

If required, this behavior can be changed by editing the file `project_name.app` by changing the following settings:

```
[Options]
BooleanTrueAboveZero=value
```

If **BooleanTrueAboveZero** is set to the default value of "0", the project behaves as described above.

If **BooleanTrueAboveZero** is set to "1", the project behaves as follows:

- When writing any numeric value less than or equal to 0 (i.e., ≤0) to a Boolean tag, the tag assumes a value of 0 (false).
- When writing any numeric value greater than 0 (i.e., >0) to a Boolean tag, the tag assumes a value of 1 (true).

NOTICE **This is a global runtime setting.**

If it is only to be changed how certain tags are handled, this setting is not to be changed.

7.2.4 Using Array Tags

"WinStudio" tags can consist of a single value or an array of values.

The maximum array size is 16384 as long as it does not exceed the maximum number of tags supported by the license (Product Type) selected for the project.

Each array position (including the position 0) counts as one tag for licensing restrictions, because each position has an independent value.

An array tag is a set of tags with the same name that is identified by indices (a matrix of **n** lines and 1 column). The maximum array size depends on the product specification.

The following syntax can be used to access an array tag:

`ArrayTagName[ArrayIndex]`

Example: `tank[0]`, `tank[1]`, `tank[2]` and `tank[500]`.

A maximum index for each array tag has to be specified in the **size** column of each datasheet.

n can be specified to indicate the array tag has positions from 0 to **n**.

For example, if the size of TagA is 3, the tag elements could be `TagA[0]`, `TagA[1]`, `TagA[2]` and `TagA[3]`.



Use the array tag whenever possible as it optimizes memory use and simplifies the configuration task.

For example, if a display is required to monitor each tank, array tags can be used to configure a single display containing tags linked to all tanks.

For example (using the `tk` tag as an index containing the number of the tank): `pressure[tk]`, `temperature[tk]` and `temperature[tk+1]`.

An array tag can be a tag, a numeric value or an expression with the arithmetic operator "+".

When referring to an array with an index using the "+" arithmetic operation, the following syntax has to be used:

`ArrayTagName[NumValue1+NumValue2]`

Where `NumValue1` and `NumValue2` can be an integer tag or a numerical constant.

Example: `temperature[tk+2]` or `temperature[tk+6]`.

Using array tags in any "WinStudio" task can save a significant amount of project development time.

For example, if tag points related to the temperature of four tanks are required. The conventional configuration method is the following:

- `temperature1`: High temperature on tank 1
- `temperature2`: High temperature on tank 2
- `temperature3`: High temperature on tank 3
- `temperature4`: High temperature on tank 4

Using array tags simplifies this task, as follows:

- `Temperatur[j]`: High temperature on tank {j}

If a four-position array tag is created, the system creates five positions (from 0 to 4).

Example:

```
tag_example[15] //start position=0, end position=15
```

Therefore, the `tag_example[15]` array has 16 elements.

If using another tag to reference the index of an array and if the value of the tag is outside the limits of the array, the following results are given:

- If `IndexTag` is greater than the size of the array, `MyArray[IndexTag]` points to the end position of the array; and
- If `IndexTag` is less than 0, `MyArray[IndexTag]` points to the start position of the array.

Array tags

An "array" tag consists of a set of tags that all have the same name but use unique indices (a matrix of **n** lines and one column) to differentiate between each tag. An "Array index" can be a fixed value, another tag or an expression. Maximum array sizes are determined by product specifications.

Array tags can be used for the following purposes:

- Simple configurations
- Enable multiplexing in screens, recipes and communication interfaces
- Saving development time during tag declaration

Array tags can be specified in one of two formats:

- For a simple array tag, enter:

  `ArrayTagName[ArrayIndex]`

- For a complex array tag (where the array index is an expression consisting of a tag and an arithmetic operation), enter:

  `ArrayTagName[ArrayIndex + c]`

  The following apply here:

  – `ArrayTagName` is the tag name
  – `[ArrayIndex]` is the unique index (fixed value or another tag)
  – "**+**" is an arithmetic operation; and
  – **c** is a numerical constant

- For each array tag, a maximum index has to be specified by entering a value (**n**) in the Array Size column of the "Application Variables" datasheet or in the Array Size field on a "New Tag" dialog.

  (See "Creating project database Tags on page 104").

  When creating an Array Tag with **n** positions, "WinStudio" actually creates **n+1** positions (from **0** to **n**).

  For example, if `ArrayTag[15]` is specified, the array has 16 elements, where 0 is the start position and 15 is the end position.

- Array tags may not contain any spaces. If "WinStudio" reads a tag, it begins with the first character and continues until it finds the first space or zero. Consequently, the system does not recognize any characters following the space as part of the array tag

  For example, if `a[second + 1]` is entered, "WinStudio" recognizes `a[second` as the tag and considers it invalid because "WinStudio" does not find (recognize) the closing bracket.

  However, if `a[second+1]` is entered, it is recognized as a valid array tag

Array tags can be specified everywhere, where a variable name would be used. As array tags greatly simplify configuration tasks and can save development time, it is recommended using them whenever possible.

As an example, temperature monitoring of four tanks is observed. The conventional configuration method is:

- `temperature1` – high temperature on tank 1
- `temperature2` – high temperature on tank 2
- `temperature3` – high temperature on tank 3
- `temperature4` – high temperature on tank 4

Array tags can be used to simplify this application task as follows (where **[n]** represents the tank number):

- `Temperature[n]` – high temperature on tank **[n]**

The following table provides some additional examples of array tags:

Array tag examples

Array tag examples	Description
Tank[1], Tank[2], Tank[500]	Simple arrays, where the indexes (1, 2, and 500) are numerical constants. For example, tank numbers.
Tank[tk]	A simple array, where the array index (tk) is a tag, e.g. a tag representing a tank number.
Tank[tk+1]	Compound, at which the Array index (tk+1) is an expression, e.g., the value of tk (tank number) plus 1.

If using another tag to reference the index of an array and if the value of the tag is outside the limits of the array, the following results are given:

- If `IndexTag` is greater than the size of the array, `MyArray[IndexTag]` points to the end position of the array; and
- If `IndexTag` is less than 0, then `MyArray[IndexTag]` points to the start position of the array (i.e., `MyArray[0]`).

7.2.5 Using classes

"Class" tags are compound tags that permit a high-degree of encapsulation within the Tags database. Where basic tags receive a single value, "classes" are designed to receive multiple values.

Class tags can be created by grouping basic or array tags, which then become class "members". The maximum number of members for any class depends on the product specification.

Class tags can be specified in two formats:

- For a simple class tag the syntax is

  `TagName.ClassMemberName`

  (Where the period is used as a separator.)

  For example, several different conditions are to be monitored (such as "temperature", "level" and "pressure") in a tank, a class tag can be created as follows:

  – `Tank.Temperature`
  – `Tank.Level`
  – `Tank.Pressure`

- For creating a compound class tag (using an array tag) the syntax is

  `ArrayTagName[ArrayIndex].ClassMemberName`

  (again, the period is used as a separator.)

  If "temperature", "level" and "pressure" conditions are to be monitored in multiple tanks, a class tag can be created as follows:

  – `Tank[tk].Temperature`
  – `Tank[tk].Level`
  – `Tank[tk].Pressure`

  Where "tk" is an array index representing the tank number.

Classes folder

The **Classes** folder includes all project classes.

Classes are compound tags consisting of user-defined data-type structures or Tag Types on page 111 ("Integer", "Real", "Boolean" and "String").

Classes allow for high-level data encapsulation in the project database.

A class-type tag provides a set of values for its class elements.

To define a class, "class elements" and their types have to be defined.

Class elements are variables that hold values for an object with particular characteristics. Thus, defining a class can be very useful for projects with a repeatedly occurring group of variables.

Classes in the "Classes" folder as well as tags of the "Class" data type in the folder, Application Variables on page 104are indicated by the symbol .

To access the elements of a class, use the following syntax with a period (".") as separator:

```
TagName.MemberName
```

For example: `tk.LEV` or `tk.TMP`.

If the "tk" tag is an Array on page 112, the following syntax is to be used:

Program:

```
ArrayTagName[ArrayIndex].MemberName
```

For example: `tk[1].LEV` or `tk[n].TMP`

A class-type tag contains a set of values (rather than a single value) associated with the class.

Class tags are created by grouping simple tags, which become the class elements. The maximum number of elements for any class depends on the product specification. Class elements can include the standard values "Integer", "Real", "Boolean" and "String" as mentioned previously.

To create a new class, use one of the following methods to open the "Insert Class" dialog:

- Select the "New Class" entry in the context menu of the "classes" folder; or
- Create a new class tag in the Project Tags on page 104 folder.

If the "Insert Class" dialog displays, enter a class name in the **Name** field and then click **OK** to close the dialog.

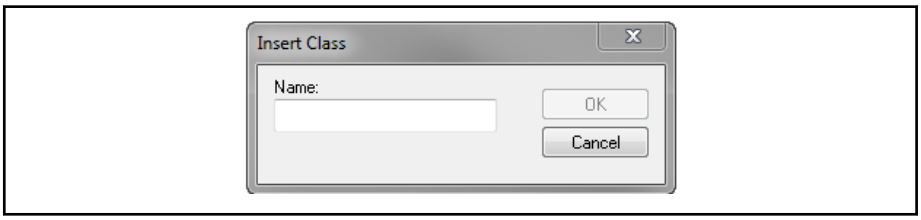


Fig. 7-6: Insert Class dialog

A unique class name has to be entered. It is not possible to create two classes with the same name. Additionally, shared tags and system tags cannot be configured as classes.

If the "Class" datasheet is displayed, it can be used to create, modify or delete any class elements and their viewable properties. (Classes cannot be edited from the "Tag Properties" dialog.)

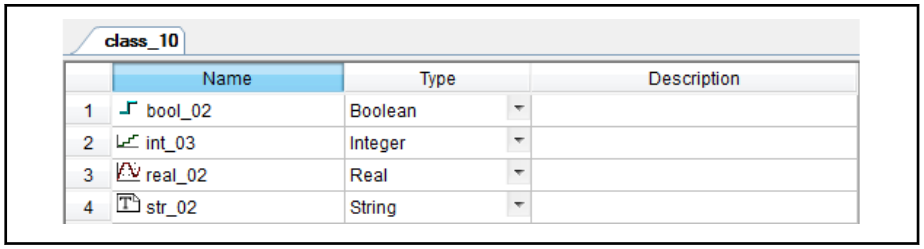


Fig. 7-7: Class datasheet

The maximum number of class elements for each class is 512 as long as it does not exceed the maximum number of tags supported by the license (Product Type) selected for the project. If a tag is created from a class type, each element from the class counts as one tag for licensing restrictions, because each class element has an independent value.

To edit a class element or property, right-click on the item and use the standard Windows commands to cut (<Ctrl>+<X>), copy (<Ctrl>+<C>), or paste (<Ctrl>+<V>). It is also possible to undo the most recent modifications to a field by using <Ctrl>+<Z>.

To edit element properties, proceed as follows:

- **Name** field: Enter a name for the class element or the element property. The first character has to be a letter and the name can contain up to 255 characters
- **Type** combo box: Select a data type ("Boolean", "Integer", "Real" or "String")
- **Description** field: Enter a description of the class element for documentation purposes

Elements of a class cannot be of another class type. Also, a unique class name has to be created. The name of an existing class cannot be reused. However, elements with the same name can be created in different classes.

To delete a class and all its elements, right-click on a class folder and select delete.

"WinStudio" disables the delete option if any runtime tasks are running.

In addition, classes cannot be deleted if they are associated with any tag.

☞ Changes in classes are immediately effective. It is not required to save or close the class editor to apply the changes.

7.2.6 Using Indirect Tags (Pointers)

"WinStudio" supports indirect access to tags in the database.

Example: a tag `X` of the String type.

This tag can hold the name of any other tag in the database (i.e., it can provide a pointer to any other type of tag, including a class type).

Syntax for an indirect tag:

`@IndirectTagName`

A tag named `X` holds a string "TEMP".

Reading and/or writing to `@X` provides access to the value of the "TEMP" variable.

Any tag created as a string-type tag is potentially an indirect tag (pointer).

To refer to a class-type tag, a String-type tag can be referred to that points to a class tag.

Class	`TANK` with element `Level`
Tag	`TK` of the class `TANK`
Tag	`XCLASS` of the String type

To access the `TK.Level` value, the `TK.Level` value has to be stored within the `XCLASS` tag and the `@XCLASS` syntax has to be used.

It is also possible to refer directly to an element of a class-type tag, identifying a class-type that points to a class element.

For example:

Class	`TANK` with element `Level`
Tag	`TK` of the class `TANK`
Tag	`XCLASS` of the class `TANK`

To access the `TK.Level` value, the `TK` value has to be stored within the `XCLASS` tag and the `@XCLASS.Level` syntax has to be used.

When creating tags for indirect use, place an `X` in the tag column rather than creating them as Strings. For type, enter the type of tag a reference is created for.

Follow the `XCLASS` example: `@Z Integer`, `@X Class:TANK`.

Indirect tags

Indirect tags "point" to other database tags.



Using indirect tags can save development time as no duplicate tags (and the logic built into them) have to be created.

Indirect tags can be created from any String-type tag simply by typing the `@` symbol in front of the tag name, i.e. `@TagName`.

- To reference a simple tag, assume the `strX` tag (a String tag) holds the value "Tank", which is the name of another tag, then reading from or writing to `@strX` provides access to the value of the `Tank` tag.
- To reference a class-type tag and element, simply create a String tag that points to the class tag and the element.

  For example, if a tag `strX` (a string tag) holds the value `"Tank.Level"`, which is the name of the class tag, then reading from or writing to `@strX` provides access to the value of the `Tank.Level` member.
- It is also possible to refer directly to an element of a class-type tag, identifying a class-type that points to a class element.

  For example: to access the `Tank.Level` element of the class, the "Tank" value has to be stored within the `strX` tag and the `@strX.Level` syntax has to be used.

7.2.7 Choosing the Tag Scope

"WinStudio" allows to decide whether a tag creates only one instance on a project server or copies on each local station.

- **Server** (default): The tag is created on the project server and shared by all connected clients (e.g., Thin Client, Secure Viewer). A change to the tag value affects the entire project.

- **Local**: A virtual copy of the tag is maintained separately on each local station (server + clients), a change to the tag value affects only the station on which the change was made.

7.3 Creating database tags

7.3.1 Creating tags on the datasheet

To create tags on the "Application Variables" datasheet, proceed as follows:

1. Open the **Variables** folder
2. Open the "Application Variables" datasheet by double-clicking on the respective element in the Project Explorer:

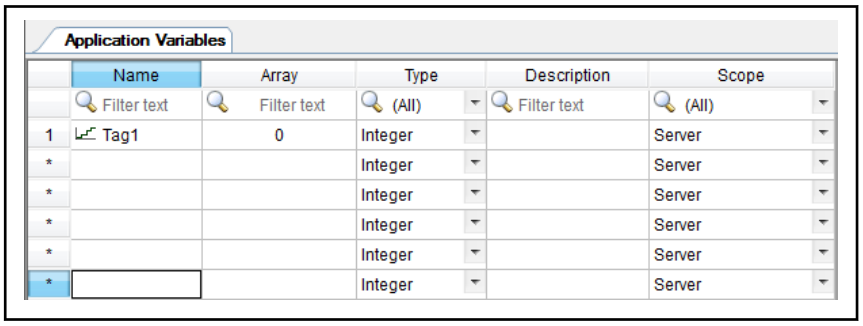


Fig. 7-8: Application Variables datasheet

3. Locate an empty line in the datasheet and configure the following fields.



The Tab key on the keyboard can be used to move to the next column.

- **Name** column: Enter a name using the proper syntax.

  (For more information, refer to:Tag Name Syntax on page 109)
- **Array size** column:
  - For an array tag, enter a value to specify the maximum index of the array.
  - For any other tag type, enter zero (0).
- **Type** combo box: Click the arrow to select a tag data type (Boolean, Integer, Real, or String) from the list.

  (For further information, refer to Choosing a Tag Data Type on page 111)
- **Description** column (optional): Enter a description for documentation purposes only.
- **Scope** combo box: Click the arrow to specify whether the tag value is shared with (displayed on) Thin Client stations.

  (For more information, refer to:Choosing the Tag Scope on page 118.)

4. Click into a new line to create further tags.

   If no more tags are to be created, close the "Application Variables" datasheet.

The following example illustrates a variety of tags configured in an "Application Variables" datasheet.

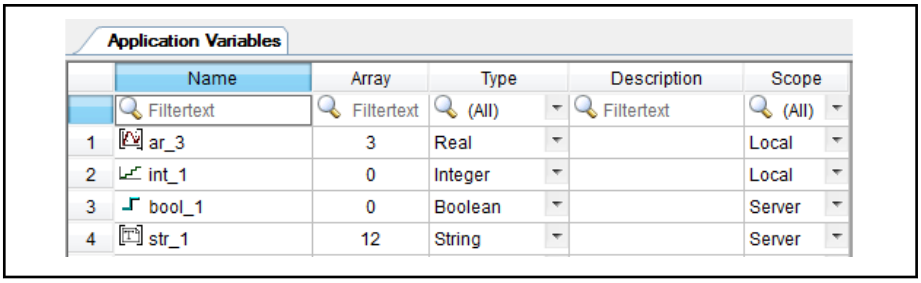


Fig. 7-9: Example for datasheet Application Variables

7.3.2 Creating tags ""on-the-fly""

Instead of opening the Application Variables datasheet, every time a new tag is to be created, individual tags can be created "on-the-fly" by performing any of the following actions:

- In the "Project Explorer", in the context menu of the "Application Variables" element, select the "Insert Variables" entry.
- Enter a new tag name into any **Tag/Expression** text field (available from "Object Properties" dialog, worksheets, etc.). If the "Question" dialog asks whether a new tag is to be created, click **Yes** .

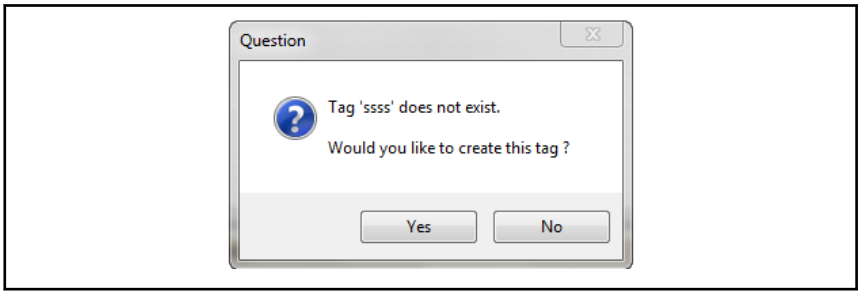


Fig. 7-10: Creating a new tag

Any of these actions causes a "New Tag" dialog to be displayed that can then be completed as needed.

(For more information, refer to: Application variables, general information on page 104.)

7.3.3 Editing Tags

The properties of a tag can be changed at any time during development or runtime.

Changes in the Application Variables datasheet

Use the following steps to edit tags in the "Application Variables" datasheet:

1. Open the "Application Variables" datasheet and look for the desired tag.
2. Double-click in the column containing the information to be changed, and enter the new information into the datasheet.
3. After editing, close the datasheet. Changes are applied immediately into the tags database.

	3.1	Right-click and select the respective option from the context menu to sort data on the "Application Variables" sheet and/or insert/remove additional columns to/from the sheet
	3.2	Any tag and its properties can be used by means of the standard Windows commands cut (<Ctrl>+<X>), copy (<Ctrl>+<C>) or paste (<Ctrl>+<V>)
	3.3	It is also possible to undo the last modification to a field (<Ctrl>+<Z>)

7.4 Creating classes

To create a new class tag, proceed as follows:

1. Open the context menu of the "Classes" element and select the **Insert Class** entry
2. If the "Insert Class" dialog is displayed, enter a **Name** into the respective text field

   (For further information, refer to Tag Name Syntax on page 109 and Choosing the tag type on page 109)

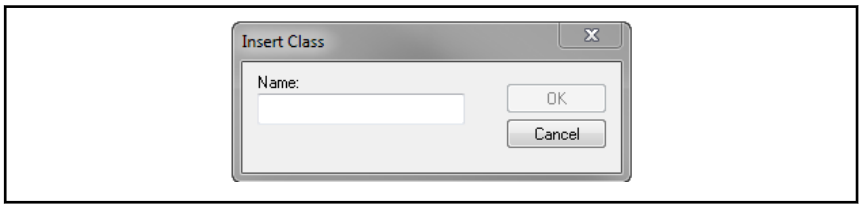


Fig. 7-11: Insert Class dialog

3. Click **OK** to close the "Insert class" dialog. A new element is created in the Project Explorer. Open this element by double-click.

   The columns in this worksheet are configured as follows:

   - **Name**: Enter a class element name.
   - **Type** selection list: Click the arrow to select a data type (Boolean, Integer, Real, or String) for the class element from the list.
   - **Description** (optional): Enter a description of the class element for documentation purposes.

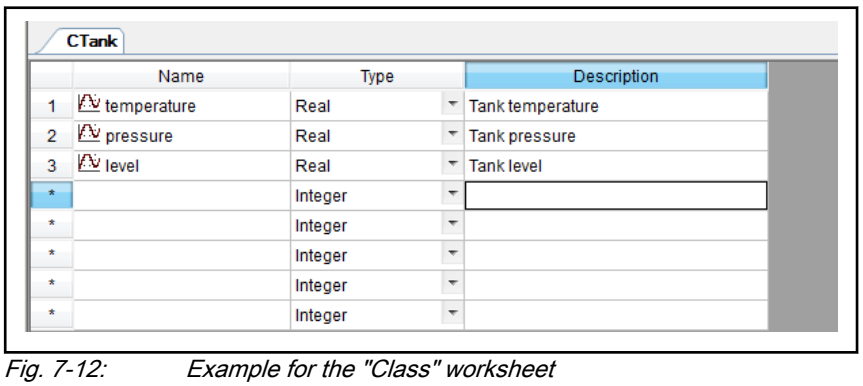


Fig. 7-12: Example for the "Class" worksheet

4. Click into the next blank line and enter the information for the next class element to be included in this class. If no more elements are to be added, the "Class" worksheet can be closed.
5. Afterwards, refer to the instructions under Creating tags on the datasheet on page 119 to create and connect a tag to a new class.

Please note that the new class name ("CTank") is included in the types list if the arrow button for display is pressed (refer to line 5 in the following figure). Select the class name from this list.

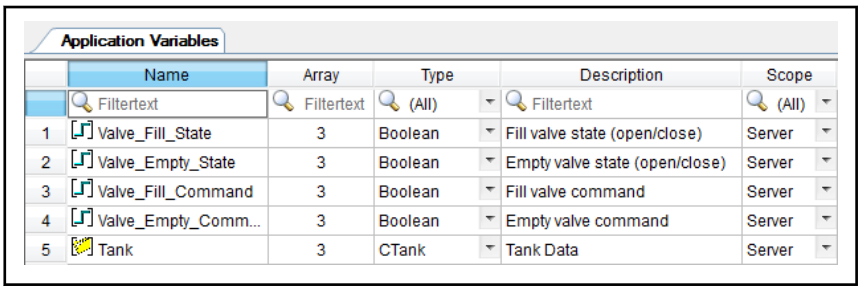


Fig. 7-13: Creating the Tank class tag

6. After finishing, save the work and close the worksheet.

7.5 Tag properties

7.5.1 Tag properties

Tag properties are meta data associated with each tag in the database.

Most project tags can be regarded as simple program variables which save values as they are typically used in this way in your project. In reality, however, each tag is a complex data structure which can be used in different ways during runtime depending on the configuration of the tag properties.

In addition to the default meta data such as array size, data type, description and web data, the tag properties are used to configure alarm conditions, get the tag quality, convert between different measuring units, access individual bits, retain values also for project restarts, save historical data and so on.

Many tag properties can be directly edited in the project tags datasheet. Each column in the datasheet (including **Name**) stands for a different tag property and while the datasheet shows the most common properties by default, it can be configured to show even more properties.

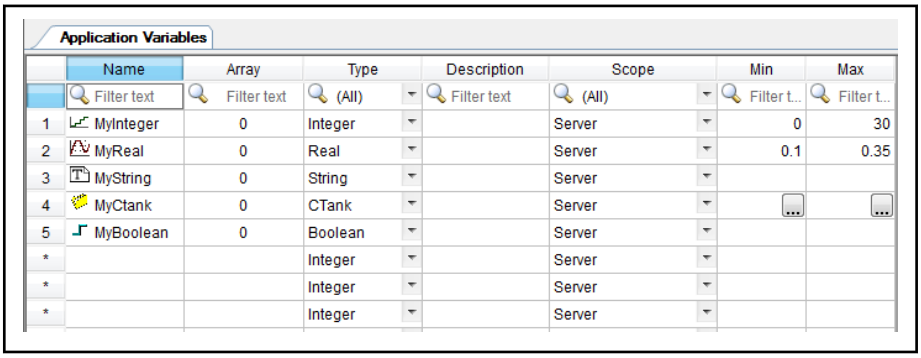


Fig. 7-14: Project Tags datasheet

Additional tag properties can be set using the Properties command which opens the Tag properties dialog box.

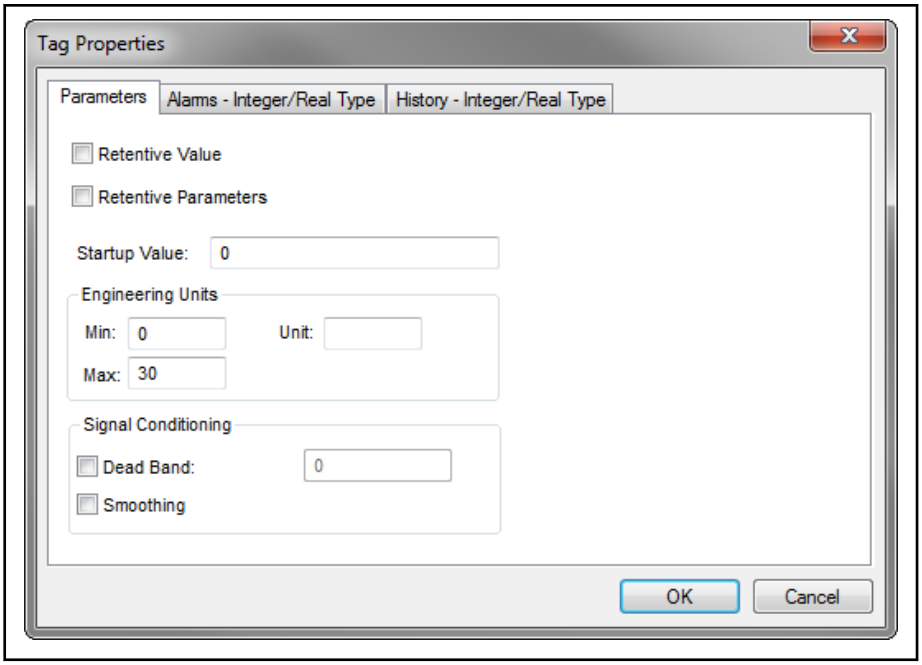


Fig. 7-15: Tag properties dialog box

During runtime, it is possible to get and set many tag properties exactly as you would get and set the values of the project tags themselves.

See also

- Setting tag properties using the project tags datasheet on page 104

7.5.2 Referencing a tag property instead of a project tag

Use a special syntax to reference a tag property as you would reference a project tag.

Everywhere in your project where a project tag would be referenced - for example, when configuring a tag/expression to control the Visibility of a screen object on page 255 - a tag property can be referenced instead. Use the following syntax for this:

Syntax

```
tagname->property
```

Examples

```
MyInteger->Quality

MyReal->HiHi

MyClass.FirstMember->MemberName

MyArray[1]->Index
```

This works everywhere in Rexroth WinStudio, also in VBScript and in function parameters, and can be used to call and determine the values of the tag properties.

If tag properties are referenced in this way they are also referred to as tag fields.

For the correct name form of properties and some property-specific limitations, refer to the Complete list of tag properties on page 136.

7.5.3 Editing the tag database during runtime

Use the tag database functions (TagsDB) to add or remove project tags, classes and class members during runtime and to specify properties and alarm conditions for project tags.

There are several important aspects which are to be considered when using TagsDB functions since they do much more than specifying and calling tag values. They change the structure of the tag database which, if not done properly, may lead to serious problems for a running project and all connected clients. Most TagsDB functions can only be executed under the following conditions.

First, the Rexroth WinStudio development application has to be installed and run on the project server. It has to be fully licensed for engineering and runtime and the project has to be opened in the application. The TagsDB functions use the database editor of the development application in the same way you do when you change your project manually during runtime. Due to this limitation, the TagsDB functions cannot be used in projects which are designed for Windows Embedded devices and were downloaded there.

In addition, the TagsDB functions can only be called by scripts which are executed on the project server. None of the connected clients — not even the project client which runs on the same computer as the project server, since it runs on this computer as a separate process — can make structural changes to the tag database without influencing other clients, reducing their runtime performance and potentially damaging the database. This is why the following applies in general:

- The functions can be called by the startup script (which is executed when the project itself is running), by script groups (which are constantly executed by the background task) and by all global procedures which are called by them; and
- the functions cannot be called by the graphics script (which is executed separately by each client), by screen scripts (which are attached to individual screens) or by command animations.



To bypass these limitations, do one of the following:

- Create a global procedure to call the TagsDB functions and then call the RunGlobalProcedureOnServer on page 711 function to execute this procedure; or
- Create a script group to call the TagsDB functions, configure a tag/expression trigger to control the execution of this script group and change the value of this tag/expression.

As a third step, first the TagsDBBeginEdit function has to be called in each script which calls TagsDB functions to make structural changes to the tag database, to block the database for editing and prevent any other runtime changes. Then the TagsDBEndEdit function must be called at the end of the script to complete the changes and to allow the database to resume the normal runtime behavior. Both functions must be called in the same script since the script (or, strictly speaking, the program thread executing the script) claims the tag database for itself while it is blocked. It is not possible to call TagsDBBeginEdit in one script and then TagsDBEndEdit in another script.

If a project is edited during runtime, the project server and each connected client has to be updated with changes as soon as they were made. This is usually no problem if the project is edited manually since the changes are performed slowly and one after the other. Using the TagsDB functions, however, many changes can be made quickly and simultaneously so that updat-

ing the project runtime server and the connected clients with all these changes while the project is running may significantly reduce the runtime performance. This is why the project runtime server — including all background tasks such as alarms, trends and other scripts — are stopped when TagsDBBeginEdit is called to maintain the performance and to protect the tag database. Subsequently, the changes are applied as a batch when TagsDBBeginEdit is called. In addition, the screens which were already open on connected clients are reopened during this updating process and the corresponding OnOpen screen scripts are executed again.



The TagsDBBeginEdit function has a permanent effect. This means that: If the function is called to lock the tag database during project runtime and then stop the project, the database remains locked and cannot be edited manually.

Restarting the project may unlock the database or not, depending on the design of your project and on the function call which locked the database in the first place. This means that while the project is stopped, the database spy should be used to manually call the TagsDBEndEdit function. If it is executed successfully, the project can be restarted securely.

Examples

The following example shows how the TagsDB functions are used in VBScript to add a new class, to then add a new class member to this class, to then add a new tag to this class and to then set an alarm and a trend on this tag:

```
If($TagsDBBeginEdit()=0) Then
     If($TagsDBAddClass("TempClass")=0) Then
          If($TagsDBAddClassMember("TempClass","TempMember","Real")=0) Then
               If($TagsDBAddTag("TempTag","TempClass",2,0)=0) Then
                    If($TagsDBSetAlarm("TempTag[0].TempMember",1,0,3.5)<>0) Then
                         $Msg = "Alarm not Set"
                    End If
                    If($TagsDBSetTrend("TempTag[0].TempMember",0,1)<>0) Then
                         $Msg = "Trend not Set"
                    End If
               Else
                    $Msg = "Tag not created"
               End If
          Else
               $Msg = "Class Member not added"
          End If
     Else
          $Msg = "Class not created"
     End If
     $TagsDBEndEdit()
Else
     $Msg = "Tag functions not enabled"
End If
```

Please note in particular how the script starts with the function TagsDBBeginEdit and then ends with the function TagsDBEndEdit. Also look at how the nested `If…Then…Else` structures ensure that each function is executed successfully (i.e. returns the value 0), before the next one is started.

Here is another example that shows how to remove the alarm, trend, tag, class member and class which were added in the previous example in reverse order:

```
If($TagsDBBeginEdit()=0) Then
     If($TagsDBRemoveAlarm("TempTag",1)<>0) Then
          $Msg = "Alarm not removed"
     End If
     If($TagsDBRemoveTrend("TempTag")<>0) Then
          $Msg = "Trend not removed"
```

```
     End If
     If($TagsDBRemoveTag("TempTag")=0) Then
          If($TagsDBRemoveClassMember("TempClass","TempMember")<>0) Then
               $Msg = "Class member not removed"
          End If
          If($TagsDBRemoveClass("TempClass")<>0) Then
               $Msg = "Class not removed"
          End If
     Else
          $Msg = "Tag not removed"
     End If
     $TagsDBEndEdit()
Else
     $Msg = "Tag functions not enabled"
End If
```

It is not necessary to remove the alarm and trend before removing the tag on which they are located since they are discarded together with the rest when the tag is removed. They are only mentioned in this example for completeness. The class member and the class, however, cannot be removed until the tag has been removed successfully.

7.5.4 Properties of Integer and Real tags

Each Integer and Real tag in the tag database has several properties (or meta data) in addition to its actual value. You can set these properties with the help of the project tags database or with the help of the functions of the tag database.

Parameter

The following list describes the general parameters of Integer and Real tags. These parameters determine how the selected tag is used in your project.

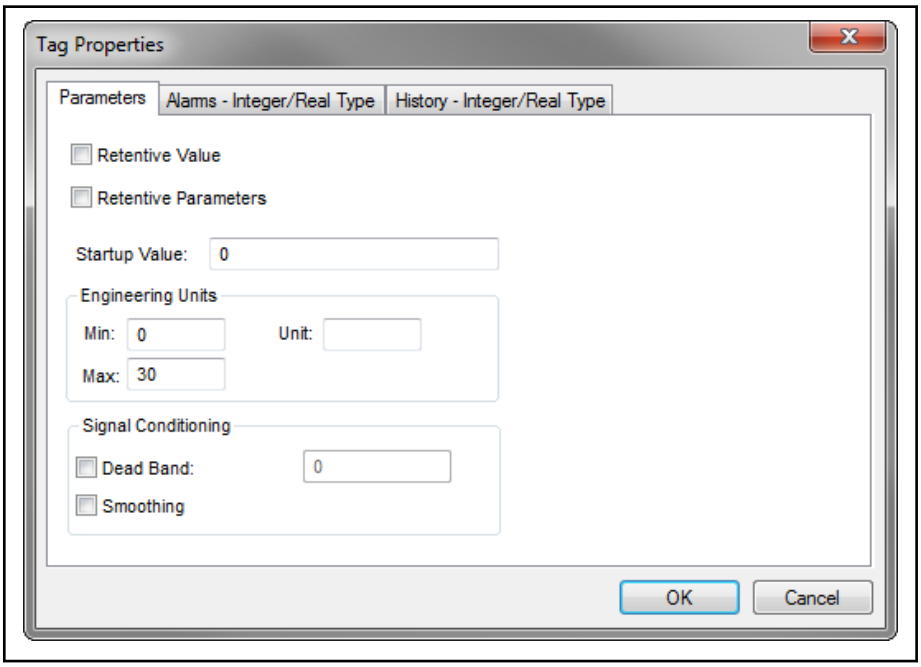


Fig. 7-16: Parameters of Integer and Real tags

Remanent value

Continuously saves the actual tag value during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved value.

When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Remanent parameters:

Continuously saves the tag properties for the project tag during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved properties.

When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Start-up value

The initial value of the project tag when the project is executed.

If the **Retentive value** option is selected, the most recently saved value is used instead of the startup value.

Engineering units

Min

The minimum permissible value of the project tag. Each attempt to set a value below this minimum is ignored and the project generates a log message indicating that someone tried to set a tag value outside the specified range.

For further information, refer to Change handling of tag values outside the validity range on page 141.

If Min and Max are 0, there is no minimum value.

Max

The maximum permissible value of the project tag. Each attempt to set a value above this minimum is ignored and the project generates a log message indicating that someone tried to set a tag value outside the specified range.

For further information, refer to Change handling of tag values outside the validity range on page 141.

If Min and Max are 0, there is no minimum value.

Unit

A brief description or reference (up to 9 characters) for the project tag.

This parameter is typically used to describe the units (e.g. kg, BTU, PSI) in which the tag value is indicated. This serves as a reference only and does not influence the actual tag value.

Signal conditioning

Dead Band

The minimum amount by which the tag value needs to change so that the new tag value is saved. (By default, each change of the tag value is saved.)

To enable dead band, select the **Dead band** checkbox and enter the dead band value in the field on the right. Do not enter a percentage; the dead band value must be specified in the same unit as the tag value.

Smoothing

Creation of an average of all consecutive changes of the tag value to reduce statistical interferences.



This may change the actual tag value.

Alarms



It is also possible to use an alarm worksheet to set the alarm properties of a project tag. This dialog box illustrates the changes you make to this worksheet and vice versa. For further information, refer to Alarm worksheet on page 313.

The following list describes the alarm properties of Integer and Real tags. These properties determine how the project checks a selected tag for alarm conditions.

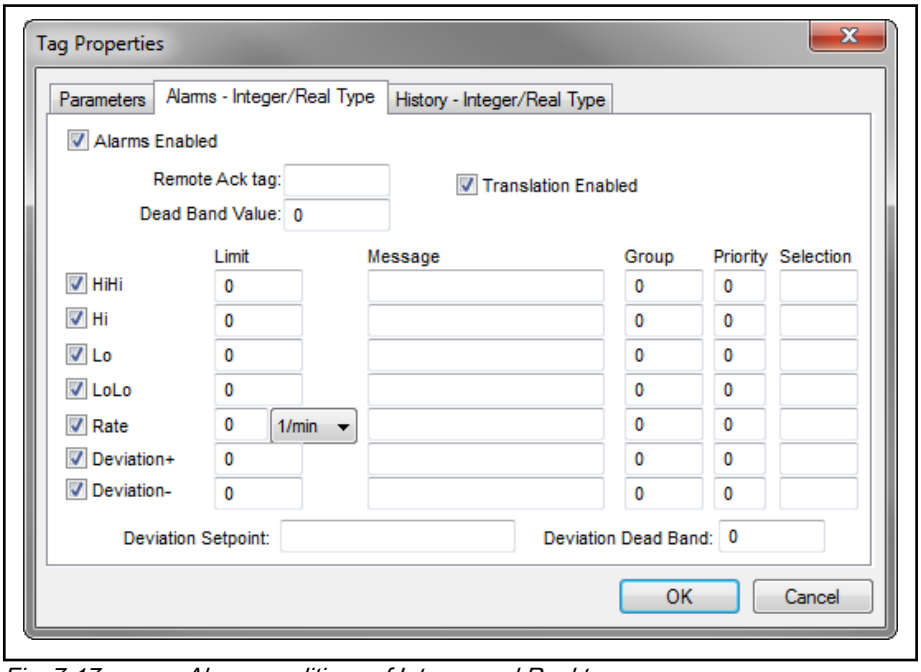


Fig. 7-17: Alarm conditions of Integer and Real tags

Alarm active

Activates the check for alarm conditions for this tag as configured in the following.

Remote ack tag

The name of another tag which can be used to acknowledge alarms on this tag. If the value of the specified tag changes, all tags which have not yet been acknowledged are acknowledged.

Dead band value

The minimum amount by which the tag value needs to remain in its normal range so that an active alarm is normalized.

A tag, for example, has a HiHi alarm limit of 90 and a dead band value of 5. If the ag value is equal or above 90, the alarm becomes active. After that, the alarm is normalized if the tag value is at or below 85.

Translation enabled

Enables the translations of the alarm messages which are configured for this tag. For further information on the translation, refer to Project localization on page 487.



Only the original alarm messages are saved in the historical data base. The translated alarm messages are saved in a separate file in your project folder: `projectname/Database/alarm.txt`

Alarms

The alarm messages which are configured for this tag. To enable a certain type of alarm condition, activate the corresponding checkbox on the left. If it is activated, additional properties are displayed:

Limit

The limit value the actual tag value has to exceed so that an alarm is activated.

Message

The message which is displayed when the alarm becomes active. Depending on the configuration of the project, messages can be displayed in an alarm/event control object, sent to personnel via email, saved in the historical database and/or sent to the runtime log.

Group

The alarm group/alarm worksheet to which this alarm condition belongs.

Priority

The priority number of this alarm. When alarms are displayed in an Alarm/event control object on page 331, the operator can filter and/or sort the alarms by priority.

Selection

An alias associated with the alarm (e.g., AreaA, AreaB, etc). When alarms are displayed in an Alarm/event control object on page 331, the operator can filter and/or sort the alarms by their selection values.

Deviation setpoint

The setpoint for the alarm conditions deviation+ and deviation-. If the actual value of the tag deviates from its setpoint value, the corresponding alarm is activated

Deviation dead band

The dead band for the alarm conditions deviation+ and deviation-. The actual value of the tag has to deviate by more than this amount from the setpoint value so that the alarm is activated.

For further information on alarm types and their use during runtime, refer to Alarm worksheet on page 313.

History

It is also possible to use a trend worksheet to set the history properties of a project tag. This dialog box illustrates the changes you make to this worksheet and vice versa.

For further information, refer to Trend worksheet on page 340.

The following list describes the history properties of Integer and Real tags. These properties determine how the project saves historical data for the selected tag.

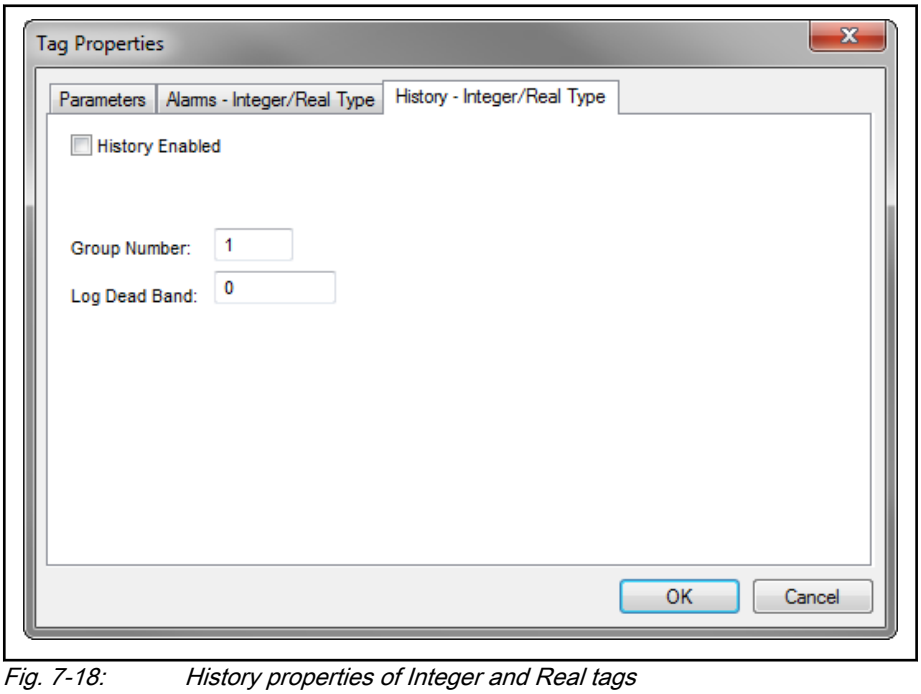


Fig. 7-18: History properties of Integer and Real tags

History active

Activates the saving of historical data for this tag.

Group number

The trend group/trend worksheet to which a project tag is assigned.

Log dead band:

The amount by which the actual value of the tag needs to change so that the change is saved in the historical database.

7.5.5 Properties of Boolean tags

Each Boolean tag in the tag database has several properties (or meta data) in addition to its actual value. You can set these properties with the help of the project tags database or with the help of the functions of the tag database.

Parameter

The following list describes the general parameters of Boolean tags. These parameters determine how the selected tag is used in your project.

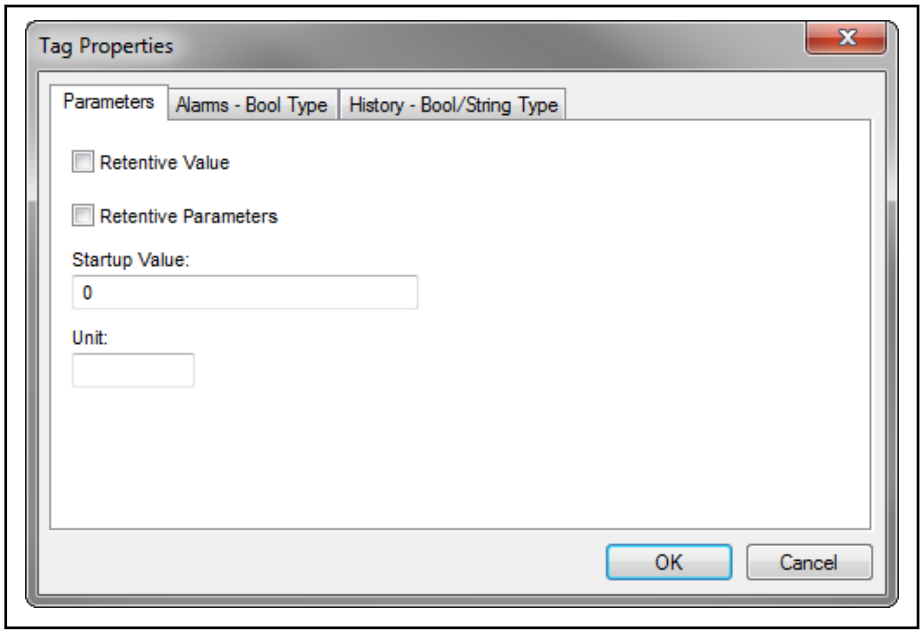


Fig. 7-19: General parameters of Boolean tags

Remanent value

Continuously saves the actual tag value during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved value.

☞ When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Remanent parameters:

Continuously saves the tag properties for the project tag during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved properties.

When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Start-up value

The initial value of the project tag when the project is executed.

If the **Retentive value** option is selected, the most recently saved value is used instead of the startup value.

Unit

A brief description or reference (up to 9 characters) for the project tag.

This property is typically used to describe the units in which the tag value is indicated. Boolean and String tags, however, do not have units. This means that this property can be used to add to the normal tag description in this case.

Alarms

It is also possible to use an alarm worksheet to set the alarm properties of a project tag. This dialog box illustrates the changes you make to this worksheet and vice versa. For further information, refer to Alarm worksheet on page 313.

The following list describes the alarm properties of Boolean tags. These properties determine how the project checks a selected tag for alarm conditions.

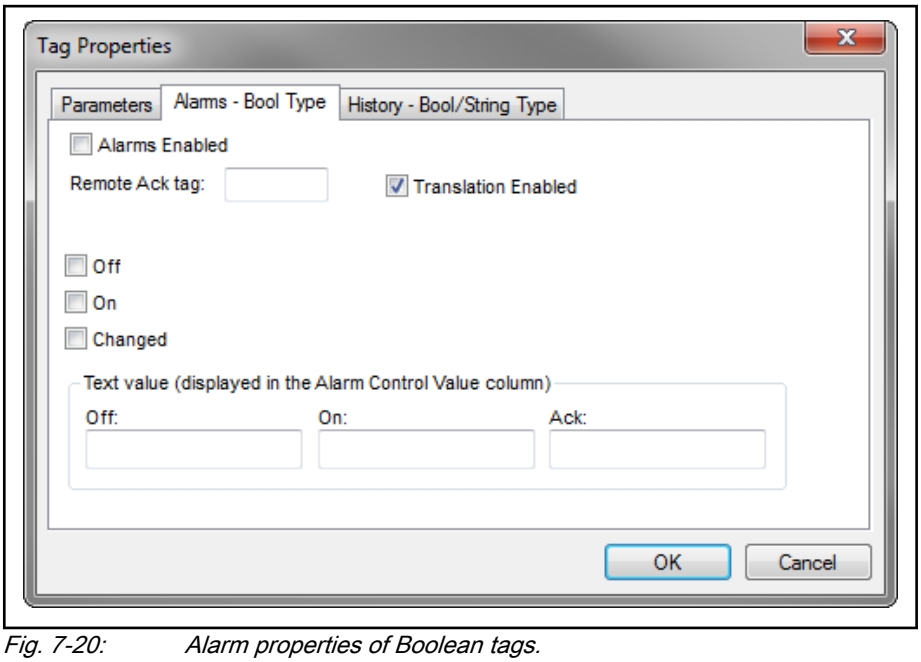


Fig. 7-20: Alarm properties of Boolean tags.

Alarm active

Activates the check for alarm conditions for this tag as configured in the following.

Remote ack tag

The name of another tag which can be used to acknowledge alarms on this tag. If the value of the specified tag changes, all tags which have not yet been acknowledged are acknowledged.

Translation enabled

Enables the translations of the alarm messages which are configured for this tag. For further information on the translation, refer to Project localization on page 487.

☞ Only the original alarm messages are saved in the historical data base. The translated alarm messages are saved in a separate file in your project folder: `projectname/Database/alarm.txt`

Alarms

The alarm messages which are configured for this tag. To enable a certain type of alarm condition, activate the corresponding checkbox on the left. If it is activated, additional properties are displayed:

Message

The message which is displayed when the alarm becomes active. Depending on the configuration of the project, messages can be displayed in an alarm/event control object, sent to personnel via email, saved in the historical database and/or sent to the runtime log.

Group

The alarm group/alarm worksheet to which this alarm condition belongs.

Priority

The priority number of this alarm. When alarms are displayed in an Alarm/event control object on page 331, the operator can filter and/or sort the alarms by priority.

Selection

An alias associated with the alarm (e.g., AreaA, AreaB, etc). When alarms are displayed in an Alarm/event control object on page 331, the operator can filter and/or sort the alarms by their selection values.

Text value

Text which is displayed in the value column of an alarm/event control object if the individual alarm types become active. There are usually mnemonics which correspond to the conditions of the Boolean tags - for example “closed” if Alarm Off is active and “open” if Alarm On is active. If these Mnemonics are not configured, the actual value of the tag (0 or 1) is displayed instead.

For further information on alarm types and their use during runtime, refer to Alarm worksheet on page 313.

History



It is also possible to use a trend worksheet to set the history properties of a project tag. This dialog box illustrates the changes you make to this worksheet and vice versa.

For further information, refer to Trend worksheet on page 340.

The following list describes the history properties of Boolean tags. These properties determine how the project saves historical data for the selected tag.

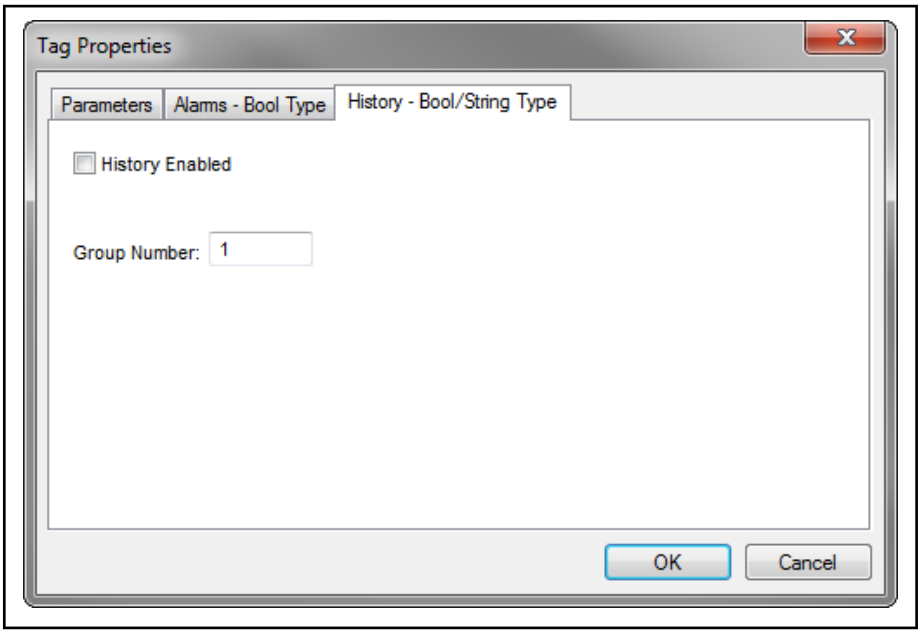


Fig. 7-21: History properties of Boolean tags

History active

Activates the saving of historical data for this tag.

Group number

The trend group/trend worksheet to which a project tag is assigned.

7.5.6 Properties of String tags

Each String tag in the tag database has several properties (or meta data) in addition to its actual value. You can set these properties with the help of the project tags database or with the help of the functions of the tag database.

Parameter

The following list describes the general parameters of String tags.

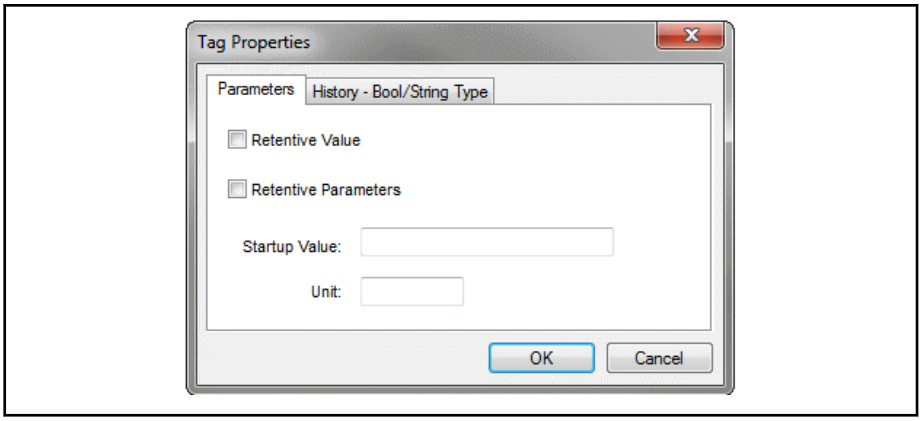


Fig. 7-22: General parameters of String tags

Remanent value

Continuously saves the actual tag value during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved value.

When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Remanent parameters:

Continuously saves the tag properties for the project tag during runtime in case the project stops unexpectedly. If the project is executed again, the project tag starts with the most recently saved properties.

When this option is selected, the hard disk access at runtime increases and thus the performance can be reduced.

Start-up value

The initial value of the project tag when the project is executed.

If the **Retentive value** option is selected, the most recently saved value is used instead of the startup value.

Unit

A brief description or reference (up to 9 characters) for the project tag.

This property is typically used to describe the units in which the tag value is indicated. Boolean and String tags, however, do not have units. This means that this property can be used to add to the normal tag description in this case.

History

It is also possible to use a trend worksheet to set the history properties of a project tag. This dialog box illustrates the changes you make to this worksheet and vice versa.

For further information, refer to Trend worksheet on page 340.

The following list describes the history properties of String tags. These properties determine how the project saves historical data for the selected tag.

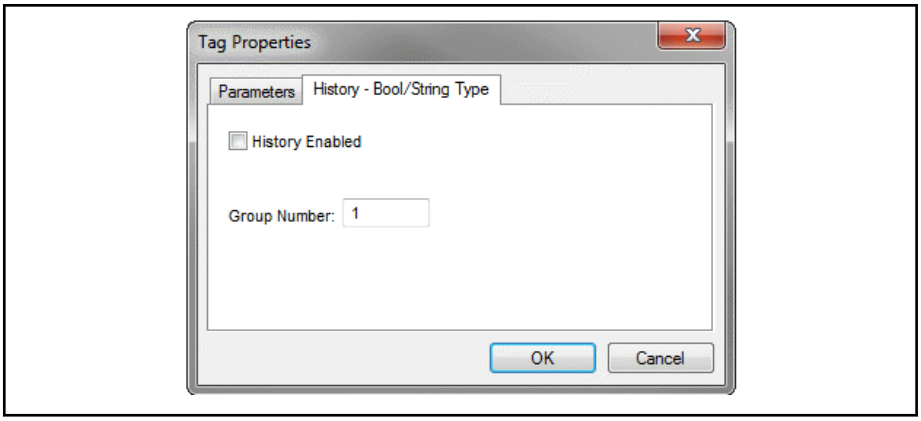


Fig. 7-23: History properties of String tags

History active

Activates the saving of historical data for this tag.

Group number

The trend group/trend worksheet to which a project tag is assigned.

7.5.7 List of tag properties

Tag properties (also known as "tag fields") are metadata attached to each tag in the database. Most of these properties can be set using the "Tag properties" dialog. To open it, click on the **Tag properties** button in the "Cross references" tool window.

To access a tag property at runtime, the following syntax (without spaces) used wherever tags would usually be specified:

```
tag_name->property_name
```

The following tag properties can be accessed at runtime:

Tag property	Description	R or R/W	Data type	Available on data type…				Retentive
				Bool	Int	Real	Str	
Name	The name of the tag, as configured in the Project Tags database.	R	String, up to 32 characters	Y	Y	Y	Y	n/a
Member-Name	The name of the class element in a properly configured class on page 106. **NOTE**: The syntax has to be: `Class.Member->MemberName` Example: `Tank.Fst->MemberName = Fst`	R	String, up to 32 characters	Y	Y	Y	Y	n/a
Size	Array size, if the tag is not an array tag, it returns the value 0	R	Integer	Y	Y	Y	Y	n/a
Index	The index number of an element in an array on page 112. (An array is any tag larger than 0.) **NOTE**: The syntax has to be: `Tag[Index]->Index` Example: `Tag[1]->Index = 1`	R	Integer	Y	Y	Y	Y	n/a

Tag property	Description	R or R/W	Data type	Available on data type…				Retentive
				Bool	Int	Real	Str	
Description	The description of the tag, configured in the tag datasheet.	R	String	Y	Y	Y	Y	Y
Quality	Tag quality (192=GOOD; 0=BAD). The project updates this field every time the tag receives the result of an expression or a value from a communication task (e.g. as driver or OPC). If the expression is invalid (e.g. division by zero) or if there is a reading communication error associated with the tag, the project sets the quality to BAD.	R	Integer	Y	Y	Y	Y	N
TimeStamp	Time and date of the last change of the tag value.	R	String	Y	Y	Y	Y	N
Blocked	This property can have two values: This is useful when all actions associated with a specific tag are to be dynamically disabled. Even if a tag is blocked, it still counts towards the total number of tags for licensing purposes. • 0: The tag is blocked and all runtime tasks ignore it. It is effectively removed from the project database. • 1: The tag is unblocked and all runtime tasks can access it normally.	R/W	Boolean	Y	Y	Y	Y	N
Unit	A brief description (up to 9 characters) of the measuring unit for the tag value. For example, **Kg**, **kWh**, **Pa**.	R/W	String, up to 9 characters	Y	Y	Y	Y	Y
Max	The maximum value that can be written to the tag at runtime.	R/W	Real	N	Y	Y	N	Y
Min	The minimum value that can be written to the tag at runtime.	R/W	Real	N	Y	Y	N	Y
B0 … B31	Value (0 or 1) of any of the 32 bits (b0, b1, b2, … b31) of an Integer tag. (B0: LSB B31: MSB)	R/W	Boolean	N	Y	N	N	N

DisplayValue	A converted tag value that is only displayed on-screen.
	`DisplayValue = (Value / UnitDiv) + UnitAdd`
	This is used when the actual tag values have a measuring unit (see **Unit** above), but need to be displayed on the screen in another measuring unit (see **DisplayUnit** below). For example, degrees Celsius and Fahrenheit.
	If the user the changes **DisplayValue** at runtime, the conversion is reversed before the change is actually written to the tag.
	`Value = (DisplayValue – UnitAdd) * UnitDiv`
		R/W	Real	N	Y	Y	N	n/a

Tag property	Description	R or R/W	Data type	Available on data type…				Retentive
				Bool	Int	Real	Str	
DisplayUnit	A brief description (up to 9 characters) of the measuring unit for **DisplayValue**. **NOTE**: This property can only be set by using the functions SetDisplayUnit on page 768 and SetTagDisplayUnit on page 768.	R	String, up to 9 characters	N	Y	Y	N	N
UnitDiv	Number by which the tag value is divided to get **DisplayValue**. To perform no division, **UnitDiv** is to be 1. **NOTE**: This property can only be set by using the functions SetDisplayUnit on page 768 and SetTagDisplayUnit on page 768.	R	Real	N	Y	Y	N	N
UnitAdd	Number added to the tag value to get **DisplayValue**. To perform no addition, **UnitAdd** is to be 0. **NOTE**: This property can only be set by using the functions SetDisplayUnit on page 768 and SetTagDisplayUnit on page 768.	R	Real	N	Y	Y	N	N

DisplayMax	The maximum value that can be entered to **DisplayValue** at runtime.
	`DisplayMax = (Max / UnitDiv) + UnitAdd`
	If **DisplayMax** is changed at runtime, **Max** is also changed as follows:
	`Max = (DisplayMax – UnitAdd) * UnitDiv`
		R/W	Real	N	Y	Y	N	N

DisplayMin	The minimum value that can be entered to **DisplayValue** at runtime.
	`DisplayMin = (Min / UnitDiv) + UnitAdd`
	If **DisplayMin** is changed at runtime, **Min** is also changed as follows:
	`Min = (DisplayMin – UnitAdd) * UnitDiv`
		R/W	Real	N	Y	Y	N	N

HiHiLimit	Limit value for the HiHi alarm	R/W	Real	N	Y	Y	N	Y
HiLimit	Limit value for the Hi alarm	R/W	Real	N	Y	Y	N	Y
LoLimit	Limit value for the Lo alarm	R/W	Real	N	Y	Y	N	Y
LoLoLimit	Limit value for the LoLo alarm	R/W	Real	N	Y	Y	N	Y
RateLimit	Limit value for the Rate alarm	R/W	Real	N	Y	Y	N	Y
DevSetpoint	Command value for deviation alarms.	R/W	Real	N	Y	Y	N	n/a
DevPLimit	Limit value for the Deviation+ alarm	R/W	Real	N	Y	Y	N	Y
DevMLimit	Limit value for the Deviation- alarm	R/W	Real	N	Y	Y	N	Y
HiHi	If 0, the HiHi alarm is not active. If 1, the HiHi alarm is active.	R	Boolean	Y	Y	Y	N	n/a

Tag property	Description	R or R/W	Data type	Available on data type…				Retentive
				Bool	Int	Real	Str	
Hi	If 0, the Hi alarm is not active. If 1, the Hi alarm is active.	R	Boolean	Y	Y	Y	N	n/a
Lo	If 0, the Lo alarm is not active. If 1, the Lo alarm is active.	R	Boolean	Y	Y	Y	N	n/a
LoLo	If 0, the LoLo alarm is not active. If 1, the LoLo alarm is active.	R	Boolean	Y	Y	Y	N	n/a
Rate	If 0, the Rate alarm is not active. If 1, the Rate alarm is active.	R	Boolean	Y	Y	Y	N	n/a
DevP	If 0, the Deviation+ alarm is not active. If 1, the DevP alarm is active.	R	Boolean	N	Y	Y	N	n/a
DevM	If 0, the Deviation- alarm is not active. If 1, the DevM alarm is active.	R	Boolean	N	Y	Y	N	n/a
AlrStatus	Integer value with the status of the currently active alarm associated with the tag. Each bit of this integer value indicates a specific status: • Bit 0 (LSB): HiHi Alarm active • Bit 1: Hi alarm active • Bit 2: Lo alarm active • Bit 3: LoLo alarm active • Bit 4: Rate alarm active • Bit 5: Deviation+ Alarm active • Bit 6: Deviation+ Alarm active Examples: If the tag >AlrStatus returns the value 2, the Hi alarm is active. If 3 is returned, HiHi alarms and Hi alarms are active at the same time. It this property returns 0, there are no active alarms associated with this tag. For Boolean tags, only the values 1 (bit 1), 4 (bit 2) or 16 (bit 4) can be returned.	R	Integer	Y	Y	Y	N	N
Ack.	This property can have two values: This works as a global acknowledgement for the tag and is only set to 0 if all alarms for the tag have been acknowledged. • 0: There are no alarms associated with this tag that require acknowledgment. • 1: There is at least one alarm associated with this tag that requires acknowledgment.	R	Boolean	Y	Y	Y	N	N

Tag property	Description	R or R/W	Data type	Available on data type…				Retentive
				Bool	Int	Real	Str	
UnAck	This property can have two values: If this value is manually set to 1, active alarms (if any) are acknowledged. The value of this property is always the opposite of the **Ack** property. • 0: There is at least one alarm associated with this tag that requires acknowledgment. • 1: There are no alarms associated with this tag that require acknowledgment.	R/W	Boolean	Y	Y	Y	N	N
AlrAckValue	Text associated with the **Acknowledged** state of a Boolean tag. This text is displayed in the “Value” column of an alarm/event control on page 331. This text can also be edited in the Tag properties dialog on page 122 (Alarms – Bool type).	R/W	String, up to 32 characters	Y	N	N	N	Y
AlrOffValue	Text associated with the **Normalized** state of a Boolean tag. This text is displayed in the “Value” column of an alarm/event control on page 331. This text can also be edited in the Tag properties dialog on page 122 (Alarms – Bool type).	R/W	String, up to 32 characters	Y	N	N	N	Y
AlrOnValue	Text associated with the **Active** state of a Boolean tag. This text is displayed in the “Value” column of an alarm/event control on page 331. This text can also be edited in the Tag properties dialog on page 122 (Alarms – Bool type).	R/W	String, up to 32 characters	Y	N	N	N	Y
AlrDisable	This property can have two values: • 0: The alarms associated with this tag are enabled. This means that if an alarm condition occurs, the alarm becomes active. • 1: The alarms associated with this tag are disabled. This means that even if an alarm condition occurs, the alarm does not become active.	R/W	Boolean	Y	Y	Y	N	N

- If a property under "saving" is marked with "n/a", this tag definition (e.g., name, size) is implicit or the value of the property is continuously derived at runtime (e.g., alarm activation, display value). To release a tag for saving, select the **Retentive parameters** option in the Tag properties dialog on page 122.
- If the project attempts to write a value outside the array specified in the **Min** and **Max** properties, the tags database will not accept the new value and a warning message is written in the Output on page 79 window. If both **Min** and **Max** properties are configured with "0" (zero), any value applied to the tag type is written to the tag.
- Tag properties can only be changed for the complete array. If the tag properties are changed for individual array elements, they are automatically changed for the complete array.
- Tag properties (such as bit arras) cannot be used to configure the worksheets Alarm on page 313 or Trend on page 340.
- Although tag properties can be applied to system tags on page 108, those properties do not remain if the project is downloaded to a CE device.

7.5.8 Change handling of tag values outside the validity range.

Sometimes a project tag contains a value which is outside its normal range. It is possible to change the handling of tag values outside this range during runtime.

This topic refers only to Integer and Real tags since they are the only tax for which Min and Max properties have a meaning. Boolean tags can only assume the values 0 and 1 and String tags do not have numeric values at all.

A project tag has many different possible values. The range is determined by the Min and Max properties of this tag. If the tag receives a new value which is outside its validity range - i.e. if the value is below the minimum or exceeds the maximum - the received value is ignored and the existing value is maintained by default. The tag quality is set to INSECURE and a warning message is sent to the output window.

In some situation, however, it is reasonable to change the handling of these values outside the validity range - in particular it may be necessary to emulate how a different hardware or software of the vendor handles these values, for example Wonderware InTouch.

To change the behavior, the project file (e.g. `projectname.APP`) has to be changed manually and the following settings have to be added:

```
[Options]
WriteOutOfRange={FALSE|TRUE}
CapOutOfRange={FALSE|TRUE}
```

WriteOutOfRange

The default setting is FALSE. If this setting is TRUE, tag values outside the validity range are accepted (i.e. the tag quality is not set to INSECURE).

CapOutOfRange
The default setting is FALSE. If this setting is TRUE, the tag values are capped at the minimum and maximum of the tag.

If no setting is made in the project file, the standard value is used.

Change handling of tag values outside the validity range:

1. Browse for your project file which usually can be found here: `[…]\Documents\WinStudio v7.4 Projects\projectname\projectname.app`
2. Open the project file in a standard text editor such as Notepad and add or change the WriteOutOfRange and CapOutOfRange settings. The following table describes how the two settings interact:

Settings	Behavior

`[Options]`
`WriteOutOfRange=FALSE`
`CapOutOfRange=FALSE`
	The existing value is ignored and the existing value is kept. The tag quality is set to INSECURE and a warning message is sent to the output window.
	This is the default behavior.

`[Options]`
`WriteOutOfRange=FALSE`
`CapOutOfRange=TRUE`
	If the received value is below Min, the tag value is set to Min and if the received value is above Max, the value is set to Max.
	The tag quality is set to INSECURE.

`[Options]`
`WriteOutOfRange=TRUE`
`CapOutOfRange=FALSE`
	Min and Max are ignored when the tag values are set. The tag value is set to a value it receives and the tag quality is set to GOOD.
	Min and Max are still used for linear conversions in device communication.

`[Options]`
`WriteOutOfRange=TRUE`
`CapOutOfRange=TRUE`
	If the received value is below Min, the tag value is set to Min and if the received value is above Max, the value is set to Max. The tag quality is set to GOOD.

3. Save and close the project file.

7.6 Using tags in the project

Once a tag has been added to the project database it can be used in the project by associating it to objects on a screen.

The basic process for associating tags to screen objects includes the following steps:

1. In the WinStudio screen, select the object to which the tag is to be applied.
2. Click one of the buttons in the Animations group on page 246 to apply that animation to the object.
3. Double-click on the object to open its Object Properties dialog on page 178.
4. Locate the **Tag** text box for that property and enter the tag name into the field. Tag text box names and locations vary, depending on the type of property that is used.

   Example:

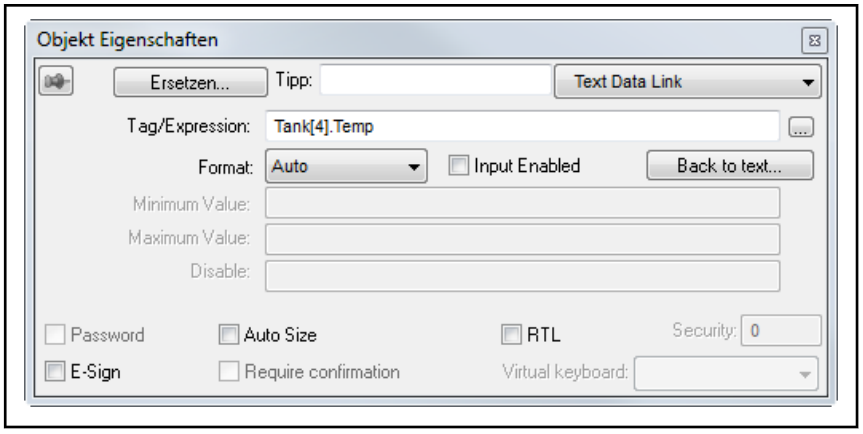


Fig. 7-24: Applying tags to an object

Comprehensive instructions for applying tags to screen objects are provided throughout the documentation where applicable.

7.7 Deleting a tag from the project database

No longer required tags can be deleted by deleting its line in the "Application Variables" datasheet.

Before deleting a tag, it is highly recommended to use the Cross reference list on page 82 tool to make sure the tag is not being used anywhere else in the project.

(If a tag is deleted that is still being used, the project cannot be verified and executed.)

Correct any screens or worksheets where the tag is being used before proceeding.

This task refers to Project tags on page 104 and on Data sheets (shared database) on page 107.

1. Stop the project on page 51 if it is running.
2. Open the Project Tags datasheet on page 104.
3. Right-click anywhere in the datasheet and then select **Disable Sort** from the shortcut menu. Tags cannot be deleted if the datasheet is sorted in any way. The tags are returned to their default order, i.e., the order in which they were created.
4. Search the line for the tag to be deleted in the datasheet
5. Right-click the line and then select **Delete Line** from the context menu. (If the option is not available, sorting needs to be disabled as described above.) A warning message is displayed to confirm this action
6. Click **Yes**. The line is deleted from the datasheet.
7. Save and close the datasheet.

7.8 Sorting or filtering the lines of a worksheet

Sort or filter the lines of a worksheet to facilitate searching the lines or finding a particular element.

Before starting this task, insert a worksheet and open it for editing. In addition it should be clear how sorting and filtering works in general spreadsheet applications.

Please note that the lines can only by sorted and filtered in the following types of worksheets:

In Project tags, Shared tags and System tags data sheets;

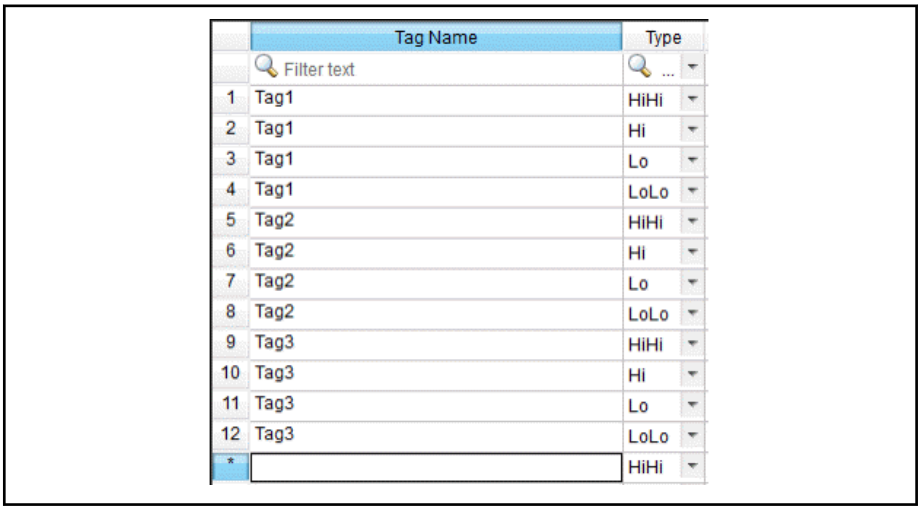
In the Translation table worksheet;

In all Task worksheets except for report and script which do not have lines;

In all Communication worksheets.

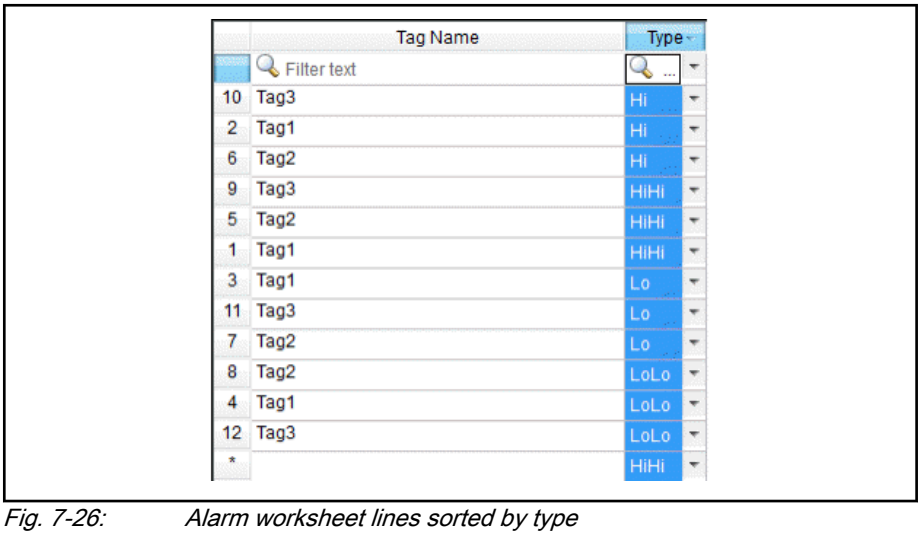
The other worksheets do not have lines to sort or filter.

The worksheets are sorted alphanumerically by selected column either in ascending ((0-9, A-Z) or descending order (Z-A, 9-0).



	Tag Name	Type
	Filter text	...
1	Tag1	HiHi
2	Tag1	Hi
3	Tag1	Lo
4	Tag1	LoLo
5	Tag2	HiHi
6	Tag2	Hi
7	Tag2	Lo
8	Tag2	LoLo
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

Fig. 7-25: Alarm worksheet lines in their original order



	Tag Name	Type
	Filter text	...
10	Tag3	Hi
2	Tag1	Hi
6	Tag2	Hi
9	Tag3	HiHi
5	Tag2	HiHi
1	Tag1	HiHi
3	Tag1	Lo
11	Tag3	Lo
7	Tag2	Lo
8	Tag2	LoLo
4	Tag1	LoLo
12	Tag3	LoLo
*		HiHi

Fig. 7-26: Alarm worksheet lines sorted by type

The worksheets are filtered according to the string entered in the selected column. Only lines that match the string will be displayed.

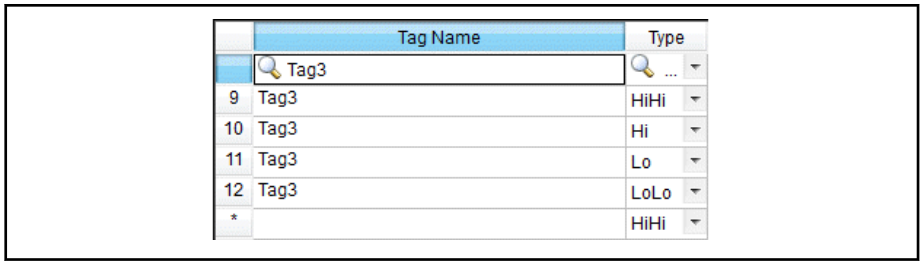


Fig. 7-27: Alarm worksheet lines filtered by tag name is "Tag3"

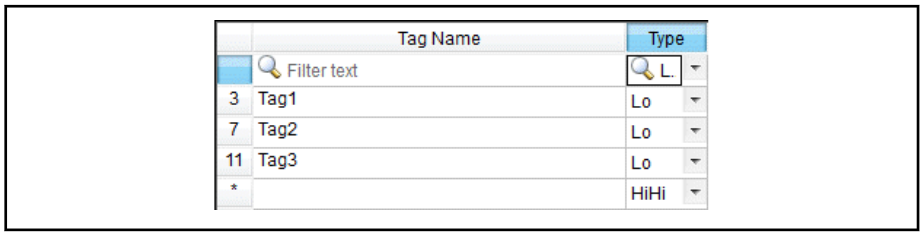


Fig. 7-28: Alarm worksheet lines filtered by type is "Lo"



It is still possible to delete lines while they are sorted or filtered.

Sorting or filtering lines:

1. To sort the lines, click on the column header by which you would like to sort. Click once to sort in ascending order and again to sort in descending order. The current order (i.e. the sorting direction) is indicated by the arrow to the right of the column name.



It is not possible to sort by several columns.

2. To cancel the sorting and restore the original order of the lines, click on the header of the first (numbered) column.
3. To filter the lines, enter the sting by which you would like to filter in the top line (zero) of the worksheet and press either **Tab** or **Enter**.

   *It is possible to use * and ? as wildcard characters in your string:*

   - * stands for any number of characters or no character. Tag* for example corresponds to `Tag`, `Tag3`, `Tag34567`, `TagA` and `Tag_TEMP`.
   - ? stands for exactly one character. `Tag?`, for example, corresponds to `Tag3` and `TagA`, while `Tag?????` corresponds to `Tag34567` and `Tag_TEMP`.

   It is possible to filter several columns. Only lines that match the filter string in all columns will be displayed.
4. To cancel filtering and restore the original order of the lines, delete the string you entered and either press **Tab** or **Enter**.

Please note that sorting or filtering the lines of a worksheet only facilitates the editing of this worksheet. How the worksheet is executed during runtime does not change. The lines are executed in their original order according to the numbering (column on the left), unless a line is moved or deleted.

7.9 Renaming and removing tags

7.9.1 Replacing tags globally

When clicking on the **Global Replace** tool from the Tag Properties toolbar in WinStudio, the following window is displayed:

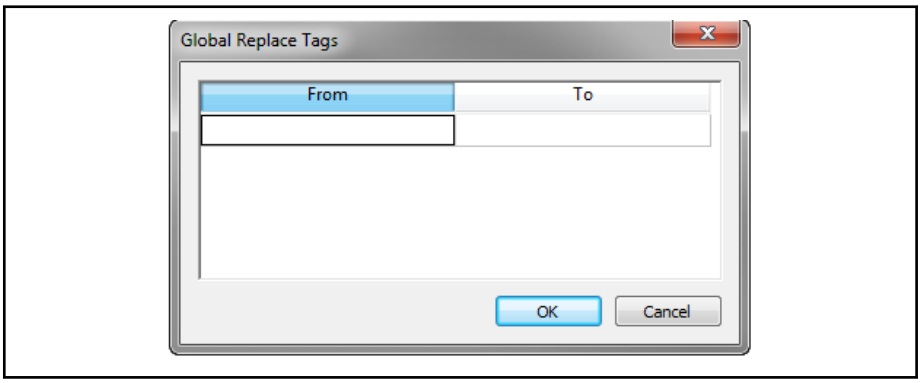


Fig. 7-29: Global Replace dialog

From the "Global Replace" dialog, any tag(s) from any document (screens and worksheets) of the whole project can be replaced. Both the **From** and the **To** column can be replaced.

When replacing compound tags (array size > 0 and/or type = class) a specific array position (for example, `TagA[1]`) or class element (for example, `TagB.MemberX`) or both (for example, `TagC[3].MemberY`) can be configured.

If only the main tag name (for example, **TagC**) in the **From** column is configured, all tags from this main tag are modified for the tag configured in the **To** column.

If an invalid replacement value is configured (for example, replace the **Main Tag** tag from a class-type tag for a simple tag (not a class tag)), the **OK** button is disabled.

When the **OK** button is pressed, the tags configured on the "Global Replace" dialog are replaced in the order that they were configured on the dialog interface.

☞ All documents have to be closed (screens and worksheets) before executing this command.

When changing the tag name on the project tags database worksheet on page 104, "WinStudio" asks whether this tag is to be replaced through the whole project.

The **Replace** option is provided in the **Edit** menu. Use this option to open the "Global Replace" dialog. However, changes are only applied to the screen or worksheet currently active.

7.9.2 Replacing Tags in a Document or Screen Object

To replace all tags in a screen object, double-click the object to open its "Object Properties" dialog and then click **Replace**.

To replace all tags in the current document, carry out one of the two following actions:

- Open the "WinStudio Screen" menu in the IndraWorks main menu and select the entry **Replace tags and strings in the current document**

- Display the "WinStudio Table Functions" toolbar and click on the **Replace tags and strings in the current document** button.

In both cases, the "Replace" dialog is displayed as described below.

One or more tags can be replaced by clicking the **Whole Tag Name** tab. The left column, **From**, displays the tags currently used. Into the right column, **To**, the new names can be entered.

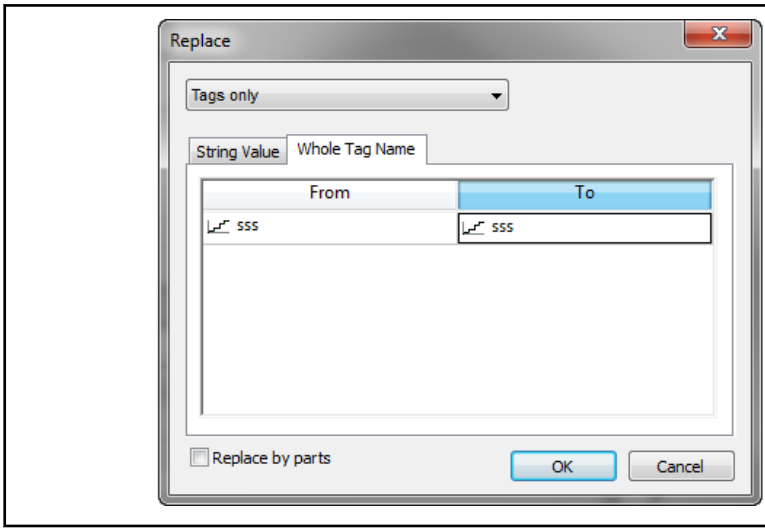


Fig. 7-30: "Replace" dialog, "Whole Tag Name" tab

Note that this does not "rename" or "delete" any tag. Tags used in the object are only replaced by other tags from the database.

By activating the "Replace by parts" option, replacing parameters (e.g. array index) within classes and arrays becomes easier.

One or more strings can also be replaced (e.g., button captions, descriptive texts) by clicking the **String Value** tab.

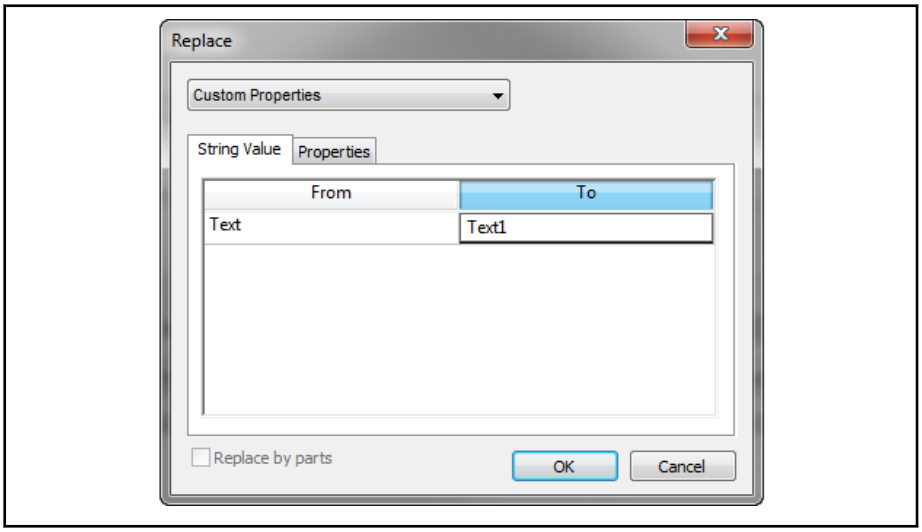


Fig. 7-31: "Replace" dialog, "String Value" tab

After editing all entries, click **OK**.

7.9.3 Removing Unused Tags from the Project Database

The **Remove unused tags** tool is used to scan the project database for unused tags to select and remove them.

"Unused tags" are tags defined in the project database that have not been used in any screen or task worksheet.

Since the project has a limited number of available tags (as determined by the product/license type), some or all of these unused tags can be removed to decrease the project tag count.

To search for unused tags and remove them, proceed as follows:

1. Save and close all open project screens and worksheets.
2. Execute the "Verify Application" function in the context menu of the "WinStudio" folder and press the "Verify" button in the "Verify Application" dialog.

   The project is automatically verified by the development application. If unused tags are found, a message is displayed in the Messages window.

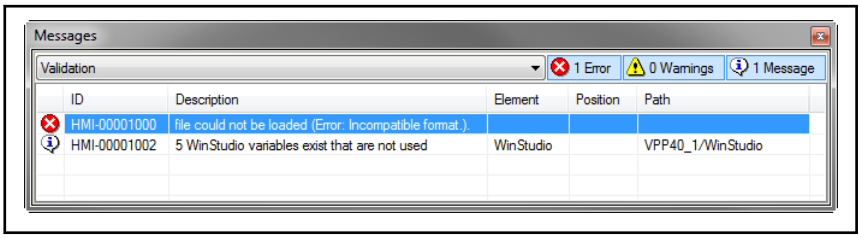


Fig. 7-32: Information on unused tags on "Verify Application"

Double-click on the respective line in the Messages window. The following dialog is displayed:

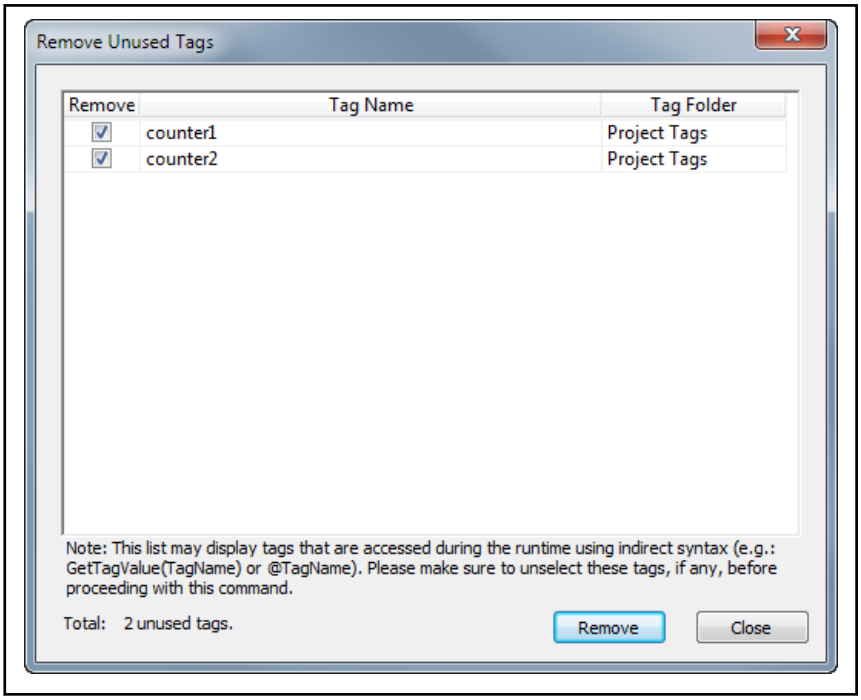


Fig. 7-33: List of unused tags in the "Remove Unused Tags" dialog

3. *Determine which tags are to be removed, if any*
   - If all of the listed tags are to be removed, click **Remove**.
   - To keep some of the listed tags, clear the "Remove" option for these tags and then click **Remove**.



The listed tags may include some that are accessed at runtime using indirect syntax (e.g., **GetTagValue(TagName)** or **@TagName**, where the value of **TagName** is the name of an unused tag).

- If none of the listed tags are to be removed, click **Close**.

WinStudio removes the selected tags and then asks to verify the project again

4. Click **Yes** to verify the project again

7.9.4 Tags database reset

Execute the function **Reset Tags Database** to reload the tags database on the local station.

This command affects all tags stored in the "Application Variables" folder.

This option is useful for resetting the application variables and restoring the values they had when the project was loaded for the first time.

If the project is stopped while the development environment remains open, the tags are not reset by default if the project is executed again. Therefore, this command can be used to reset these tags before the project is executed again.

If this command is executed, the **startup value** configured for each tag (Tags properties dialog on page 150) is entered into the respective tag.

If no **Startup Value** has been configured for a numeric tag (**Boolean**, **Integer** or **Real**), the value "0" (zero) is entered into the tag.

If no **Startup Value** has been configured for a String tag, the empty value (" ") is entered into the tag.

This command is disabled (in gray) if there is at least one runtime task running on the local station.

Before executing this command, all runtime tasks have to be closed (**Stop application** in the context menu of the "WinStudio" folder).



The tags in the "Internal Variables" and "Shared Variables" folders are not affected by this command.



If the application variables are to be reset automatically on running the project (**Start application** in the context menu of the "WinStudio" folder), the option **Reset tags database when starting project** on the Preferences tab of the Project Settings dialog on page 69 can be selected.

7.9.5 Using available tags

Click the **Object Finder**  to open the "Object Finder" dialog, which lists all **Tags** and **Functions** currently configured for the project.

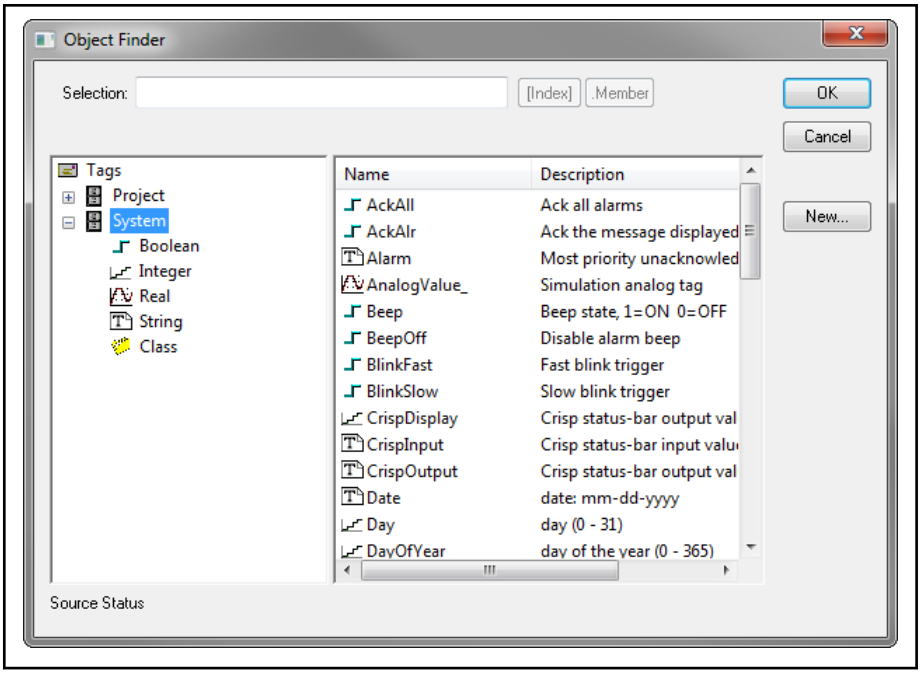


Fig. 7-34: Object Finder dialog

To select an existing tag/function, double-click on the tag/function name, and then click **OK** to close the box. The selected name is displayed in the **Tag name** text box.

- To select a specific array index, click the **Index** button after specifying the array tag on page 112 name.
- To select a specific element of a class tag, click the **Element** button after specifying the class tag on page 106 name.
- To create a new tag, click the **New** button.
- When the "New Tag" dialog is displayed, enter the following information and click **OK** to close the box.
  - **Name**
  - **Array size**
  - **Type** (Boolean, Integer, Real, String, Class:Control, Class:msgonline or Class:Alr)
  - **Description**
  - **Scope** (local or server)

7.9.6 Changing tag properties

The "Tag Properties" dialog can be accessed via "WinStudio Cross References".

Click the **Tag Properties** tool to configure parameters for each tag.

(For further information on specifying tag properties, refer to understanding tag properties and parameters on page 122.)

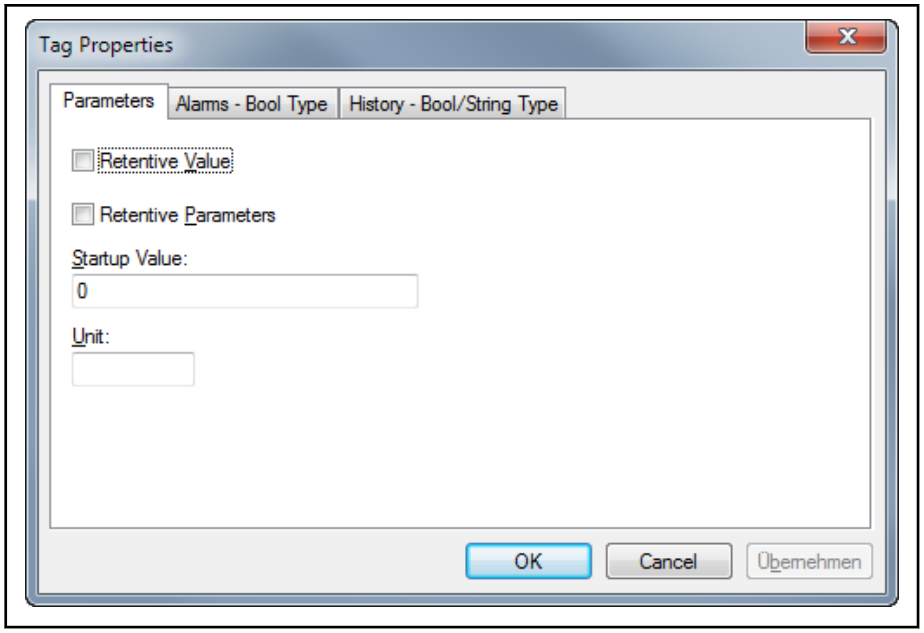


Fig. 7-35: Example for Tag Properties dialog

The "Tag Properties" dialog that displays for Boolean, Integer, Real, and String tags differs in content.

7.10 Importing an external database

7.10.1 Import Wizard

The Import Wizard is a powerful tool that reduces engineering time during project development.

The Import Wizard enables import of tags from different data sources directly to the project tags database. Depending on the data source, not only tag names can be imported but also the communication interface (the link between the tags and the PLC addresses).

Select the **Variable Import Wizard** in the "Tools" context menu of the "WinStudio" folder to display the "Import Database Wizard" dialog as step-by-step guide through the process of importing tags.

Three steps for importing tags from these data source types are necessary:

- WinStudio Project Database
- OPC Server Database
- CSV Database
- ODBC Database

1 Step: Select the Source Type

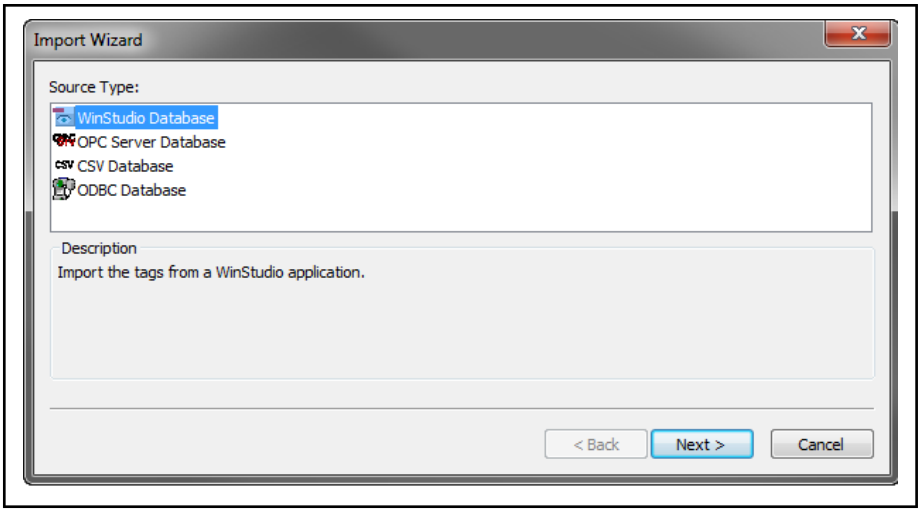


Fig. 7-36: Import Wizard

Click on the data source type from which the tags are to be imported. Click on **Next**.

Continue with the respective section of the instructions required to complete the database import: "Data source types" on page 154

2 Step: Configure the Source Type Settings

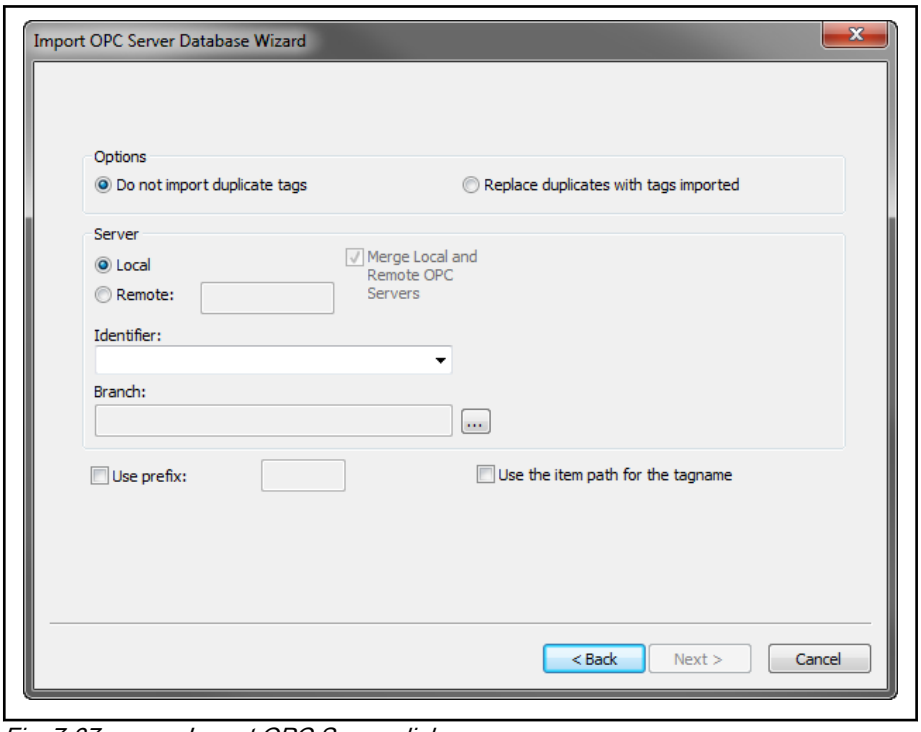


Fig. 7-37: Import OPC Server dialog

Most of the settings in the second window depend on the data source type selected in the first step. The screenshot above is an example of one data source type (OPC Server Database). The settings that are common for any data Source Type are described below:

- **Options**: Select **Do not import duplicate tags** to prevent that imported tags overwrite tags with the same name that already exist in the tags database of the current project.

  Select **Replace duplicates with tags imported** to overwrite tags in the Tags Database with imported tags of the same name.

- **Use prefix**: Enable to specify a prefix (up to 4 characters) to be added in front of the name of the imported tags.

  It is useful to use a prefix to differentiate the imported tags from the tags created manually.

The other settings vary according to the data source selected in the first step and they are described in the specific sections for each "Data source types" on page 154.

After configuring the settings in this dialog, click **Next**.

3 Step: Filter the tags

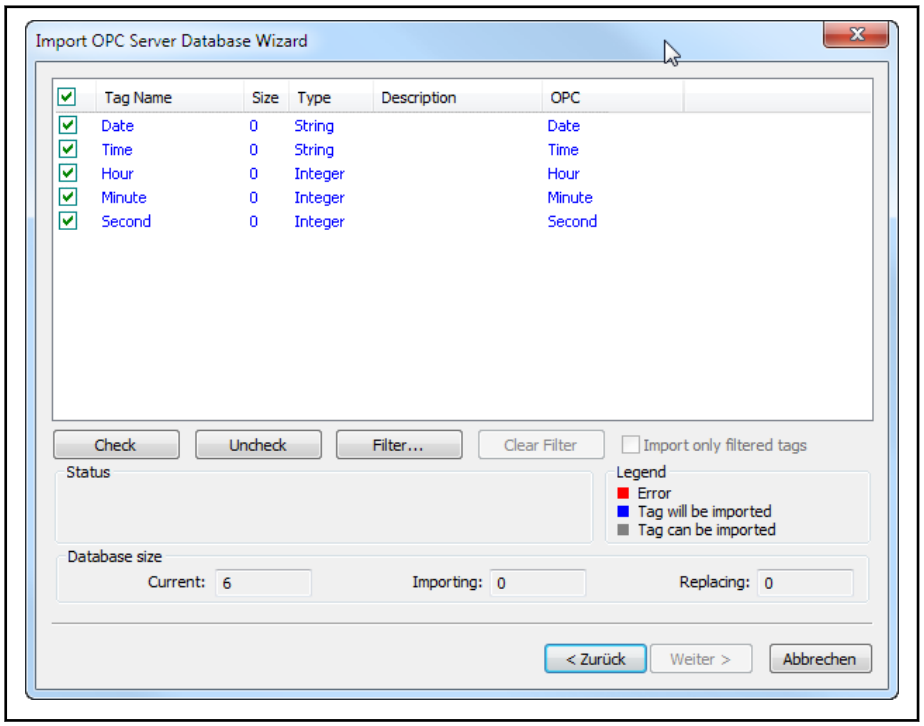


Fig. 7-38: Import OPC Server dialog

The preceeding figure is an example of one data source type (OPC Server Database). The fields and settings that are common for all data source types include the following:

- **Grid**: Displays the list of tags found on the data source.
  - **Checkbox**: Enable to import the tag from the data source to the tags database of the current project.
  - **Tag Name**: Name of the tag
  - **Size**: Array size of the tag
  - **Type**: Data type of the tag (Boolean, Integer, Real, String or Class:<ClassName>)
  - **Description**: Description of the tag
- **Check**: Click to select/import all tags in the grid
- **Uncheck**: Click to uncheck all tags in the grid
- **Filter**: Click to filter the tags. The Filter dialog is displayed on which a mask can be specified for each column in the grid. Placeholders (* and ?) can be used to filter data.
- **Clear Filter**: Click to reset the filter.

- **Import only filtered tags**: Check this option to import only the tags that are displayed in the grid (filtered).
- **Status**: Displays a message describing the status of the tag currently selected in the grid. This information is especially useful to indicate why a tag cannot be imported.
- **Legend**: Describes the meaning of the colors that indicate the tag status:
  - (Red) **Error:** Tag cannot be imported because it is not supported by "WinStudio". Refer to the text field for a detailed description of the error.
  - (Blue) **Tag will be imported:** The tag will be imported after clicking the Finish button.
  - (Gray) **Tag can be imported:** The tag can be imported but it has not been selected.
- **Database size** pane: Displays summary information regarding the current Import Wizard:
  - **Current**: Displays the current number of available tags in the "Application Variables" datasheet.
  - **Importing**: Indicates the number of tags selected to be imported
  - **Replacing**: Indicates the number of tags configured in the project tags database of the current project that will be replaced by an imported tag with the same name.

After selecting the tags to import, click the Finish button or click Cancel to abort the operation.

The other settings vary according to the data source selected in the first step and they are described in the following sections for each data source type.

Data source types

Continue with the respective section for the instructions to complete the import database procedure:

- Importing from Other Project Databases on page 154
- Importing from OPC Server Databases on page 156
- Importing from CSV Databases on page 157
- Importing from ODBC Databases on page 158

7.10.2 Importing from...

Importing from Other Project Databases

This wizard allows to import the objects (tags and worksheets) of other "WinStudio" projects.

If only the tags (rather than the whole project) are imported from a remote computer, the TCP/IP client worksheet can be automatically created to link the tags between both stations (the local and the remote) and to share the value of these tags between both stations at runtime.

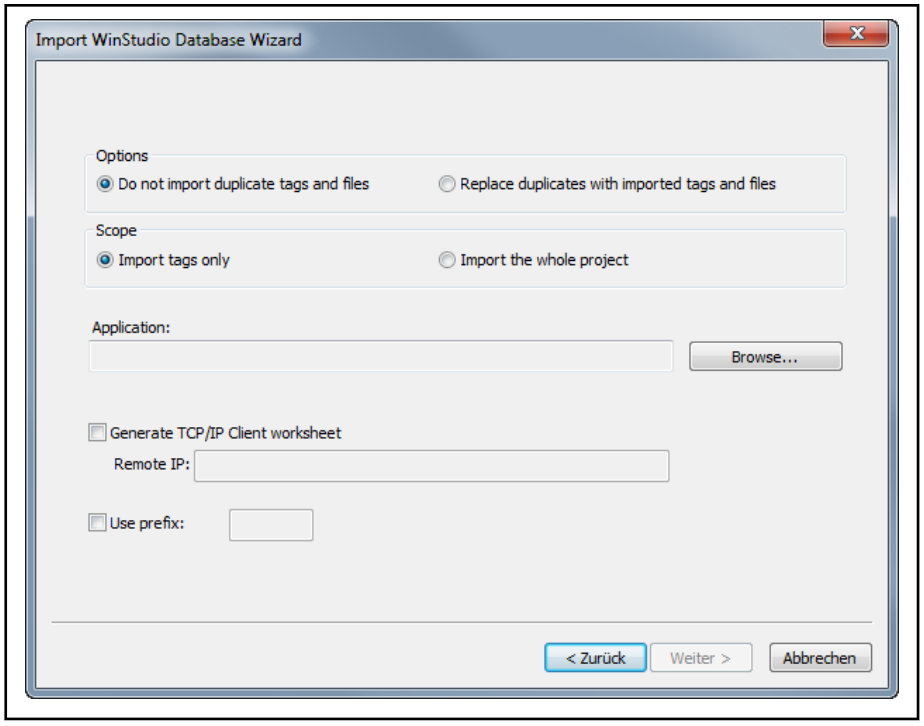


Fig. 7-39: "Import WinStudio Database Wizard" dialog

- **Import tags only**: If this option is selected, the tags of the other project are imported to the current project. The other elements of the project (worksheets) are not imported.
- **Import the whole project**: If this option is selected, the following elements of the other project are imported to the current project.
  - Tags database
  - Global procedures
  - Screens
  - Screen groups
  - Web pages
  - Alarms
  - Trends
  - Recipes
  - Reports
  - ODBC
  - Math
  - Scripts
  - Scheduler
  - Drivers
  - OPC
  - TCP/IP
  - DDE

  This option is useful for merging projects and importing template projects.



When selecting the option to **Import the whole project**, the following worksheets are always imported, regardless of existing worksheets with the same number in the current project: ODBC, Math, Script, Scheduler, Drivers, OPC, TCP/IP and DDE.

If there are worksheets with the same number in the current project, worksheets imported from the other project are inserted as additional worksheets in the current project and the number of each worksheet is automatically increased to avoid replacing files on the current project.

- **Do not import duplicate tags and files**: If this option is selected, the following elements are not imported in case there is already an equivalent element in the current project:
  - Tags database (tags with the same name are not imported)
  - Global procedures (the global procedures are not imported)
  - Screens (screens with the same name are not imported)
  - Screen groups (screen groups with the same name are not imported)
  - Web pages (Web pages with the same name are not imported)
  - Alarms (alarms assigned to tags with the same name are not imported)
  - Trend (trend logs assigned to tags with the same name are not imported)
  - Recipes (recipes with the same name are not imported)
  - Reports (reports with the same name are not imported)
  - Script (the startup script is not imported)
- **Application**: Click **Browse...** and select the **project_name.APP** file with the tags to be imported.
- **Generate TCP/IP client worksheet**: If the project is to share tags with another "WinStudio" project running on a remote server, select this option and enter the IP address of the respective server. "WinStudio" automatically configures the TCP/IP client worksheet to exchange data with the remote project.
- **Use prefix**: Select this option to specify a prefix (up to 4 characters) that is added in front of the names of the imported tags. It is useful to differentiate the imported tags from the tags created manually.

Importing from OPC Server Databases

This wizard enables tag import from either a local OPC server or a remote one.

If tags are imported from the OPC server, the OPC client worksheet is created automatically to link the tags. For this reason, the communication interface between the "WinStudio" OPC client and the external OPC server does not need to be configured.

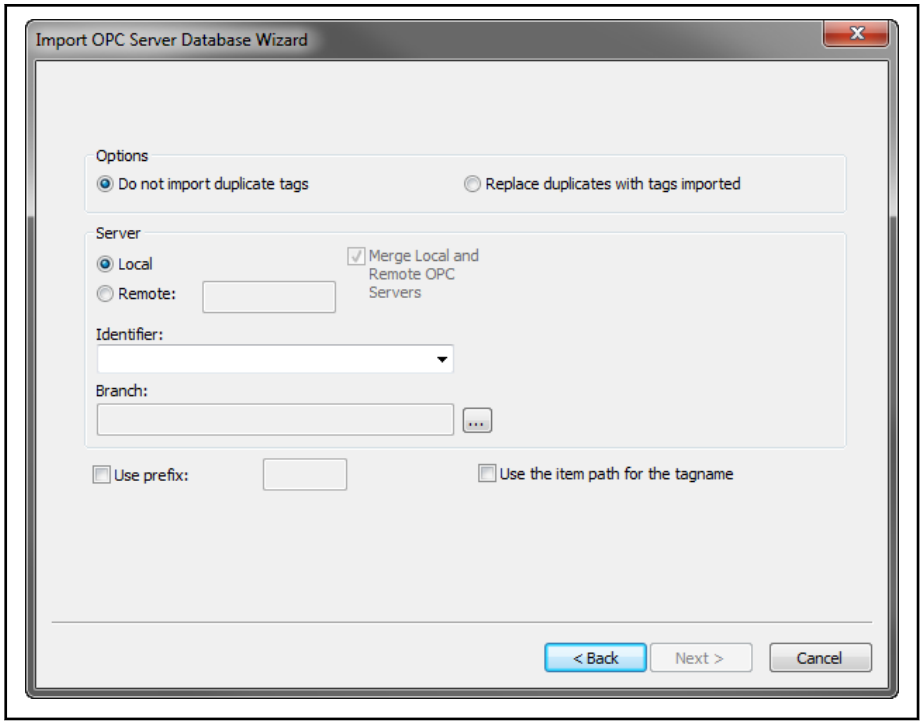


Fig. 7-40: Import OPC Server Database Wizard

- **Local/Remote**: Provide the following options:
  - **Local:** Select this option to import tags from an OPC server installed on a local computer.
  - **Remote:** Select this option to import tags from an OPC server installed on a remote computer. Enter the IP Address (or the host name) of the remote computer on which "WinStudio" is running in the **Remote** field.
- **Merge Local and Remote OPC Servers**: If a remote server is selected, select this option to display the list of OPC servers installed on the local and remote computer. Uncheck this checkbox to display only the list of OPC servers installed on the remote computer.
- **Identifier**: Displays the list of available OPC servers.
- **Branch**: Click on the Browse button (**...**) to select the branch of the OPC server from which the tags (items) are to be imported. Leave this field blank to import tags from all branches configured on the OPC server.
- **Use the item path for the tagname**: Check this option to add the path name to the item name when importing tags from the OPC server. Uncheck this option to use only the item names configured on the OPC server.

For this data source type, the table displayed in step 3 of the **Import Wizard** (in the context menu of the "WinStudio" folder) offers an additional **OPC** field displaying the item name on the OPC server.



Refer to Steps 1, 2 and 3 of **Import Wizard** for the settings and fields that are common for all source types.

Importing from CSV Databases

This wizard enables importing tags from a text file into the CSV (Comma Separated Values) format or any similar format.

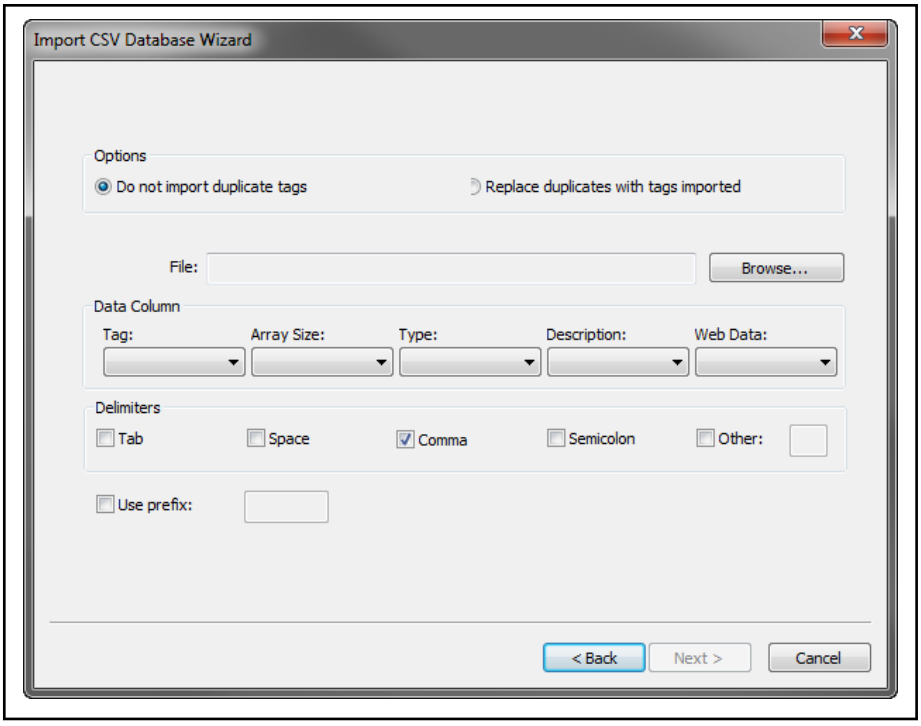


Fig. 7-41: Import CSV Database Wizard

- **File name**: Click the "Browse" button to select the text file from which the tags are imported.
- **Data Column**: Select a number for each tag property that corresponds to its column number in the import file.

  For example, if the Tag, Array Size and Type are listed in the second, third and first columns in the import file, respectively, select 2 in Tag, 3 in Array Size and 1 in Type. The Tag property (tag name) is mandatory but the other properties are optional.

  For properties that are not included in the text file, select the option **Not used**. "WinStudio" inserts defaults or leaves the field blank, according to the following list:

  – **Array size**: 0
  – **Type**: Integer
  – **Description**: <Blank>
  – **Web Data**: Local
- **Delimiters**: Select the delimiter(s) used in the text file to divide one column from another. For a CSV file, the delimiter is Comma (the default). More than one delimiter can be selected simultaneously and a custom delimiter can be entered by using the option "Other".



Refer to Steps 1, 2 and 3 of **Import Wizard** for the settings and fields that are common for all source types.

Importing from ODBC Databases

This wizard enables importing tags from an external SQL relational database such as Microsoft Access, SQL server, Oracle, My SQL, Sybase and others via the ODBC interface.

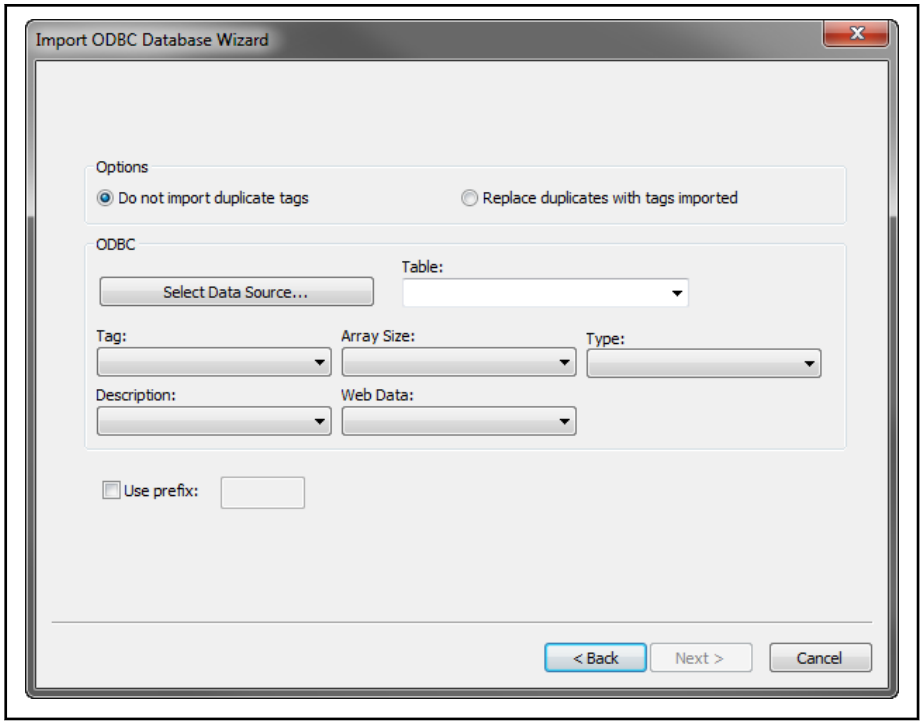


Fig. 7-42: Import ODBC Database Wizard

- **Select Data Source**: Click to select the ODBC Data Source Name (DSN) linked to the database from which the tags are to be imported. The DSN has to be created before by means of the Data Sources (ODBC) window (**Control Panel ▸ Administrative Tools ▸ Data Sources [ODBC]**).

  After selecting a DSN, the other fields in this window are automatically filled with information from the selected database.
- **Table**: Select the table that holds the tags in the import database.
- **Tag**: Select the name of the column that holds the tags in the import database.
- **Array Size**: Select the name of the column that holds the array size for the tags in the import database.
- **Type**: Select the name of the column that holds the tag type in the import database.
- **Description**: Select the name of the column that holds the tag description in the import database.
- **Web Data**: Select the name of the column that holds the Web Data for the tags in the import database.

Refer to Steps 1, 2 and 3 of **Import Wizard** for the settings and fields that are common for all source types.

Importing Studio XML screens

With the help of the import wizard, XML templates can be imported and WinStudio screens can be generated based on them. The design of this external text file corresponds to the XML scheme for WinStudio screens.

Further editing is possible provided that an XML template file is available

A WinStudio XML template file contains the same information as a regular binary screen file (*.SCR). In contrast to the binary file, the content is available in readable form and is therefore more flexible and portable.

If XML templates were created for your screens, use the import wizard to import them in your WinStudio project.



It is also possible to use the ImportXML on page 749 function to import Studio XML screen files during runtime.

Importing one or several XML templates for WinStudio screens

1. In the context menu of the "WinStudio" folder, select **Tools** ▸ **Variable import wizard**.

   The Import wizard dialog window is displayed.

2. In the **Source type** list , click **Studio XML screen** and then **Next**. The next step of the import wizard is displayed.

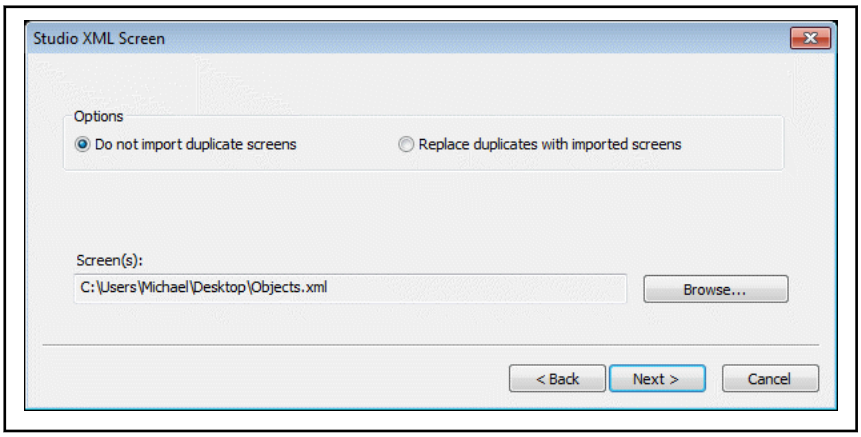


Fig. 7-43: Selection of the screens to be imported

3. Under **Options**, select if the imported screens are to automatically replace the existing screens in your project. Screens are considered as duplicates if they have the same file name. `Objects.xml` and `Objects.scr` would be duplicates, for example.

   - If you do not want the imported screens to automatically replace the existing screens in your project, select "Do not import duplicate screens". A warning is displayed for each duplicate you are trying to import.
   - If the imported screens are to automatically replace the existing screens in your project, select "Replace duplicates with imported screens".

4. Click **Browse** to open the file browser.
5. With the help of the file browser, select XML template files to import and click on "Open". It is possible to select several files by clicking on additional entries while holding down the <CTRL> key. The selected files are displayed in the **Screen(s)** field:
6. Click on **Next**. The selected screens are processed. In the next step of the import wizard, the WinStudio variables which are included in the XML templates are displayed,

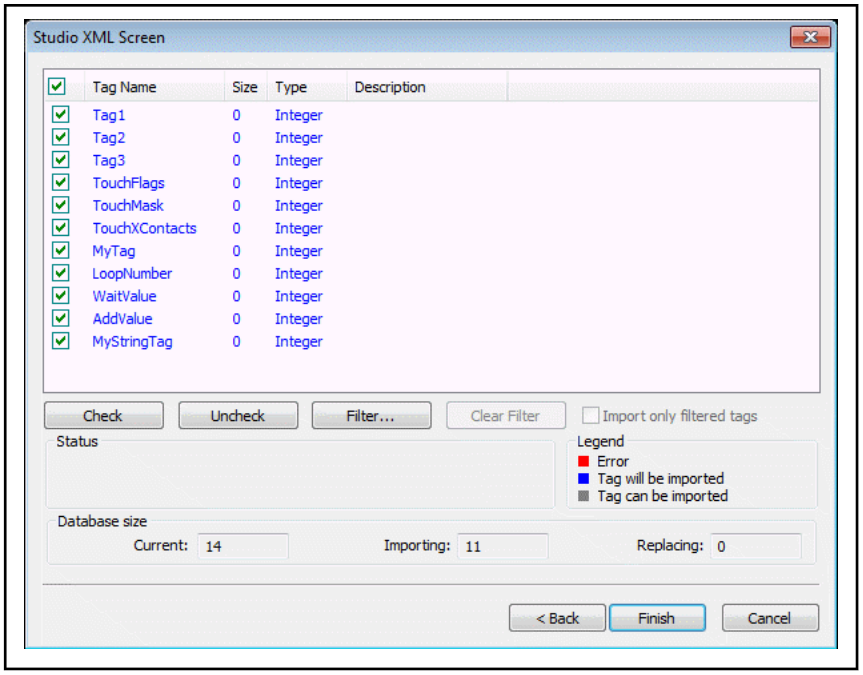


Fig. 7-44: Selection of the WinStudio variables to be imported.

7. Select the variables to import together with the screens from the list:
   - Individual variables can be selected or deselected with the help of the checkbox.
   - If all variables are to be selected or deselected, use the **Check** or **Uncheck** buttons.
   - To filter the tag list, click on **Filter** and configure filter expressions for one or several columns. It is possible to use placeholder characters (* and ?) in the filter expressions.
8. Click **Finish**.

The screens and the corresponding tags are imported into your project. In addition, the screens are automatically published for Thin Clients and Mobile Access — i.e. the corresponding `*.HTML`, `*.SCC` and `*.SSMA` files are automatically generated and saved in the web folder of your project so that the imported screens can be selected immediately.

7.11 Tag integration

7.11.1 Tag integration, general information

With the automatic tag integration function, variables configured in control systems can be directly selected in the WinStudio variable browser (Object Finder) without having to create a respective tag and configuring a driver sheet first.

This is different from the other options to include third-party tags in the project, such as importing the tag definitions or configuring a worksheet to associate project tags with external items.

With tag integration, the third-party tags are "live", i.e., they are continuously and bilaterally updated at runtime as long as the third-party control application or device are also running, and they can be used anywhere in the project where project tags would normally be used.

There are two types of tag integration:

1. **Integrated**: All tags from the source can be browsed in the Object Finder, but individual tags are neither added to the project tags database nor counted against the project tag limit until they are actually used somewhere in the project.
2. **Shared**: All tags from the source are automatically added to the project tags database and counted against the project tag limit.



This option is obsolete and available for compatibility with older projects. If this type is to be used in a new project, please contact Customer Support for support.

Tag integration is part of the project settings.

- Integrating tags from IndraLogic on page 162

  WinStudio can connect to and share tags on Bosch control systems and IndraLogic devices.

- Using the Variable selection on page 149 to select integrated tags

  After adding a tag integration source to the project, the tags from that source can be viewed and selected in the Object Finder like any other project tag.

- Renaming integrated tags in the project on page 166

  If WinStudio integrates tags from control systems, the variable names cannot be applied without changes. Some changes are made to improve tag management and to adhere to the local tag name syntax.

7.11.2 Sharing tags on IndraLogic devices

WinStudio can connect to and share tags on Bosch control systems and IndraLogic devices.

Automatically sharing tags via IndraWorks

If WinStudio is executed along with Rexroth IndraWorks on the same station, all device communication and tag sharing is managed by IndraWorks. No configuration in the development application is required. WinStudio automatically connects to all registered devices and makes their tags available for use in your project. The tags are dynamically and continuously updated and can be browsed using the Shared Database on page 107 folder or the Object Finder on page 149.

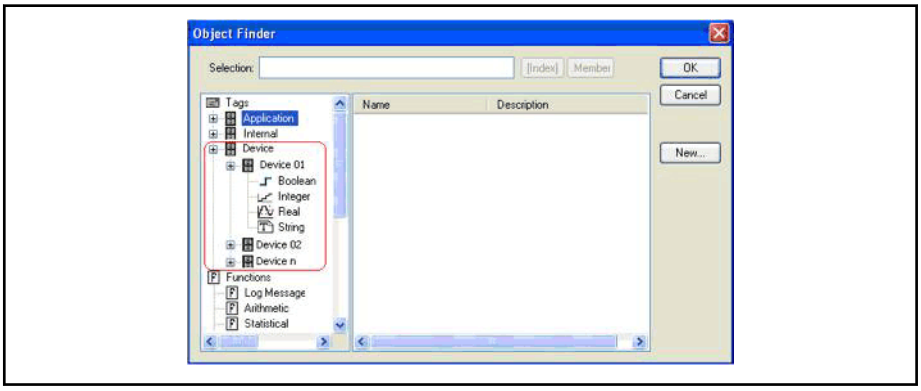


Fig. 7-45: Browsing devices in the object finder

The tags are grouped by device in the "Object Finder" to make them easier to browse, and they are also categorized by data type to ensure compatibility

with WinStudio objects and functions. These shared tags can be used in the project just like any other project tag.

For further information on using Rexroth WinStudio and configuring IndraLogic devices, please refer to the "IndraWorks Operation and Programming Guide".

Configuring WinStudio to communicate with devices

If the WinStudio project and the used control system are managed together in one IndraWorks project, no further settings are required.

If variables from control systems are to be used that are not included in the project, proceed as follows to add them:

1. In the Project Settings dialog of the WinStudio folder, open the **Communication** tab.

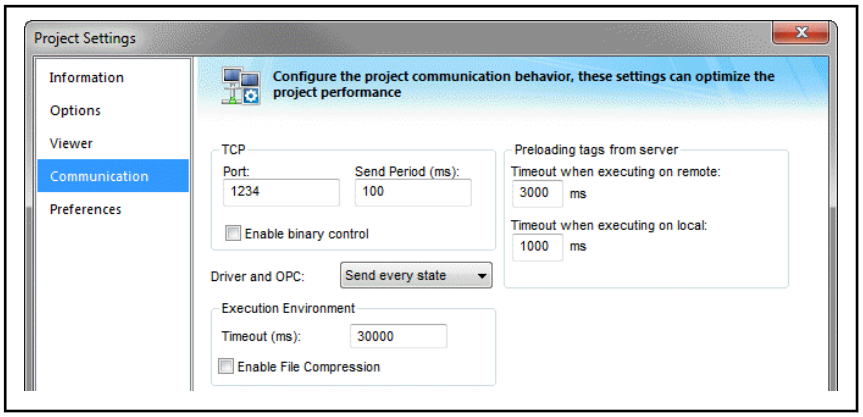


Fig. 7-46: Communication tab of the Project Settings dialog

2. In the "Tag Integration" pane, click **Add**. The "Tag Integration Source" dialog is displayed:

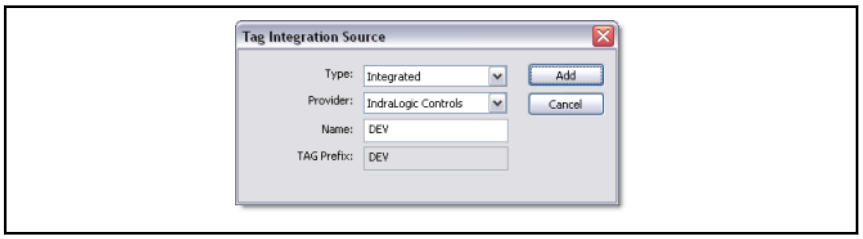


Fig. 7-47: Selecting the source for tag integration

3. In the **Type** list, select **Integrated** if it is not already selected.
4. Select the communication type supported by the control system from the **Provider** list.
5. In the **Name** field, type a name for the target device. This is the name that is displayed in the "Object Finder" and under which the tags are grouped. The name has to follow the proper tag name syntax. Otherwise, WinStudio does not accept it. For further information, refer to "Understanding the tag name syntax" on page 109. As you type the device name, WinStudio automatically duplicates it in the **TAG Prefix** box. This prefix is added to the local project tags that are created to synchronize with the device tags. (Refer to "How device tags can be renamed in WinStudio" below.)
6. Click on **Add**. A configuration dialog is displayed to adjust special settings for the selected provider.

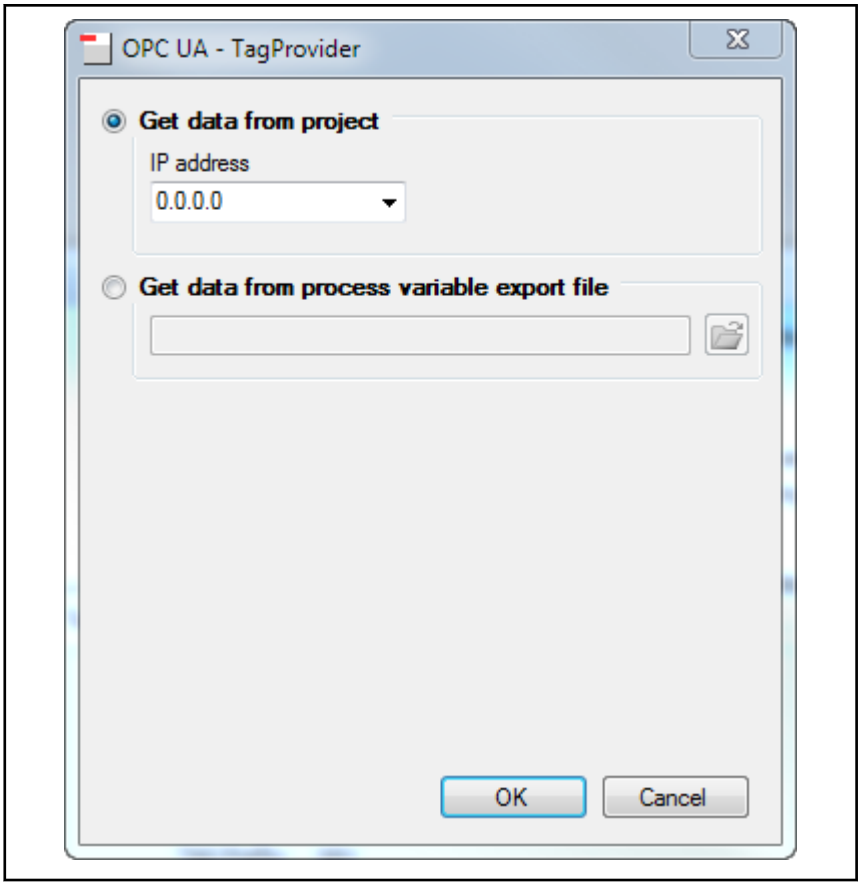


Fig. 7-48: Variable provider configuration for OPC UA communication

7. Click on **OK**.

How device tags can be renamed in WinStudio

If WinStudio creates local project tags to correspond to the device tags, the device tag names cannot be applied directly without changes. Some changes are made to improve tag management and to adhere to WinStudio's tag name syntax.

The full, original name of a device tag can always be retrieved by accessing the "DeviceTag" tag field on the corresponding project tag.

For example, `tagName->DeviceTag`.

Inserting the tag prefix

First of all, since the project may connect to multiple devices that have the same control program and device tags, the development application automatically inserts the device name / tag prefix specified when adding the device.

For example, for a device tag named…

`MyDeviceTAG[1].ClassMember`

…the corresponding WinStudio project tag is named…

`device_name.MyDeviceTAG[1].ClassMember`

This allows to differentiate between similar tags on different devices.

Multidimensional arrays

WinStudio does not support multidimensional arrays. For this reason, if device tags have more than one array index, each index after the first will be indicated by `_Index_`.

For example, for a device tag named…

`MyDeviceTAG[1][2][3].ClassMember`

…the corresponding WinStudio project tag is named…

`device_name.MyDeviceTAG[1]_2__3_.ClassMember`

Nested classes

WinStudio does not support nested classes. For this reason, if device tags have more than one class member, each class member after the first will be represented by `_ClassMember`.

For example, for a device tag named…

`MyDeviceTAG.ClassMember.ClassMember2`

…the corresponding WinStudio project tag is named…

`device_name.MyDeviceTAG.ClassMember_ClassMember2`

Refreshing tag information from devices

In the "Object Finder", configured devices can be displayed with three different symbols:

Symbol	Description
	Device tag information was extracted from a symbol file and then cached.
	Device tag information is currently being received from a connected device.
	Connection to device has failed. Tag information might not be up-to-date.

To refresh the device tag information, either by rereading the symbol file or by trying to reconnect to the device, click **Refresh Devices** in the "Object Finder". All configured devices are refreshed at the same time.

Changes in the device tag information might break tag references in the project. To find broken tag references, verify the project (**WinStudio ▸ Verify application**).

Synchronizing the project tags database with the device

If a target device is added to the project, the development application reads the device symbol file and adds corresponding tags to the project tags database. After that, the database has to be synchronized to reflect any changes in the device tags. Proceed as follows to synchronize the database:

1. If the **Automatic Synchronization** option is selected for the target device, the database is automatically synchronized whenever the project is verified (**Verify** on the Home tab of the ribbon) or the tags are browsed using the Object Finder.

   In this case, WinStudio gets a list of the current tags on the target device and compares it to the tags database. The "Database Synchronization" dialog is displayed, listing any tags that have been deleted from the device since the last sync.

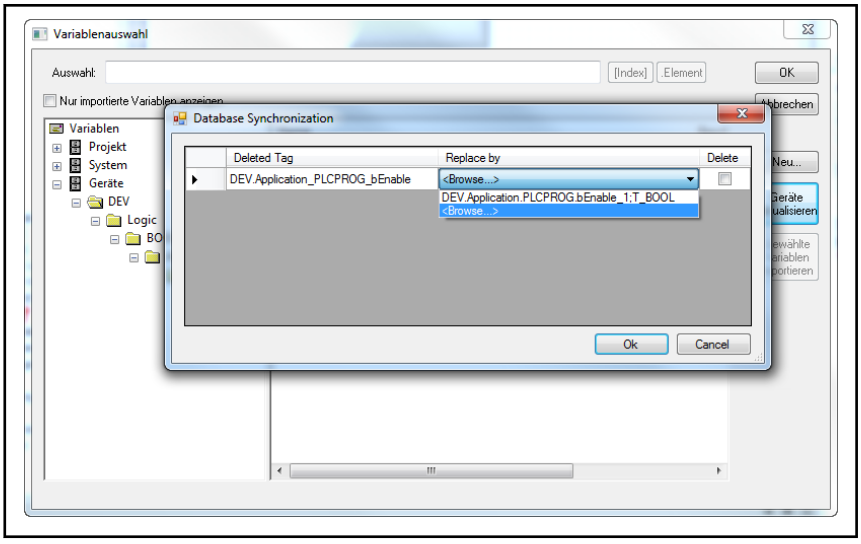


Fig. 7-49: Database Synchronization dialog

For each tag in the list, either a tag from the project can be deleted or all respective tags (in objects and worksheets) can be replaced by a new tag:

- Proceed as follows to replace a tag:
  1. Click on the **Replace by** column for this tag.

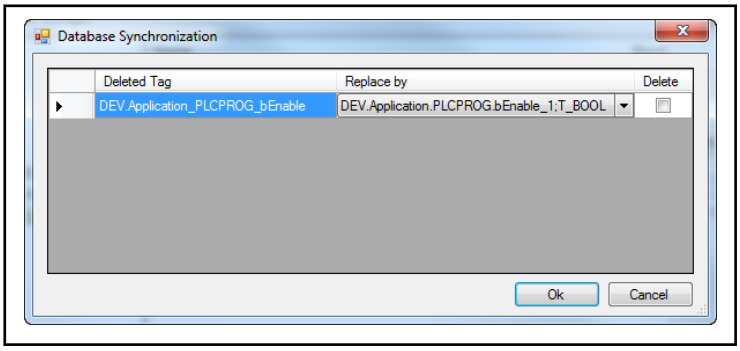


Fig. 7-50: WinStudio, added tags

  2. Select an existing variable from the list or click **Browse** to insert a variable.
  3. Click **OK** to close the "Database synchronization" dialog.
- To delete a tag, click (check) the **Delete** checkbox for that tag. If a tag is deleted from the project while still being used by objects and worksheets, errors will occur the next time the project is verified or executed. The Cross reference list on page 82 tool can be used to identify the use of tags in the project.

2. Click **OK** to apply the changes and close the "Database Synchronization" dialog.

Please note: If a listed tag is not replaced or deleted, the tag is listed again the next time the database is synchronized.

7.11.3 Renaming integrated tags in the project

When WinStudio integrates tags from third-party devices and software, it cannot directly transcribe the tag names. Some changes are made to improve tag management and to adhere to the local tag name syntax.



The full, original name of an integrated tag can always be retrieved by accessing the `DeviceTag` property on the tag.

For example, `tagname->DeviceTag`.

Inserting the tag prefix

First of all, since the project may connect to multiple devices that have the same control program and device tags, WinStudio automatically inserts the tag prefix specified when adding the tag integration source. For example, for a integrated tag named…

```
MyDeviceTAG[1].ClassMember
```

…the corresponding WinStudio project tag is named…

```
tagprefix.MyDeviceTAG[1].ClassMember
```

This allows for the differentiation between similar tags from different sources.

Multidimensional arrays

WinStudio does not support multidimensional arrays. For this reason, if integrated tags have more than one array index, each index after the first will be indicated by `_Index_`. For example, for a integrated tag named…

```
MyDeviceTAG[1][2][3].ClassMember
```

…the corresponding WinStudio project tag is named…

```
tagprefix.MyDeviceTAG[1]_2__3_.ClassMember
```

Nested classes

WinStudio does not support nested classes. For this reason, if integrated tags have more than one class member, each class member after the first will be represented by _ClassMember. For example, for a integrated tag named…

```
MyDeviceTAG.ClassMember.ClassMember2
```

…the corresponding WinStudio project tag is named…

```
tagprefix.MyDeviceTAG.ClassMember_ClassMember2
```

8 Screens and graphics

The ability to show the process state by interacting with instruments (or computers) is described as the Human-Machine Interface (HMI).

With "WinStudio", high-resolution process screens can be created for process monitoring.

The WinStudio graphic tools consist of two modules:

- The "Screen editor" in the "IndraWorks Engineering" development environment (to create or import graphics) and
- the "Viewer" to display WinStudio screens in the runtime environment.

"Animations" can be used to create dynamic graphic objects or symbols. Animations cause objects and symbols to change appearance to display changes in the value of a tag or an expression. Each WinStudio screens usually includes several static and animated objects.

A graphic can be selected as background in WinStudio screens.

For example, the static elements in the following WinStudio screen can be part of a background graphic and animated objects can be used to display changes in the illustrated process. This creates the impression of a 3D screen.

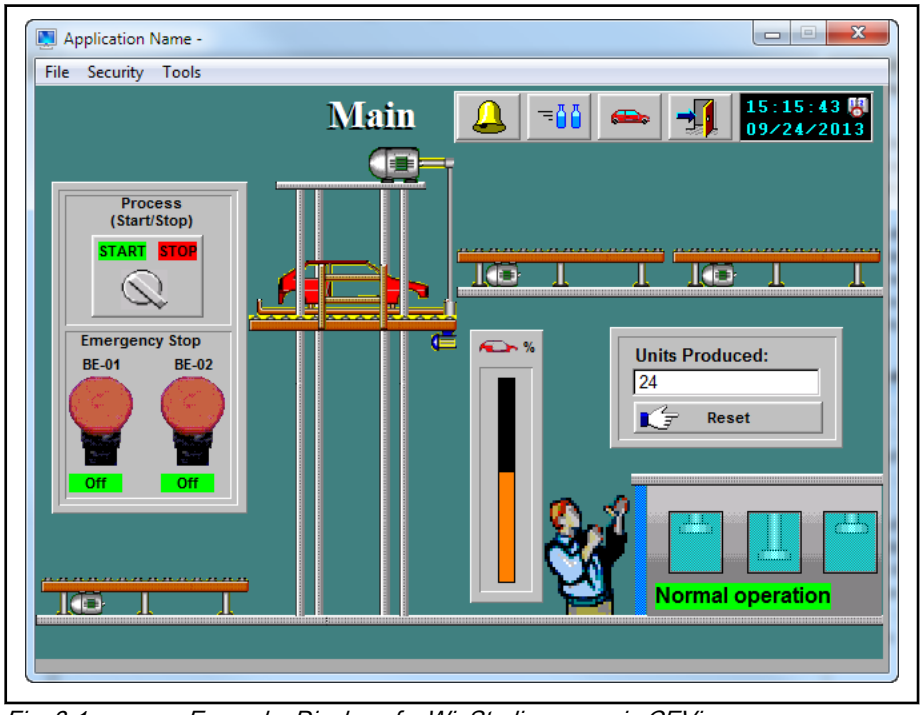


Fig. 8-1: Example: Display of a WinStudio screen in CEView

To configure the WinStudio application, a Windows-compatible pointing device, such as a mouse or touch pad, is required.

The project runtime can also be executed in the “Viewer” without a pointing device if a keypad or keyboard is configured for all commands.

8.1 WinStudio project structure

8.1.1 WinStudio folder

WinStudio folder, general information

Depending on the application type selected on adding the IndraControl V device, the **WinStudio** folder can be found directly under the "VPx-/VEx/VRx" folder or in the "ScreenApplication" subfolder.

The folder **WinStudio** includes all screens, screen groups and symbols in the project.

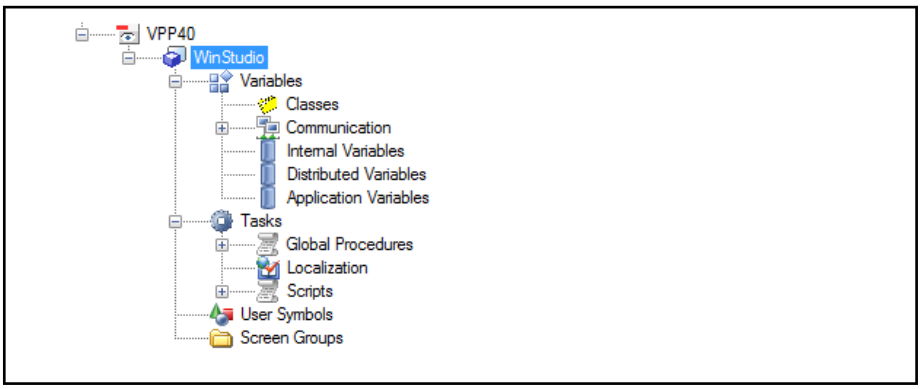


Fig. 8-2: The "WinStudio" folder

The individual folders are described on the following pages:

- **WinStudio** contains all screens on page 170 created for the current project.

  These screens can be structured in a simple list or in user-defined subfolders
- **Variables** contains all functions and editors for editing variable types as well as configuration of process integration of variables into external devices.
- **Tasks** contains different editors and functions required to create more complex applications ("math sheets", "alarms", etc.)
- **Screen groups** contains all screen groups on page 175 (individual WinStudio screens combined into manageable groups) created for the current project.
- **User symbols** contains all user-defined symbols on page 223, which can be groups of screens and/or text. Customized symbols for the project can be created and saved to this folder.
- **Graphics script** contains predefined functions that are executed when certain screen actions occur on page 930, such as when the thin client is started on a remote station.

Screen attributes dialog

The "Screen Attributes" dialog is used to configure runtime settings - such as size, location, title bar, security level, and screen logic - for a specific project screen.

Opening the dialog

To open the Screen Attributes dialog, right-click on the opened WinStudio screen and select "Attributes" in the context menu.

Dialog in detail

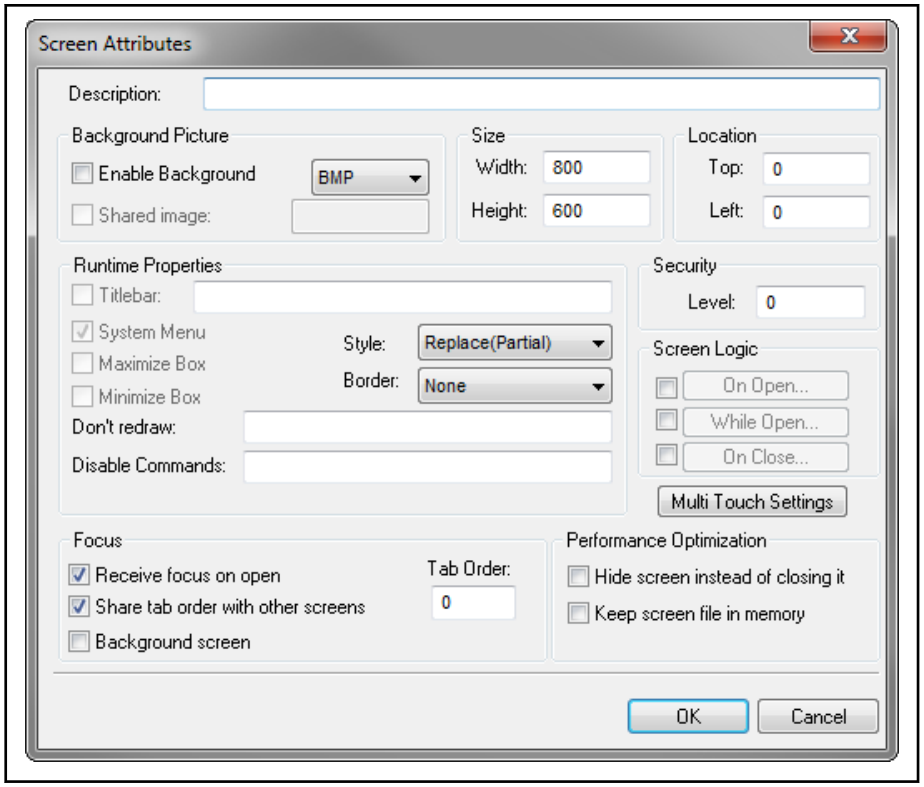


Fig. 8-3: Screen Attributes dialog

"Screen Attributes" dialog elements

Pane/element name		Description
Description		A short description of the WinStudio screen. This is not shown anywhere at runtime
Background picture	**Enable background**	Enables the background picture plane and specifies the file type of the background graphic. If this option is selected, a new BMP file is created with the same name as the WinStudio screen (e.g. `..\Screens\<ScreenName>.BMP`). If required, this screen can be edited with a third-party screen editor. For further information, refer to changing the background color or picture of a screen on page 173.
	Shared image	Uses the stored image file located in the "Screens" subfolder of the project folder. If a tag/expression is to be used instead of a fixed name to change the value during runtime, the tag/expression has to be enclosed in curly brackets. (For example: `{MyTag}` Do "not" include the extension in the file name. If the file format is not BMP, use the list to the right of the **Enable background** option above to select the correct format **Note**: Only BMP files are supported in projects developed for Windows Embedded target systems.
Size	**Width**	The default width of the screen (in pixels) when it is initially displayed. The user can change the size during runtime if the screen is set to be resizable; see **Border** below.
	Height	The default height of the screen (in pixels) when it is initially displayed. The user can change the size during runtime if the screen is set to be resizable; see **Border** below.

Pane/element name		Description
Location	Top	The default distance (in pixels) between the top of the computer display and the top of the screen when the screen is initially displayed. The user can change the size during runtime if the screen is set to be resizable; see **Border** below.
	Left	The default distance (in pixels) between the left edge of the computer display and the left edge of the screen when the screen is initially displayed. The user can change the size during runtime if the screen is set to be resizable; see **Border** below.
Runtime properties	Style	Influences the general runtime behavior of the screen: **Overlap**: Opens the screen without closing any other screens. **Popup**: Opens the screen before all other screens as dialog, however, does not close them **Replace (partial)**: Opens the screen and closes all other Replace and Popup windows. This is set by default **Dialog**: Similar to **Popup** except that the other screens are also disabled until the dialog is closed. **Replace (complete)**: Similar to **Replace (partial)** except that it also closes overlapped windows and dialogs.
	Border	The screen border type: **None**: No border; the screen is a flat, immovable rectangle on the computer display. This is set by default **Thin**: A thin border that makes the screen a movable window. Includes a title bar **Resizing**: A thick border that makes the screen a movable, resizable window. Includes a title bar
	Title bar	Shows the window's title bar with the specified window name If a tag/expression that provides this value is to be specified to programmatically change the value during runtime, the tag/expression has to be enclosed in curly brackets. (For example: `{MyTag}` **Tip:** It is useful to specify a window name even when the title bar is not shown, because when the screen is printed, the window name is included in the page header
	System menu	Provides a menu of basic window commands at the left end of the title bar
	Maximize box	Shows the Maximize button at the right edge of the title bar
	Minimize box	Shows the Minimize button at the right edge of the title bar
	Don't redraw	If this tag/expression is set to TRUE, the screen is not updated
	Disable commands	If this tag/expression is set to TRUE, the screen is locked against user action but the graphics continue to be updated
Security		The minimum security level that a user must have to access this screen.

Pane/element name		Description
Screen logic	On open	Commands and expressions to be evaluated when the screen is opened, similar to a math worksheet
	While open	Commands and expressions to be evaluated when the SCREEN is opened, similar to a math worksheet. If a tag/expression is configured in **Trigger**, the listed expressions are evaluated once each time the trigger variable changes while the screen is open
	On close	Commands and expressions to be evaluated when the screen is closed, similar to a math worksheet
Multi touch settings		Customizes the multi touch settings for this screen. For further information, refer to Multi touch settings on page 286 for project screens.
Focus	Receive focus on open	When the screen is opened, the focus automatically goes to the first object in the screen (according to object ID) that can receive focus, as if the screen was selected using the tab key
	Share tab order with other screens	When the last object in the SCREEN is left with the tab key the focus goes to the next open screen (according to **Tab order** below) rather than back to the first object in the current screen
	Tab order	Similar to object ID for screen objects, this option determines the tab order between screens when multiple screens are open. After tabbing through the last object in a screen, the focus goes to the open screen with the next higher tab order number. Each screen is to have a unique tab order number between 0 and 32767
	Background picture	If the user clicks on the screen, it remains in the background and is not displayed in front of the other open screens. If this option is activated for more than one screen, the screens are arranged in tab order. The screen with the highest number in the tab order is furthest back.
Performance optimization	Hide screen instead of closing it	If the screen is closed (by a user action or a system process), it is actually only hidden. When the screen is reopened it is displayed again. This makes it seem like the screen opens very quickly. **Note**: This option is to be selected for critical screens only. If too many screens remain open, whether visible or hidden, the overall runtime performance is affected.
	Keep screen file in memory	If the screen is closed, the screen file remains in the memory so that it does not have to be loaded from the hard drive when the screen is reopened. The screen must, however, still be redrawn. This process is not as fast as when a hidden screen is made visible again (see above). However, the screen still seems to open quickly. **Note**: This option is to be selected for critical screens only. If too many project files are retained in memory, the overall runtime performance is affected.

Background color or picture of a WinStudio screen

A WinStudio screen can have either a solid background color or an editable background picture.

Selecting the background color

By default, a newly created screen has a solid white background.

To change this background color, proceed as follows:

1. Open the screen to be edited in the editor

2. Open the context menu of the screen by right-clicking on a blank spot of the screen. Select the **Background Color** entry. A standard color selection field is displayed as context menu
3. Use the color picker to select a color. The color applies for the entire background of the screen



If a background color is only to be set for a part of a screen, draw a shape object on page 181 and send it to the background on page 269.

Activating the background picture

To use a background graphics for a WinStudio screen and to edit it, proceed as follows:

1. Open the WinStudio screen in the editor
2. Open the context menu of the screen end select the **Screen Attributes** entry. The Screen Attributes dialog on page 170 is displayed
3. Select the **Enable Background** option

   A new BMP file with the same name as the WinStudio screen is automatically saved in the "Screens" subfolder of the project folder

   (e.g. `..\Screens\<ScreenName>.BMP`)
4. Click **OK** to close the "Screen attributes" dialog
5. In the main menu, open the entry **WinStudio screen** ▸ **Edit** ▸ **Background screen**.

   Microsoft Paint is started and the BMP file is opened for editing

To use an image editor other than Microsoft Paint, manually edit "Program Settings.ini" by adding the following setting:

```
[Options]
ImageEditor=filepath
```

Existing graphics file as background

To select an existing graphics file, especially if it's in a format other than BMP, proceed as follows:

1. Copy the graphics file to be used into the "Screens" subfolder of the project folder
2. Open the respective WinStudio Screen in the development environment
3. Select the entry **Screen Attributes** in the context menu of the screen. The "Screen Attributes" dialog is displayed
4. Select **Enable background** and then select the format of the image file in the list to the right of the option.



Only BMP files are supported in projects developed for Windows Embedded target systems.

5. Select the **Shared Image** option, and in the box to the right of the option, type the name of the image file. The file extension is not entered since it is determined by the list selection in the previous step.

   It is possible to specify folders in the image folder. If, for example, `MyBackgrounds\Background1` is entered, the program searches for the image file in `[…]\Projectname\Screen\MyBackgrounds\Background1`.

   It is also possible to enter a tag or expression in curly brackets (e. g. `{MyTag}`) to have the program change the background image during runtime.

6. Select the graphics format of the graphics file from the list
7. Click **OK** to close the "Screen attributes" dialog. If the development application can find and load the specified graphics file, it is displayed in the WinStudio screen. If it cannot be opened, a warning message is displayed

8.1.2 Screen groups folder

The "Screen Groups" folder contains elements that combine several screens on page 170 to groups.

To open a specific screen group, open the "Screen Group" folder and double-click on the element.

To remove a specific screen group, right-click on the respective element and select "Delete" in the context menu.

To create a new screen group, proceed as follows:

1. Open the context menu of the "Screen Groups" folder and select **New screen group**. A new screen group is created.

   Open the context menu of the screen group. Click on "Open" or double-click on the screen group to open the configuration dialog for the screen group.

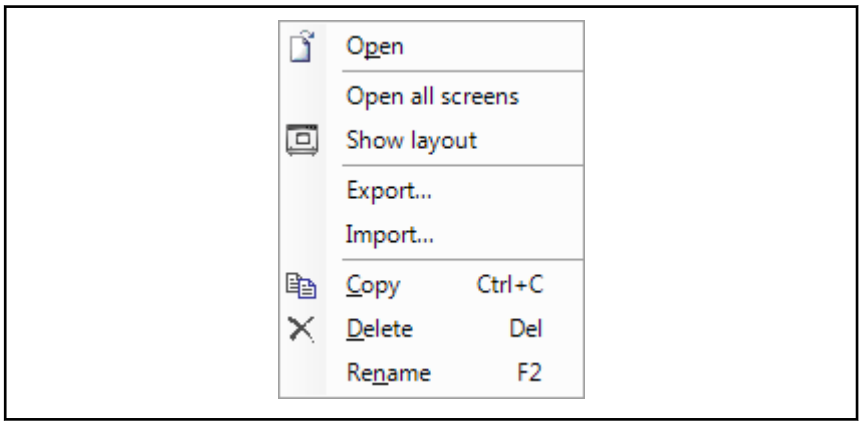


Fig. 8-4: Context menu of the screen group

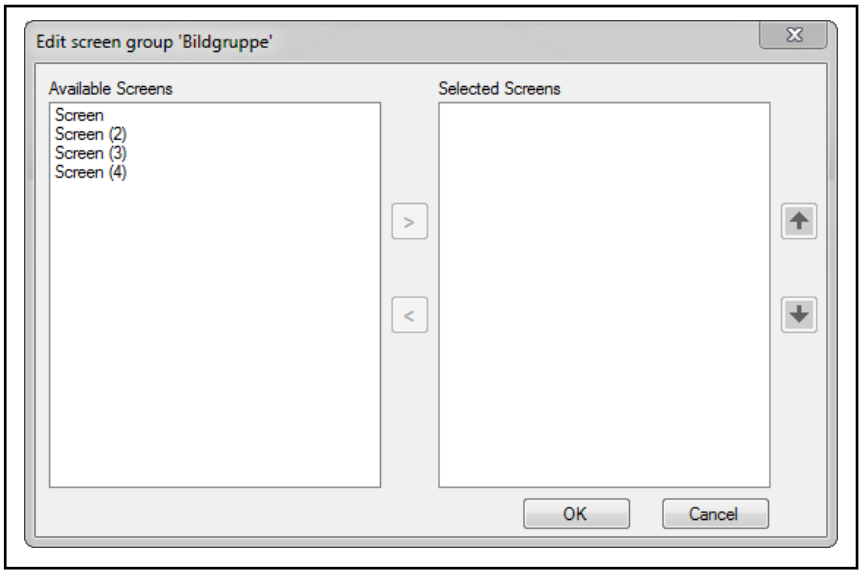


Fig. 8-5: Creating a screen group

2. Select the screens from the list "Available screens" to be included into the screen group.

   The list contains all screens of the current WinStudio project.

   To select multiple screens, press the <Ctrl> key while clicking on the screen names.

   Release the <Ctrl> key when all screens are selected.
3. Click on **>** to add the screens to the screen group.

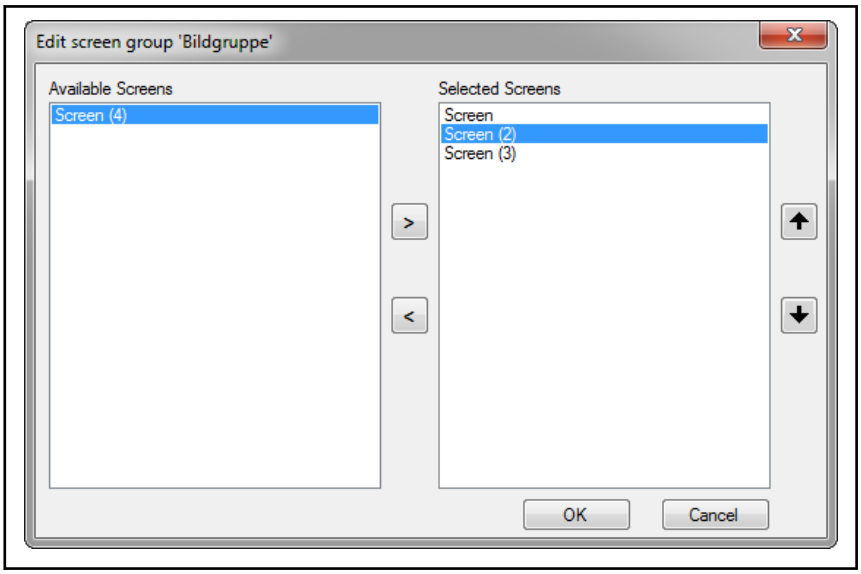


Fig. 8-6: Editing a screen group

4. Click **OK** to close the "Edit screen group" dialog.

Dialog to configure the screen group

Elements	Description
 *Fig. 8-7: Adding screens to the screen group*	The screens selected under "Available screens" are added to the screen group.
 *Fig. 8-8: Removing screens from the screen group*	The screens are removed from the screen group.
 *Fig. 8-9: Screen to top*	The screen in the screen group is moved one up.
 *Fig. 8-10: Screen up*	The screen in the screen group is moved one down.

Tab. 8-1: Table of the elements in the "Screen group" dialog

Open the context menu of the screen group and select "Open all screens". All screens configured in this screen group are now opened.

Open the context menu of the screen group and select "Show layout". The layout window is opened. It shows all open screens as preview.

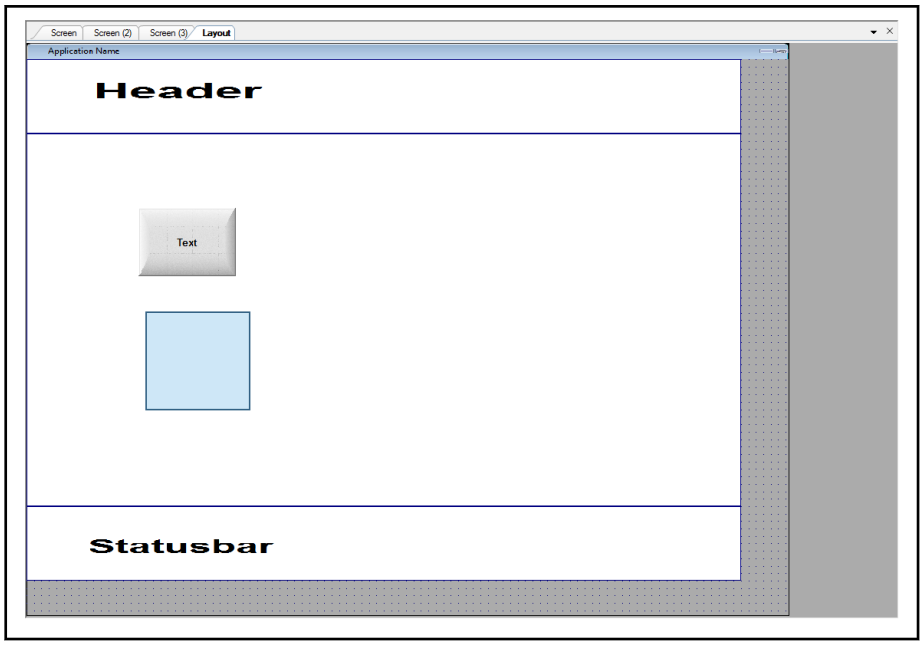


Fig. 8-11: Layout window

The context menu item "Show layout" can also be found in the context menu **WinStudio ▸ Tools**.

8.1.3 Layout tool

Click the **Layout** tool to open the **Layout** tab.

This interface displays all screens currently open in the "Screen editor" and allows to:

- **Modify the Screen Attributes**: Right-click on the screen displayed on the "Layout" tab and use the alignment options or the **Screen attributes** link to modify the screen position. You can also click and drag the screen to change its position (Top and Left) or resize it (Width and Height).
- **Visualize how the screens fit together during runtime.** This option is especially useful when creating pop-up/dialog screens or groups of screens.

The screens open in the "Layout" tab according to the order that they are open in the development environment. When you change the position of the screen tabs in the development environment (to the left or to the right), you will change the order in which these screens will be displayed in the "Layout" tab.

Right-click on the title of the "Layout" tab to display the option to enable/disable the **Auto Scale**. If you enable this option, the screens will be auto-scaled automatically to fit in the "Layout" tab.

8.2 Using screen objects and animations

8.2.1 Editing screen objects

Editing screen objects, general information

To edit WinStudio screens, the toolbars offer numerous functions in addition to the functions in the context menu of the screen editor.

To display or hide toolbars, right-click on the tool bar area and enable or disable the required toolbars.

All functions available in the WinStudio toolbars can also be found in the IndraWorks main menu under the " WinStudio Screen" menu item.



Fig. 8-12: WinStudio toolbars

Editing Object Properties

The Object Properties dialog displays the configurable properties of a screen object or animation. Each type of screen object and animation has its specific properties, but all types are displayed in the same basic dialog.

Opening the dialog

To open the "Object Properties" dialog, select one of the following options:

- Select the screen object and press <Alt>+<Enter>
- Right-click the screen object and click **Properties** in the context menu
- Double-click on the screen object.

Dialog in detail

All objects have the following properties in common:

	Description
Pin needle	If the pin needle is pressed, the "Object properties" window remains opened as long as the screen editor is open. In this mode, the properties of all screen objects can be subsequently displayed or edited without having to close and reopen the dialog
Replace	Opens the "Replace" dialog to replace strings, tags or properties on page 178 for the selected object or group of objects
Note	If the cursor is moved over the object during runtime, a tooltip is displayed. This can be used to provide quick support The text in the **Hint** field is also written temporarily to the System tag on page 108 "Hint" so that an action according to this tag value can be triggered by moving the cursor over a specific object. To display Hints/ToolTips during runtime, select the option **Enable ToolTips** in the "Project settings" dialog. This function can be activated/deactivated separately for the standard viewer (on the **Viewer** tab of the WinStudio Project Settings) and for Web Thin Clients (in the **Web settings** dialog)
Button	By means of the combo box at the right of the dialog, specific objects within a group of objects or animations can be selected for editing

Replacing Project Tags in a Document or Screen Object

To replace all tags in a screen object, double-click the object to open its "Object Properties" dialog and then click **Replace**.

To replace all tags in the current document, carry out one of the two following actions:

- Open the "WinStudio Screen" menu in the IndraWorks main menu and select the entry **Replace tags and strings in the current document**
- Display the "WinStudio Table Functions" toolbar and click on the **Replace tags and strings in the current document** button.

In both cases, the "Replace" dialog is displayed as described below.

One or more tags can be replaced by clicking the **Whole Tag Name** tab. The left column, **From**, displays the tags currently used. Into the right column, **To**, the new names can be entered.

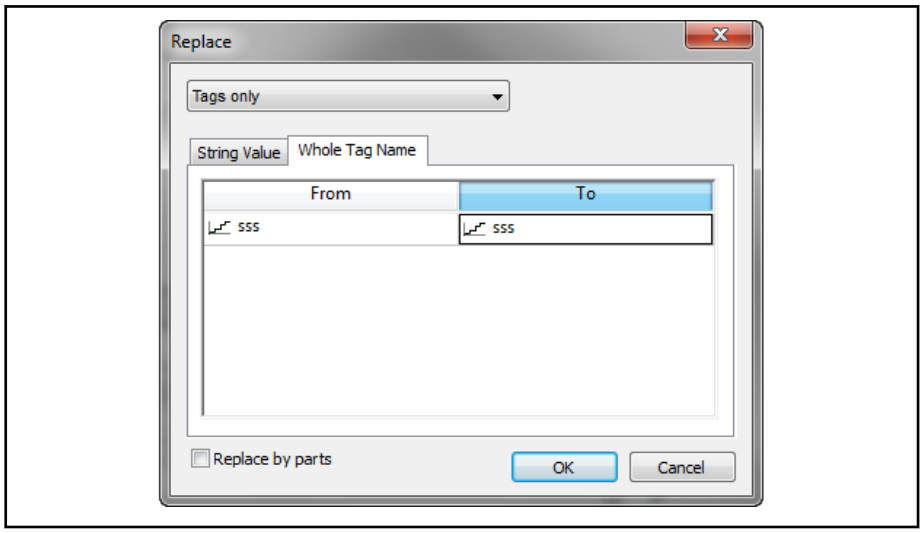


Fig. 8-13: "Replace" dialog, "Whole Tag Name" tab

Note that this does not "rename" or "delete" any tag. Tags used in the object are only replaced by other tags from the database.

By activating the "Replace by parts" option, replacing parameters (e.g. array index) within classes and arrays becomes easier.

One or more strings can also be replaced (e.g., button captions, descriptive texts) by clicking the **String Value** tab.

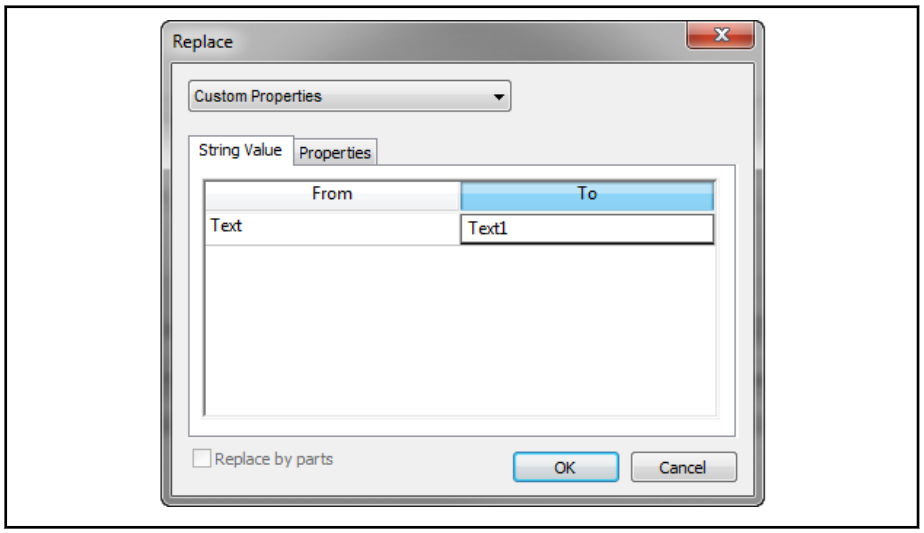


Fig. 8-14: "Replace" dialog, "String Value" tab

After editing all entries, click **OK**.

Arranging objects

The "WinStudio Alignment Functions" and "WinStudio Object Properties" toolbars offer numerous functions to align, group and design screen objects.



Fig. 8-15: "Alignment Functions" toolbar



Fig. 8-16: "Object Properties" toolbar

Some functions in the toolbars are only available if one or multiple objects have been selected in the WinStudio screen.

Move the cursor over an active "tool" to display the function in plain text.

The tools are organized into the following groups:

- **Arrange**: Arranging objects in a WinStudio screen including
  - Move to foreground and Move to background on page 269
  - To the foreground and To the background, by one level on page 269
  - Group and ungroup objects on page 270
  - Align, Center and Distribute on page 271
  - Rotating and mirroring on page 273
- Adjusting position on page 274
- **Size**: Same width / same height for selected objects on page 275 of a screen object in a WinStudio screen
- **Style**: Changing the color of screen objects
  - Fill color on page 276 of a screen object
  - Line color on page 276 of a screen object
- **Edit the color of a component**: Color Interface on page 277
- **Fonts**: Changing Fonts on page 280 of a screen object

Locking Objects in the Screen Editor

The function for moving objects in the screen editor can be locked to prevent accidental position changes of screen objects after setting the final screen layout.

- In the WinStudio screen editor, open the context menu and select the entry **Lock moving**
- To allow moving screen objects, execute the function **Lock moving** again

Optionally, the "Lock moving" function can be activated and deactivated by pressing <Ctrl>+<D>.

Undo

In the WinStudio screen editor, the last action can be undone by selecting **Undo** in the "Standard" toolbar.

Optionally, press <Ctrl>+<Z>. This way, up to 20 actions before the last action can be undone.

(Actions in the "WinStudio Object Properties" toolbar are not included in the **Undo** count, i.e. adjusting these settings cannot be undone with this function.)

Grid Settings

To display/hide the grid in the screen editor, click on **Activate/deactivate grid** in the "WinStudio Object Properties" toolbar.

To edit the grid settings, open the "Grid" dialog. Right-click in a free area in the screen editor and click **Grid Settings** in the context menu.

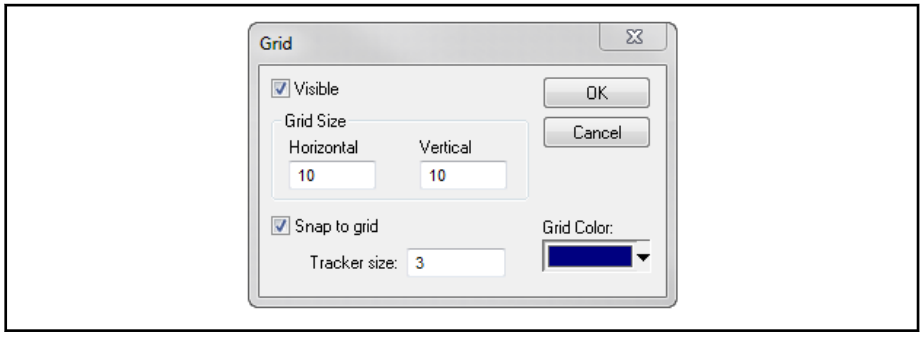


Fig. 8-17: Grid settings dialog

8.2.2 Static objects

Static objects, general information

After opening the WinStudio editor, the "WinStudio Screen tools" tool window is available.

The tool window includes various objects that can be inserted into a WinStudio screen: Rectangles, polygons, lines, text fields, etc.

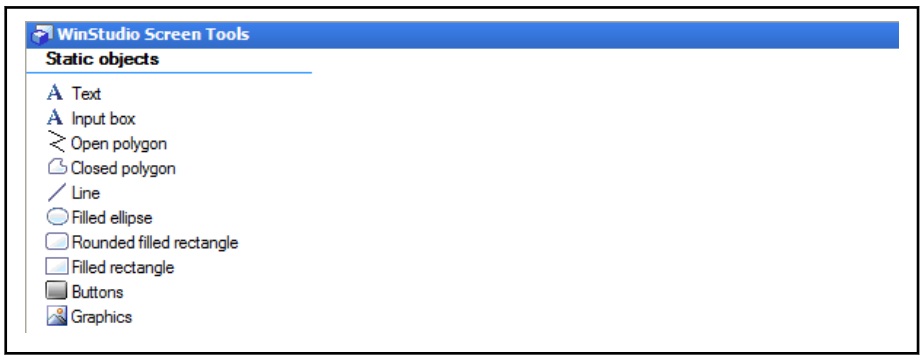


Fig. 8-18: "WinStudio Screen tools" tool window

Line Object

To draw an orthogonal line, click on the **Line** entry in the tool window.

Afterwards, click into the drawing area of the WinStudio screen editor:

1. Left-click to set the starting point of the line
2. Drag the line with the cursor to adjust the line size
3. To align the line, drag at one of the marked end points
4. To display the object properties, double-click on the object

The "Object Properties" dialog is displayed as follows:

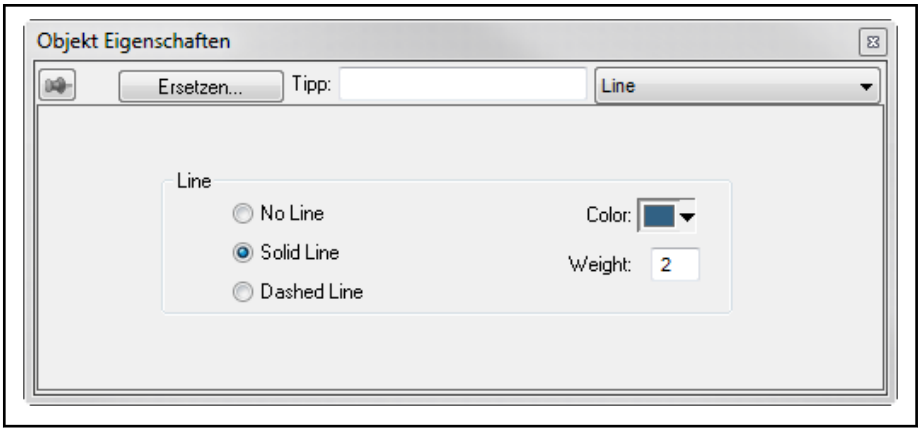


Fig. 8-19: Object Properties: Line

Use the "Object Properties" dialog to specify the following parameters for the line:

- **Line**: Specify a line style by clicking the **No Line**, **Solid Line** or **Dashed Line** button
- **Color**: Specify a line color by clicking the **Color** button. In the "Color" dialog, click on a color to select it and close the dialog
- **Weight**: Specify the line width (in pixels) by entering a number representing the line width into the text field

Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

Open polygon object

In the **WinStudio Screen Tools** tool window, click on **Open polygon** to draw an open polygon with frame in the specified foreground color.

To draw an open polygon in the drawing area, proceed as follows:

1. Left-click to set the starting point of the polygon
2. Move the cursor to a new location and click again to place the second point
3. Repeat this process until the desired polygon is created
4. Double-click to stop drawing the polygon

   To change the shape of a polygon after drawing it, select it and drag any of its points

If a polygon's individual points cannot be dragged, they may be grouped page 270. To ungroup the points, right-click on the polygon and choose **Ungroup** from the context menu

To display the object properties, double-click on the object. The "Object Properties" dialog is displayed as follows.

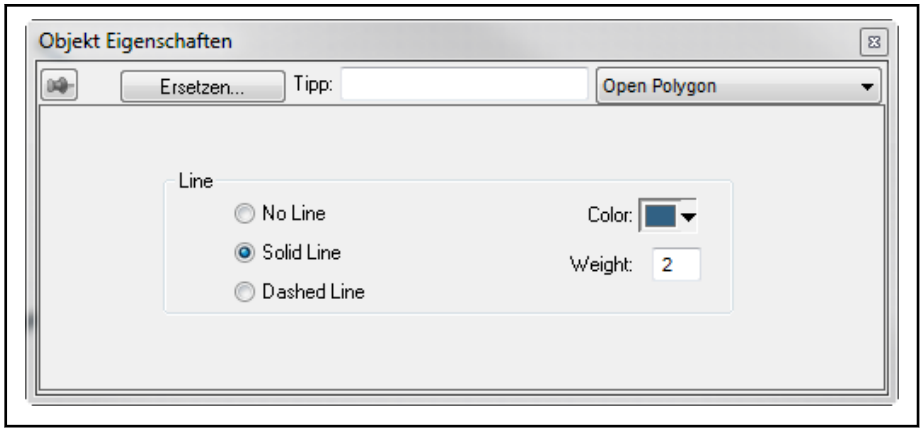


Fig. 8-20: Object Properties: Open Polygon

Use the "Object Properties" dialog to specify the following parameters for the polygon:

- **Line**: Specify a border line style by clicking the **No Line**, **Solid Line**, or **Dashed Line** button
- **Color**: Specify a border line color by clicking the **Color** button. In the Color dialog, click on a color to select it and close the dialog
- **Weight**: Specify the border line width (in pixels) by entering a number specifying the line width into the text field

Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

Closed polygon object

In the **WinStudio Screen Tools** tool window, click on **Closed polygon** to draw a closed polygon with frame in the specified foreground color.

To draw a closed polygon in the drawing area, proceed as follows:

1. Left-click to set the starting point of the polygon
2. Move the cursor to a new location and click again to place the second point
3. Repeat this process until the desired polygon is created
4. Double-click or right-click to stop drawing the polygon
5. To display the object properties, double-click on the polygon object

   To change the shape of a polygon after drawing it, select it and drag any of its points

If a polygon's individual points cannot be dragged, they may be grouped page 270. To ungroup the points, right-click on the polygon and choose **Ungroup** from the context menu

The "Object Properties" dialog is displayed as follows.

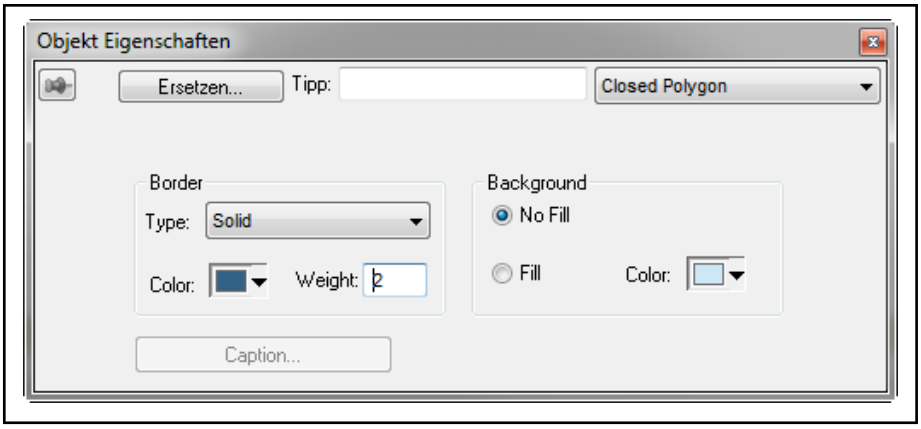


Fig. 8-21: Object Properties dialog: Closed Polygon

Use the "Object Properties" dialog to specify the following parameters for the polygon:

- **Type**: Specify a border line style by clicking the **No Line**, **Solid Line**, or **Dashed Line** button
- **Color**: Specify a border line color by clicking the **Color** button. In the "Color" dialog, click on a color to select it and close the dialog
- **Weight**: Specify the border line width (in pixels) by entering a number specifying the line width into the text field

---

☞ Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

- **Fill effect**: To specify whether the polygon is filled, click **No Fill** or **Fill**

  If the **Fill** option is selected, choose a fill color by clicking on the **Color** button In the "Color" dialog, click on a color to select it and close the dialog

Rectangle object

To create rectangles, click on **Filled rectangle** in the **WinStudio Screen Tools** tool window and proceed as follows:

1. Click in the drawing area and drag the mouse/cursor to draw the rectangle
2. Release the mouse button when the rectangle has the desired size
3. Double-click on the object to open the "Object Properties" dialog.

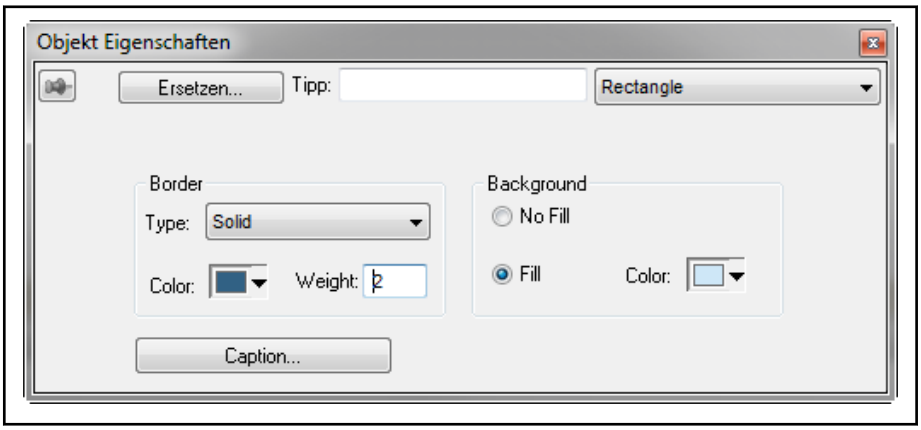


Fig. 8-22: Object Properties: Rectangle

Use the "Object Properties" dialog to specify the following parameters for the rectangle:

- **Type**: Specify a border line style by selecting **None**, **Solid**, **Dashed**, **Etched**, **Raised** or **Sunken**
- **Color**: Specify the border line color by clicking the **Color** button to open the "Color" dialog. Select a color and close the dialog
- **Weight**: Specify the border line width (in pixels) by entering a number into the respective text field



Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

- **Background**: To specify the background color of the rectangle, select **No Fill** or **Fill**

  If the **Fill** option is selected, specify a fill color by clicking on the color rectangle. In the Color dialog, click on a color to select it and close the dialog
- **Caption**: Click on this button to open the "Caption" dialog and to enter a display text

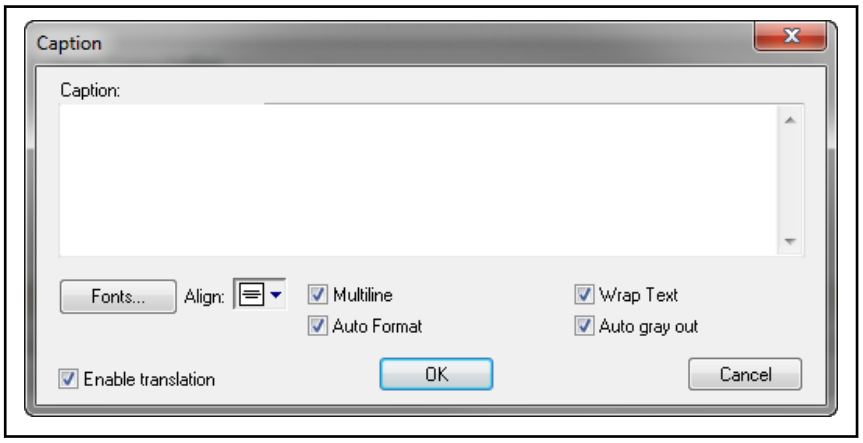


Fig. 8-23: Caption dialog

  - **Caption**: Enter the text to be displayed inside the rectangle object. Tags can be included in curly brackets (e.g. **{tagname}**)
  - **Fonts**: Specify a font style for the text by clicking the **Fonts** button
  - **Align**: Specify the alignment for the caption of the rectangle
  - **Multiline**: Activate this option to display the caption of the rectangle in more than one line
  - **Auto Format**: If this option is enabled and if the caption includes a decimal value in curly brackets (e.g. **{1.2345}**) or a tag type Real on page 111 (see **Caption** above), the value is formatted according to the virtual table created by the function SetDecimalPoints on page 767
  - **Wrap Text**: If enabled, the object automatically wraps the text when necessary
  - **Auto gray out**: The caption is grayed out at runtime if the command animation on page 246 applied to the rectangle is disabled by means of the **Disable** field or due to the security system

– **Enable translation**: If this option is enabled (activated), translation is enabled during runtime by means of the translation tool on page 487

Rounded Rectangle Object

To draw rounded rectangles, click on **Rounded Rectangle** in the **WinStudio Screen Tools** tool window and proceed as follows:

1. Click in the drawing area and drag the mouse/cursor to create the rectangle
2. Release the mouse button to stop drawing the object
3. Double-click on the object to open the "Object Properties" dialog.

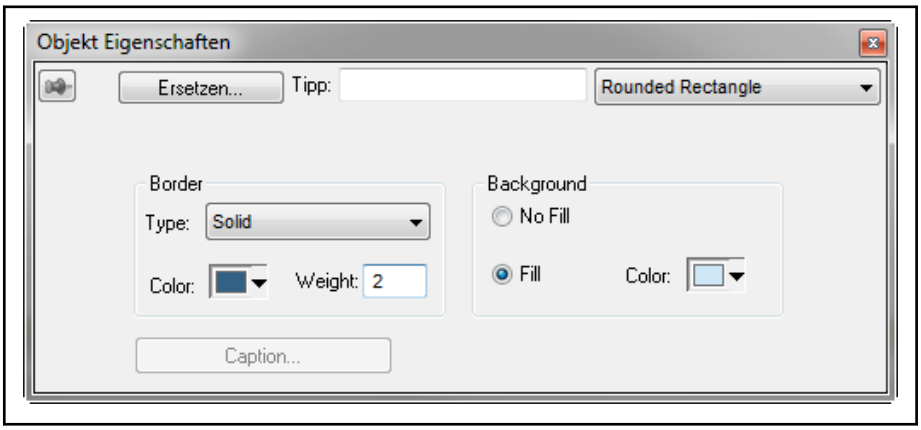


Fig. 8-24: Object Properties: Rounded rectangle

The "rounded rectangle" element cannot be used to create bar graphs for projects running on Windows Embedded target systems.

Rounded rectangles have one extra handle in the bottom-right corner to modify the arc angle.

Use the "Object Properties" dialog to specify the following parameters for the rectangle:

- **Line**: Specify a border line style by clicking the **No Line**, **Solid Line**, or **Dashed Line** button
- **Color**: Specify the border line color by clicking the **Color** button to open the "Color" dialog. Click the color to select it and close the dialog
- **Weight**: Specify a line width by entering a number (in pixels) into the text field provided

Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

- **Background**: Specify the background color of the rectangle by clicking **No Fill** or **Fill**. If the **Fill** option is selected, choose a fill color by clicking on the **Color** button

  In the "Color" dialog, click on a color to select it and close the dialog
- **Caption**: This option is not enabled for this object

Ellipse Object

To draw ellipses, chords, arcs and rings, click on **Ellipse** in the **WinStudio Screen Tools** tool window:

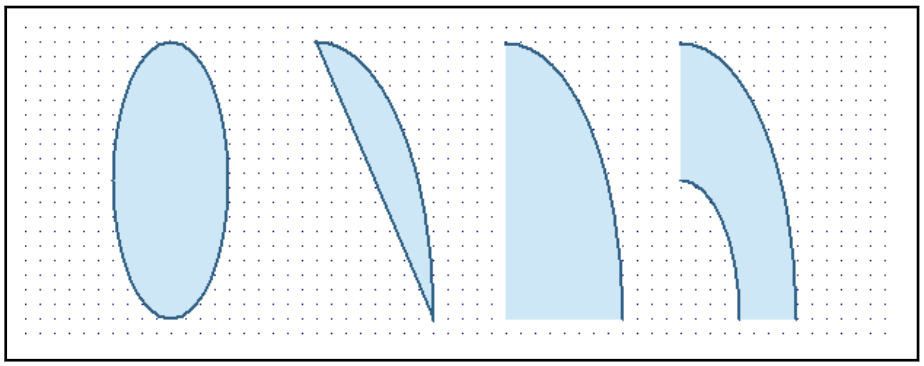


Fig. 8-25: Ellipse, chord, arc and ring



The "Ring" style is particularly useful to create installation drawings.

To create an ellipse, proceed as follows:

1. Click in the drawing area and drag the mouse/cursor to create an ellipse shape
2. Release the mouse button to stop drawing the ellipse
3. Use the "Object Properties" dialog to change the shape to a chord, arc or ring
4. Double-click on the object to open the "Object Properties" dialog.

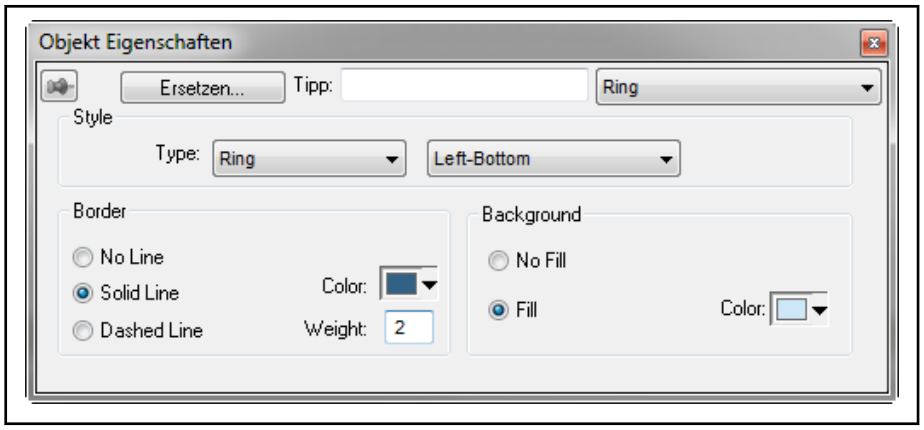


Fig. 8-26: Object Properties: Ellipse

Use the "Object Properties" dialog to specify the following parameters for the ellipse:

- **Style**: Specify the object style by selecting **Ellipse**, **Arc**, **Chord**, or **Ring** from the drop-down list

  For the objects "arc, chord or ring", additionally specify the quadrant for illustration (bottom left, top left, bottom right, top right)

  For example, to represent a half-circle pipe, create two **ring** objects. Specify one as **Left-Bottom** and the other as **Right-Bottom** then join the two objects to create a half-pipe
- **Border**: Specify a border line style for the ellipse by selecting **No Line**, **Solid Line** or **Dashed Line**

- **Weight**: Specify a line width by entering a number (in pixels) into the text field provided



Dashed lines/borders of weight greater than 1 are not supported in projects running on Windows Embedded.

- **Background**: To specify the background color of the ellipse, select **No Fill** or **Fill**

  If the **Fill** option is selected, choose a fill color by clicking on the **Color** button

  In the "Color" dialog, click on a color to select it and close the dialog

Adding graphics to a WinStudio screen

To insert a graphic into a WinStudio screen, copy the graphic to the clipboard and then paste it directly into the WinStudio screen.

This task assumes that a WinStudio screen to edit in the editor is open.

Please note that using this method to add a graphic to the WinStudio screen, the graphic file is embedded in the WinStudio screen by default. Thus, all objects are saved in one single file, which is more convenient in some cases. However, the size of the WinStudio screen file increases every time the graphic is used multiple times in one WinStudio screen. Additionally, the method is less flexible than linking an external graphic file. To still use this method, continue with the sequence described in the following.

There are two options to save a graphic to a separate file and to use it as linked graphic object:

1. **Save all graphics to separate files**: To link all graphics as external files, enable the "Save images to separate files" option first in the project settings. If this option is selected and a graphic is inserted into a WinStudio screen via the clipboard, the graphic is automatically saved as a separate file in your project folder and then inserted into the WinStudio screen as Linked picture on page 242. For further information on the "Save images to separate files" option, see Project settings: Viewer, on page 61.



To subsequently enable the option "Save images in separate files" and to save all already embedded graphics as separate files, use the "Verify application" function. One file is created for each embedded graphic. A unique identifier is generated as file name.

For the graphic file generated that way, go to **WinStudio** ▸ **Resources** ▸ **Internal resources**.

2. **Save individual graphic to separate file**: If only one particular bitmap image is to be linked as an external file, do not use the clipboard to insert it into the WinStudio screen. Insert it manually as a Linked picture on page 242 instead. Thus, it is possible to insert and embed other bitmap images according to the default settings.

The options are summarized in the following table:

Action	Save images to separate files...	
	...is selected	...is not selected
Insert graphic into the WinStudio screen	Graphic is inserted as link	Graphic is inserted as bitmap object
Link to an external graphic file	Graphic is inserted as link	Graphic is inserted as link

To insert a bitmap image into a WinStudio screen, perform the following steps:

1. Open the graphic in an image-processing application, such as Microsoft Paint.
2. Select part or all of the graphic, either with **Select** press <Ctrl>+<A>.
3. Copy the selected graphic to the clipboard, either with **Copy** or press <Ctrl>+<C>.
4. Switch to the WinStudio screen editor.
5. Open the context menu of the open WinStudio screen editor and select **Paste** or press <Ctrl>+<V>. The selected graphic is inserted into the WinStudio screen as bitmap object
6. Click on the object and drag it to the location on the screen. If required, use the object handles to modify the size.
7. Double-click on the graphic object. If **Save images to separate files** is selected in the project settings, the graphic was inserted as a Linked picture on page 242.

   Skip the remaining steps and continue with "Link to an external image file".

   If **Save images to separate files** is not selected, the graphic was inserted as bitmap object. Perform the following steps.

   The "Object properties" dialog is displayed.

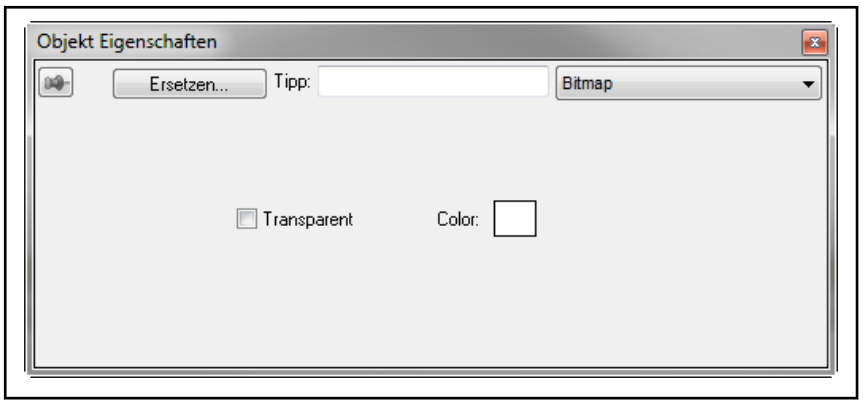


Fig. 8-27: Object properties: Bitmap

8. If parts of the graphic are shown as transparent for the background, select a transparent color:

   8.1 Select **Transparent**.

   8.2 Click and drag the tracker at the graphic object until it is for example above the desired transparent color. The tracker is an additional handle at the graphic object which appears directly in the bottom right corner of the object in the beginning. Moving the

tracker on the object does not move the object itself and does not change its size.

9. Close the "Object properties" dialog

Changing properties of multiple screen objects (static objects)

This task describes how to select two or more screen objects and how to then change the properties the selected objects have in common.

Before starting this task, a project image must be opened in the image editor.

Which properties can be changed depends on whether several objects of the same type or different object types are selected. If the objects are of the same type, you can change the properties applicable to this type. If, for example, several button objects are selected, the properties applicable to button objects can be changed.

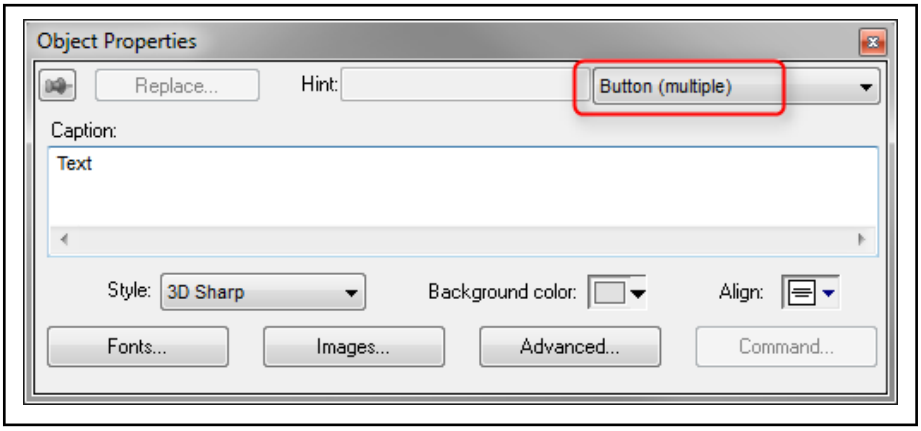


Fig. 8-28: Object properties of multiple selected button objects

For more information on the properties of a specific object type, please consult the documentation for this object



This method can only be used to change the properties of shapes and active objects. The method cannot be used to change the properties of file objects, animations, library elements or objects in a group.

However, if multiple objects of different types are selected, the properties shared by all objects can be changed. This not only includes cosmetic properties such as Border and Background but also functional properties such as Disable, Security, Enable translation and E-Sign. (Some properties may not apply to all objects. Button objects, for example, do not have a border and a rectangle object does not have a security.)

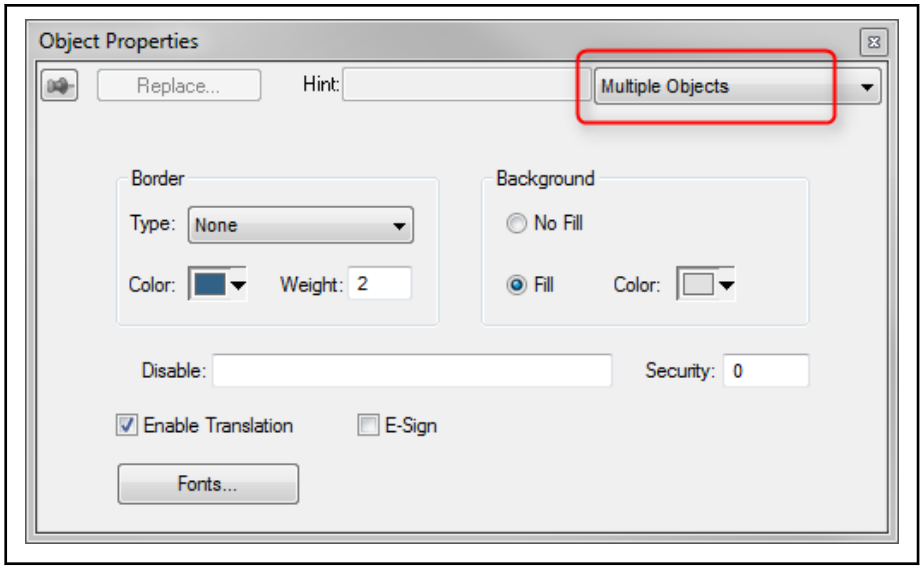


Fig. 8-29: Object properties for multiple objects of different types

In both cases, the dialog box displays the current values of the properties of the most recently selected object.

Only if the value of a property is actually changed, will the change be applied to the selected objects. All other properties remain unchanged independent from the values displayed in the dialog box.

Changing the properties of multiple screen objects:

1. Do one of the following in the screen editor:
   - Either press and hold down <Shift> or <Ctrl> on the keyboard and click on each object that is to be changed; or
   - Use the cursor to draw a selection box around all objects that are to be changed.

   The objects are selected.
2. Do one of the following:

   Right-click on the selected objects and click on **Properties** in the context menu; or

   Press <Alt>+<Enter> on the keyboard.

☞ The Object properties dialog box cannot be opened by a double-click as usual since a double-click cancels the selection.

   The Object properties dialog box is displayed for the selected objects.
3. Change the property values as desired and then close the dialog box.

The changes are applied to all selected objects.

8.2.3 Active objects

Active objects, general information

The "WinStudio Screen Tools" tool window offers the following control elements to create interactive objects.

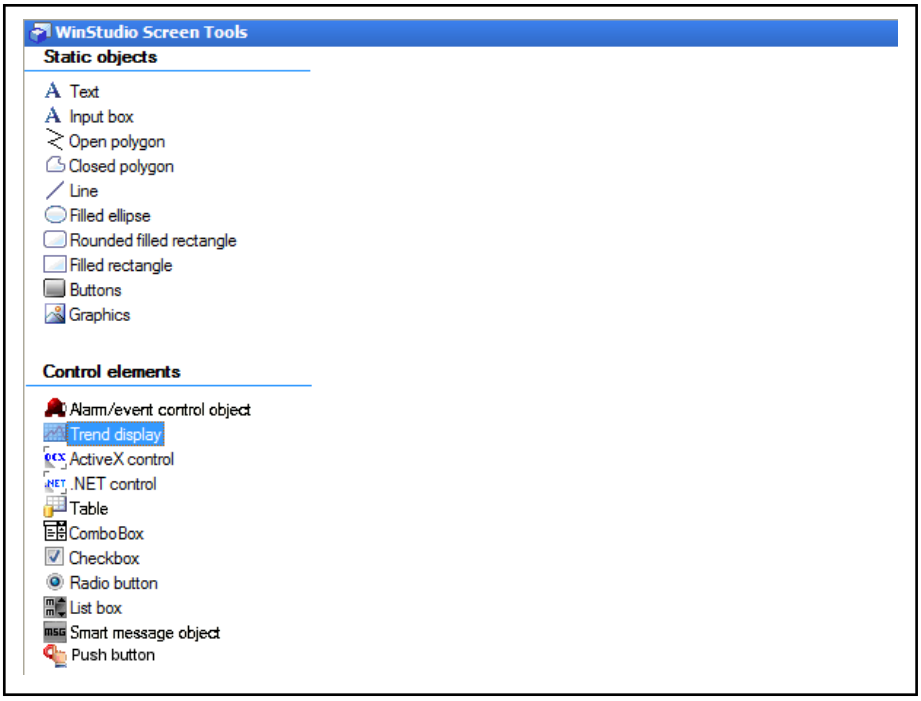


Fig. 8-30: WinStudio screen tools

Text object

To create text objects, click on **Text** in the **WinStudio Screen Tools** tool window and proceed as follows:

1. Click in the drawing area

   Enter a line of text when the cursor is displayed
2. After entering the string, double-click on the new text object to view the "Object Properties" dialog

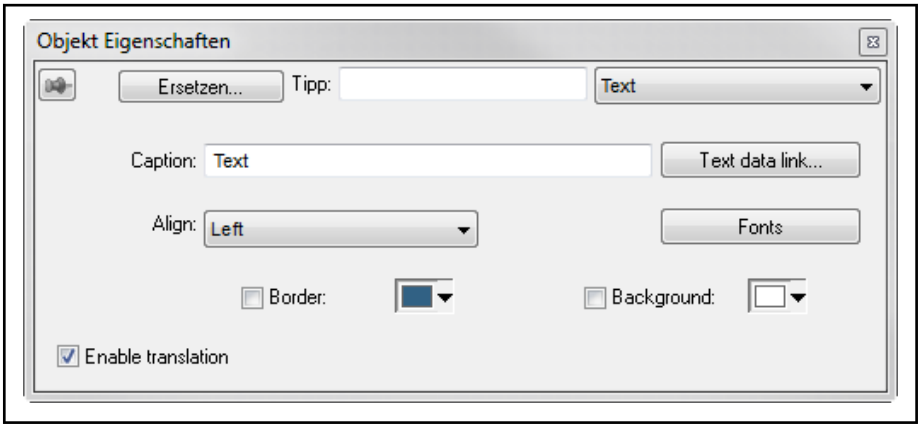


Fig. 8-31: Object Properties: Text

Use the "Object Properties" dialog to specify the following properties:

- **Caption**: Input field to edit the string entered in the "Text" object
- **Text data link** button: Click to apply the text data link animation on page 251 to the "text" object.

  If the caption does not include any placeholder characters ("###") for the text data link, click this button to automatically add those characters
- **Align**: Align the text by selecting **Left**, **Center** or **Right** from the combo box

- **Fonts**: Specify the following text parameters by clicking the **Fonts** button:
  - **Fonts**
  - **Font style**
  - **Size**
  - **Effects**
  - **Color**
  - **Script**
- **Border**: Activate the "Border" checkbox to add a border to the text field and click on the color selection button to the right to select a suitable border color
- **Background**: Specify a background color by clicking on **Background**

  In the "Color" dialog, click on a color to select it and close the dialog
- **Enable translation**: Enable this option to apply the text to the translation file

Draw and configure a Text box object

Draw a Text box screen object with multiple lines to create the possibility to input or output text.

Prior to Rexroth WinStudio v7.3+SP1, the only way to input or output texts on the screen was to draw a Text object and to apply a Text data link animation to it. This was, however, limited to one single text line per object and required additional steps to combine the object, the animation and background or border shapes.

The new Text box object is a field to input/output text which can be configured to display multiple lines. If the object is associated with a String array, each line of the field belongs to an array element: Line 1 is array index 0, line 2 is array index 1, line 3 is array index 2 and so on.

Otherwise, the Text box objects has similar functions as a Text data link animation.

Drawing and configuring a text box object:

1. Open an image for editing.
2. Click on **Input field** in the WinStudio screen tools.
3. Draw the object at the desired position on the screen and move it or change the size, if necessary.
4. Double-click on the object. The Object properties dialog box is displayed.

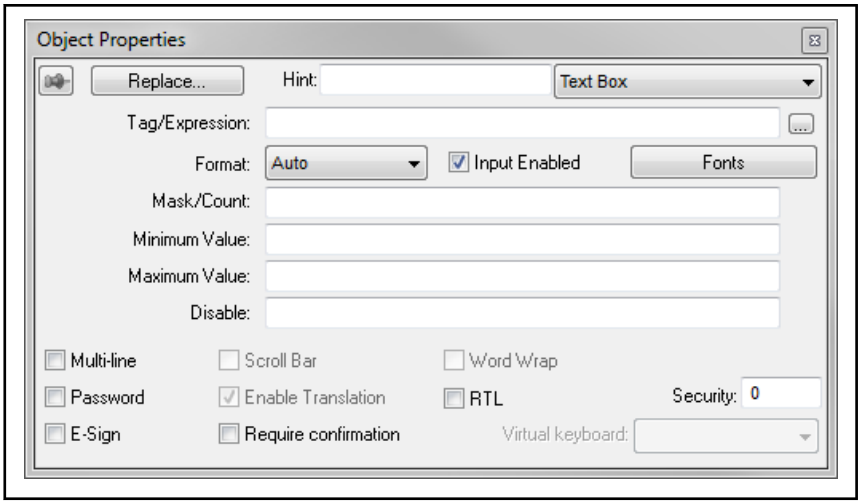


Fig. 8-32: Text box object properties

5. In the **Hint** field, enter a hint or tooltip which is to be displayed during runtime when the cursor runs across the object.
6. In the **Tag/expression** field, enter the name of a project tag or expression to associate it with the object.



Click on the Browse button (...) on the right to use the object search to apply the Tag/expression field.

A project tag can be used for input or output.

An expression can only be used for output. If several lines of text are to be input or output - this means if **Multi-line** is selected as described below - enter the name of a string array with the index 0 (e.g. `MyStringArray[0]`). Trying to start at another position may cause unexpected behavior during runtime. If, for example, `MyStringArray[4]` is entered, the first line of the object shows index 4 of the array but the second line starts again with index 0.

Do not use a different array type (i.e., Boolean, Integer or Real) for the **Multi-line** option.

7. Select from the **Format** list how the numeric value (if any) of the specified tag/expression will be formatted and displayed on-screen. Available options include **Decimal, Hexadecimal, Binary** and **Auto**. If **Auto** is selected, the value is formatted according to the virtual table created by the SetDecimalPoints on page 767 function. This option does not actually change the specified tag/expression in any way. For example: **Tag/expression** is set to an Integer tag, **Input enabled** is checked and the **Format** is set to **Hexadecimal**. You may input a new value in hexadecimal format, but it is saved in your project database as an integer.
8. **Input enabled** is checked by default. If inputs are not to be enabled, deactivate this option.
9. In the **Mask/Count** field, enter a value which limits the input:

   9.1 To mask a numerical value so that it corresponds to a specific format, enter one or several # characters. Each # represents one character of input/output. # characters can be combined with regular text: $###.##

   It's important to remember that the project runtime will always display the most significant digits of a numeric value, regardless of the number or placement of # characters in the text. That

means that if an insufficient number of # characters is available to display the value, it is transformed depending on the format of the value (as set by the **Format** option described below):

- In **Decimal** format, the number of decimal places is determined by the position of the decimal separator in the ### text. However, if an insufficient number of # characters is provided to the left of the decimal separator to display the whole value, the integer overwrites the decimal value. If you try, for example, to have a value of 112.64 displayed in #.##, 112 will be shown.
- In **Hexadecimal** and **Binary** format, if more # characters are provided than required to display the value, the runtime project fills these in with leading zeroes. If less characters are available than required, the value is simply cut off.
- In **Auto** format, the runtime project ignores the number of # characters and displays the entire numeric or string value. Numeric values are displayed in decimal format with their complete whole and fractional values, regardless of the placement of the decimal separator in the ### text. Given an exceptionally large value or long string, this may disrupt the layout of your screens.

9.2 To limit a string to a specific number of characters enter a value between 0 and 1024. 1024 is the upper limit since this value corresponds to the maximum size of the string data type. If more than 1024 characters are entered, the string is cut off and the remaining characters are discarded.

Please note that the limitation of the number of characters applies per line. This means that if the object is configured to contain multiple lines (i.e. if **Multi-line** is selected) and the object is associated with a string array, each text line / each array element can be up to this limit value.

10. In the **Minimum value** and **Maximum value** fields, enter the minimum and maximum numeric value (if available) which the user can enter. This is optional.
11. In the **Disable** box, enter the name of a project tag or expression. This is optional. If the value of the tag/expression is TRUE (1), the object is disabled.
12. If the object is to accept/show multiple lines of text, select **Multi-line**. If **Multi-line** is selected, the **Scroll bar** and **Word wrap** options are available and the **Password** option is no longer available.
13. If the object is to obscure the text input (e.g ********), select **Password**.



If a word wrap is to be inserted in a multi-line text field during runtime, press <Shift>+<Enter>, <Ctrl>+<Enter> or <Alt>+<Enter>.

Button object

To insert buttons, click on **Button** in the **WinStudio Screen Tools** tool window and proceed as follows:

1. Click in the drawing area and drag the mouse/cursor to create the button
2. Release the mouse button when the button has the desired size
3. Double-click on the object to open the "Object Properties" dialog.

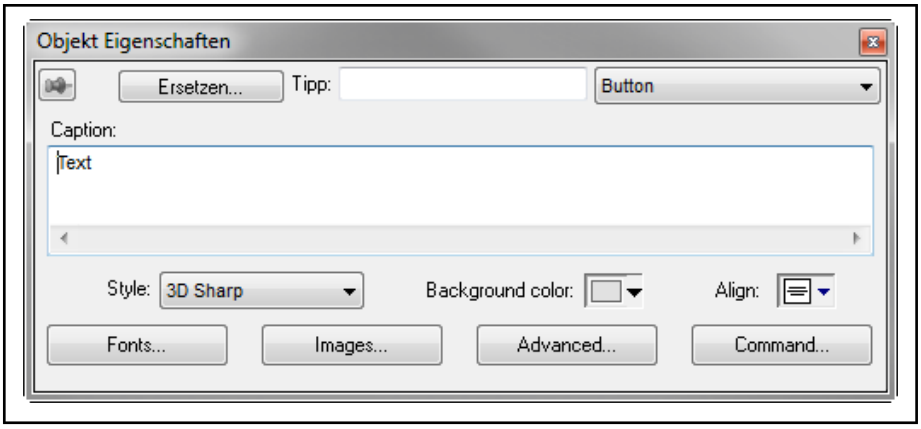


Fig. 8-33: Object Properties: Button

The following parameters of "Button" objects can be configured in the "Object Properties" dialog:

- **Caption**: Text input field for the button caption. The display text may include tags. These have to be entered between curly brackets (e.g. `{tagname}`)
- **Style**: Select one of the following display types for buttons:
  - **3D Sharp**: A raised button with visible contour at the border, suitable for touchscreens
  - **3D Soft**: A raised, rounded button suitable for touchscreens
  - **OS Like**: A button styled to match the operating system on which the project client is running, suitable for Windows desktops with Web Thin Client or Secure Viewer
  - **Standard**: Flat standard button

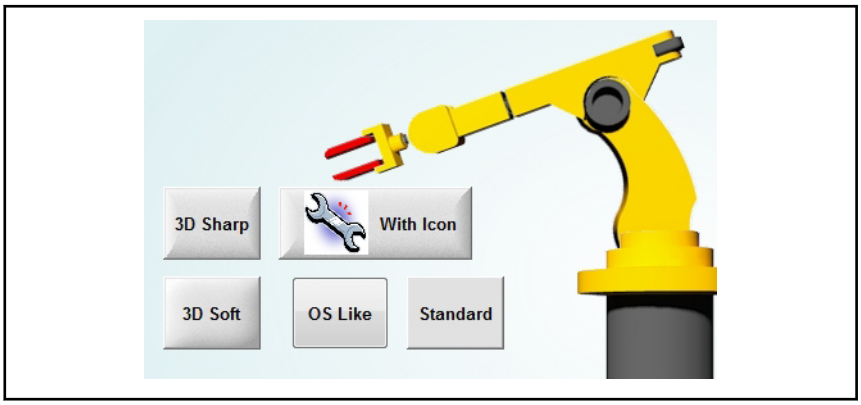


Fig. 8-34: Examples of buttons

- **Background color**: Select a background color for the button
- **Align**: Select the alignment for the caption of the button
- **Fonts**: Set the following caption parameters by clicking the **Fonts** button:
  - Fonts
  - Font style
  - Size
  - Effects
  - Color
- **Images**: Click on **Images** to insert an image file

  Specify the following parameters in the "Images" dialog:

- **File:** Enter the file path to the graphics file. To open a standard Windows file browser, click on the "Search" button next to the box



Icon files (`*.ICO`) cannot be retained in the memory. This means that they have to be loaded from the hard drive every time the project screen is opened or re-drawn. This can have a negative impact on runtime performance. If many Icon files are used in a project consider replacing them by new files in a different format such as GIF, JPG or PNG.

For further information refer to Configuring power control settings on page 54.

- By default, the image is displayed at its actual size. To change this, select User-defined from the Size list and configure the desired image width and height (in pixels)
- **Position:** Select the position of the graphic in relation to the labeling
- **Offset:** Specify the offset (in pixels)
- **Transparent color:** Select which color in the image is to be transparent. The background color of the button (see above) is visible in these areas

- **Advanced**: Specify advanced settings for the button by clicking **Advanced**

  In this dialog, the following parameters can be set:
  - **Enable translation (optional):** Enable this option to apply the text to the translation file
  - **Multiline:** If this option is enabled, the caption of the button can be displayed in more than one line
  - **Wrap text**: If this option is enabled, the object automatically wraps the text when necessary
  - **Auto gray out:** The caption is grayed out at runtime if the Command animation on page 246 applied to the button has been disabled using the **Disable** field or the Security system on page 461
  - **Auto format:** If this option is enabled and if the caption includes a decimal value in curly brackets (e.g. `{1.2345}`) or a tag of Real on page 111 type, the value is formatted according to the virtual table created by the following function: SetDecimalPoints on page 767
- **Command**: Click to automatically apply a command animation on page 246 to the button and switch to the animation properties

Pushbutton object

Select **Pushbutton** in the **WinStudio Screen Tools** tool window to create a pushbutton object by means of the command animation on page 246.

"WinStudio" offers the following preconfigured types of pushbuttons. The button types show the behavior of the hardware buttons with the respective name:

- **Momentary** (default): Changes the state (**Open** or **Closed**) if pressed and returns to its initial state if released. This pushbutton type is always displayed in its normal position when the WinStudio screen is opened

- **Maintained**: Changes the state (**Open** or **Closed**) if pressed but does not return to its initial state if released. Press the button again to change its present state. This pushbutton type retains its state if the WinStudio screen is changed
- **Latched**: Changes the state (**Open** or **Closed**) if pressed and remains in this state until it is released by changing the "Reset" tag

"WinStudio" provides the following button styles:

- Rectangular with a faceplate and indicator light
- Rectangular without a faceplate or indicator light (default)
- Rectangular with a 3-D
- Rectangular with a floating appearance

To add one or more preconfigured buttons to a screen, proceed as follows:

1. Click the **Pushbutton** tool, and position the mouse (pointer) in the WinStudio screen editor
2. Click and drag to create/adjust the size of the rectangular button.

   The button size and text font characteristics determine how much text you can display and how much area you can touch on a touch screen. You can resize the button and change the font characteristics later to permit longer messages to be shown in a given space.
3. Double-click on the object to open the "Object Properties" dialog.



Alternatively, open the "Object Properties" dialog by right-clicking on the pushbutton object or by selecting the object and pressing <Alt>+<Enter> and selecting **Properties** from the context menu

Object Properties: Pushbutton

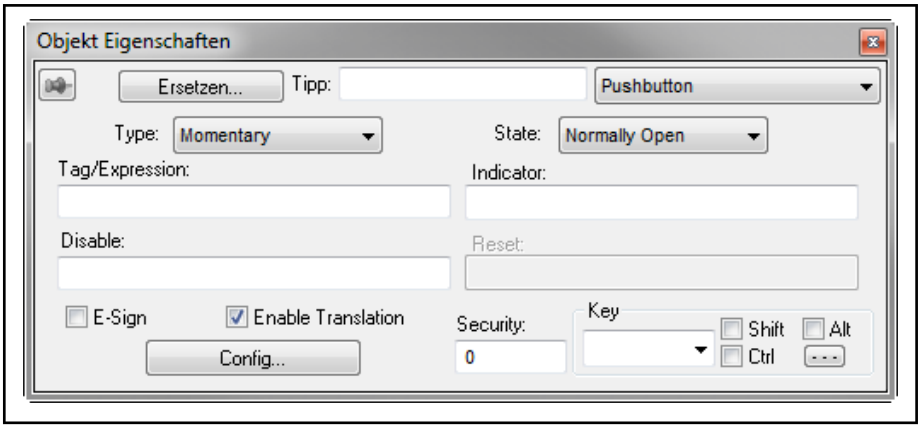


Fig. 8-35: Object Properties: Pushbutton

Use this dialog to specify the following parameters:

- **Type**: Click to select the pushbutton type (**Momentary** (default), **Switch** or **Locked**)
- **Status**: Select the button function logic by clicking on the "State" selection field. The button can be projected as N/O (normally Open) or N/C (normally Closed) contact

  Example: The "Momentary" button is projected as "N/O contact" (normally Open). As long as the button is not actuated, the runtime system reports "open" for the button. The same button projected as "N/C contact" (normally closed) reports "closed" at runtime without actuation

- **Tag/Expression**: Input field to select a WinStudio tag or to enter a program expression:
  - Enter a tag to which a certain value is assigned at runtime. This value depends on the state of the button (e.g. pressed), the preconfigured logics (e.g. N/O) and the "write value" parameterized in the "Configuration" menu
  - Enter a program expression to be executed on pressing the button



"WinStudio""does not" write the result of any expression in the **Tag/Expression** field into a tag

- **Indicator**: Enter a tag to define an indicator that causes the pushbutton to change to a specified color if the tag value matches one of two specified values. Both the colors and tag values have to be defined in the "Configuration" dialog. If this field is left blank, the indicator changes color automatically on pressing the button
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the animation
- **Reset** (only active for pushbuttons of the **locked** type): Enter a tag to control the locking state of the pushbutton as follows:
  - Enter a zero and the pushbutton remains in a locked state after pressing it
  - Enter a non-zero value and a latched pushbutton becomes unlatched after pressing it. The tag value has to be reset to zero before the pushbutton can be pressed again
- **Key** pane: Specify a keyboard code or create a key combination to change the state of a pushbutton if no pointing device (mouse or touchscreen) is available or if shortcuts are to be created in addition to pushbuttons
- **Key** selection list: Enter a key in the text field or select a alphanumeric key from the selection list. Enter only a single character or key. No numbers can be entered in this field.

  Click (enable) the **Shift**, **Ctrl**, or **Alt** field to create a combination key, i.e. the Shift, Ctrl or Alt key has to be pressed in combination with the key specified in the selection list.

  Click the browse button **...** to open the "Key Modifier" dialog to modify the combination keys. Choose **Left**, **Right**or **Left or Right**to specify the position on the keyboard on page 66 of the Shift, Ctrl, or Alt combination key. If **Left or Right** is selected, the command is executed any time one of these keys is pressed in combination with the key specified in the selection list



If you have defined custom keys for your project, you can select them in this list. For more information, see Define custom keys for selected screen objects.

- **Disable**: Enter a non-zero value to disable the pushbutton, thus, pressing the button has no effect. This field is empty by default
- **Enable translation**: If this option is enabled, the text is applied to the translation file. (Refer to Translation tool on page 487 for further information.)

- **Security**: Enter a value to specify a security level (0 to 255) for this button. If the user does not have the specified security level, the pushbutton becomes inactive. If the user has the appropriate security level or this field is left blank, the button is active
- **Config…**: Click to open the "Configuration" dialog to specify "Style" and state parameters for the pushbutton

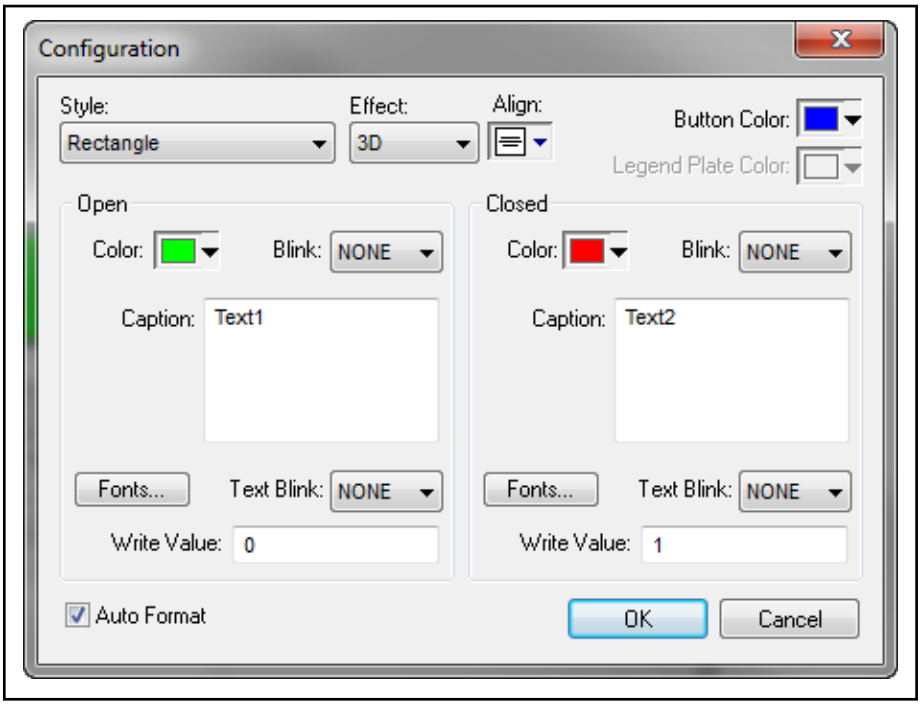


Fig. 8-36: Configuration dialog

This dialog provides the following parameters:

- **Style**: Click the combo box button to select a pushbutton style (**Rectangle** (default) or **Rectangle with faceplate**)
- **Effect** combo box: Click to select a 3-D effect for the pushbutton
  - **Floating** (default): Pushbutton resembling a flat object with a shadow
  - **3D**: Pushbuttons having rounded corners and appearing to be "pressed into the screen" when pressed

  The **Style** and **Effect** parameters can be combined to create four different pushbuttons, as illustrated in the following figure:

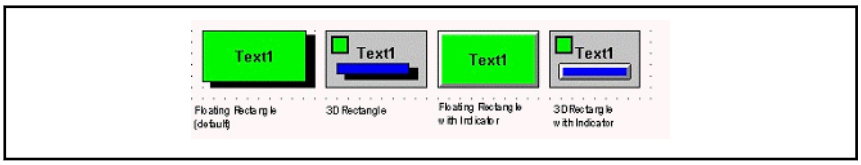


Fig. 8-37: Pushbutton styles

- **Align**: Specify the alignment for the caption of the pushbutton
- **Button Color**: Click to define the constant color of the button if a button style "with frame" is selected

  For button objects "without frame", the set button color is not displayed or in intervals if the "Blink" option is selected

- **Legend Plate Color**: Click to define the frame color for button styles "with frame" (button background = legend)

  This field is disabled if a "button style without frame" has been selected
- **Open** and **Closed** panes: By means of the following parameters, the appearance of pushbutton objects in open and closed state is configured
  - **Color**: Click to specify a default color for an indicator in each **state**. The color switch is only effective for button styles "without frame"
  - **Blink**: Click to specify whether the color defined in the **Color** field blinks and how fast it blinks for each state (**None** (no blinking, "default"), **Slow** and **Fast**)

    If the color is set to blink, the button switches between the color defined in the **Color** field and the basic color configuration of the button (Color button)
  - **Caption** text field: Use this text field to enter the caption of the button

    For button styles "with frame", the text is not displayed on the button but on the frame. Tags can be included in curly brackets (e.g., `{tagname}`).
  - **Fonts** button: Click to open the "Font" dialog to specify or change the message font characteristics for each state
  - **Text Blink** selection list: Click to define whether the specified text blinks and how fast it blinks for each state (**None** (no blinking, "default"), **Slow**, and **Fast**)

    Unlike blinking colors, the blinking text appears and disappears
  - **Write Value** selection list: Entering a value in one of the fields

    If the pushbutton is in the respective state (**Open** or **Closed**), "WinStudio" writes this value to the tag specified in the **Tag/Exp** field ("Object Properties" dialog)
- **Auto Format**: If this option is enabled and if the caption includes a decimal value in curly brackets (e.g. **{1.2345}**) or a tag of Real on page 111 type (see **Caption** above), the value is formatted according to the virtual table created by the function SetDecimalPoints on page 767

Checkbox object

The checkbox object is an input element to activate or deactivate functions on the display.

To insert a checkbox into a WinStudio screen, click **Checkbox** in the **WinStudio Screen Tools** tool window:

1. Click in the drawing area and drag the mouse/cursor to draw the checkbox and its caption
2. Release the mouse button when the object has reached the desired size
3. Double-click on the object to open the "Object Properties" dialog.

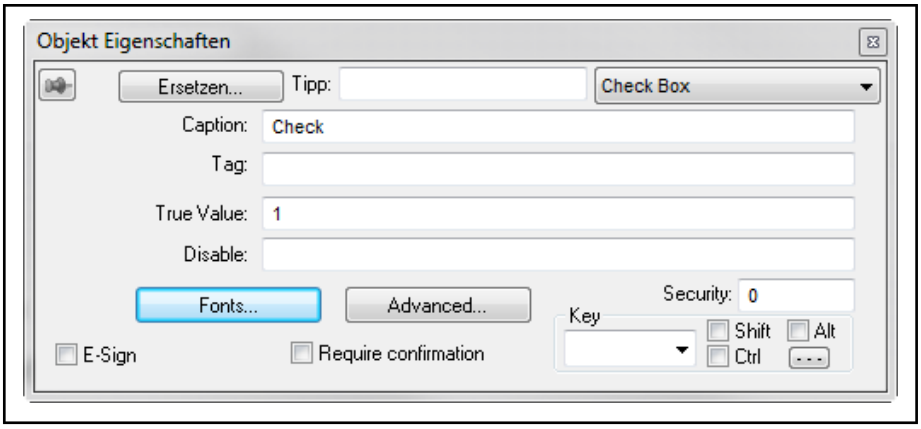


Fig. 8-38: Object Properties: Checkbox

To change the default size of the checkbox, edit the project file (`project_name.app`) to add the following setting:

```
[Objects]
CheckBoxSize=height_in_pixels
```

This setting changes the size of all checkboxes in the project and is to be used with care. This is only available for projects running in Project Viewer (Viewer.exe).

It is not available for projects running in CEView or the Web Thin Client.

Use the "Object Properties" dialog to specify the following parameters for the checkbox object:

- **Caption**: Specify a caption by entering the text into the text field. Tags can be included in curly brackets (e.g. `{tagname}`)
- **Fonts**: Specify a font style for the text by clicking the **Fonts** button
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the command
- **Acknowledgement:** Click (activate) this box to ensure "WinStudio" displays a prompt to confirm the action at runtime
- **Key** selection list: Select a key from the list to connect it with the object or group of objects. This key can then be pressed at runtime to activate/ deactivate the checkbox
  - Click (enable) the **Shift**, **Ctrl**, or **Alt** field to create a combination key, i.e. the <Shift>, <Ctrl> or <Alt> key has to be pressed in combination with the key specified in the selection list
  - Click the browse button **...** to open the "Key modifier" dialog to modify the combination keys. Choose **Left**, **Right** or **Left or Right**to specify the position on the keyboard on page 66 of the Shift, Ctrl, or Alt combination key. If **Left or Right** is selected, the command is executed any time one of these keys is pressed in combination with the key specified in the selection list

If you have defined custom keys for your project, you can select them in this list. For more information, see Define custom keys for selected screen objects.

- **Disable**: Enter a tag or expression into this field to enable and disable the object. The checkbox object is disabled if a non-zero value is entered
- **Security**: Type a value in this field to specify a security level for the object, as defined under Security. If a user logs on and does not have the specified security level, "WinStudio" disables the object
- **Tag**: By clicking on this checkbox at runtime, the value of this tag is updated. If no **Feedback** has been specified, the value of this tag is also used to indicate the current status of the object
- **True Value**: Specify a value that is to be used to change the control state to TRUE and to indicate that the control is in TRUE state. For further information on states, please refer to the states table
- **Advanced**: Click this button to open the "Advanced" dialog:

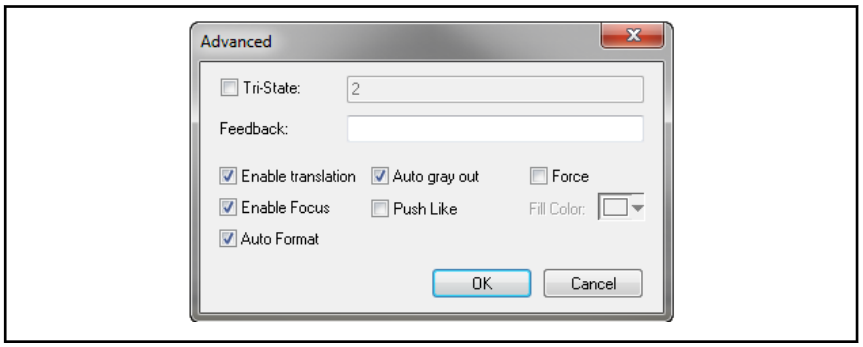


Fig. 8-39: "Advanced" dialog

– **Tri-State**: If enabled, the control element is assigned a third state (grayed out). The third state is displayed if the tag configured in the **Feedback** field applies the value specified in the **Tri-State** field

If the **Feedback** field is left empty, the third state is displayed if the tag configured in the **Tag** field applies the value specified in the **Tri-State** field

NOTICE	**The Tri-State field cannot be configured with the same value as the True Value field. It also cannot be assigned with an empty string**

– **Feedback**: Value that indicates the state of the object (FALSE, TRUE, or TRI-STATE).

If the value of the tag configured in **Feedback** is equal to the value of the tag configured in **True Value**, the state is set to TRUE

If the value of the tag configured in **Feedback** is equal to the value of the tag configured in **Tri-State**, the state is set to TRI-STATE

If none of these conditions are satisfied, the state is set to FALSE.

If the **Feedback** field is left empty, the tag configured in the **Tag** field is used as the **Feedback** tag.

– **Enable translation**: If this option is enabled, the text is applied to the translation file.

– **Auto gray out**: Turns the caption of the object to gray if it is disabled by the **Disable** field or by the security system

- **Force**: Click (activate) this checkbox to force the tag database to recognize a tag change when clicking on the object, even if the value of the tag concerned does not change
- **Enable Focus**: If this option is enabled, the object can be activated during runtime with the navigation keys
- **Push Like**: If this option is checked the control element is displayed as a button, instead of a standard checkbox
- **Fill Color**: Specify the fill color for the button. This option is only activated if the **Push Like** option is enabled
- **Auto Format**: If this option is enabled and if the caption includes a decimal value in curly brackets (e.g. `{1.2345}`) or a tag of Real on page 111 type (see **Caption** above), the value is formatted according to the virtual table created by the function SetDecimalPoints on page 767

Operating modes

The checkbox object can operate in two different modes: Normal and TriState.

For further information, refer to Operating Modes for Checkbox and Radio Button objects on page 207.

Radio button object

The radio button object can be used to create input masks in which one of several options can be selected.

To insert a radio button into a WinStudio screen, select **Radio button** in the **WinStudio Screen Tools** tool window:

1. Click in the drawing area and drag the mouse/cursor to draw the radio button and its caption
2. Release the mouse button when the object has reached the desired size
3. Double-click on the object to open the "Object Properties" dialog.

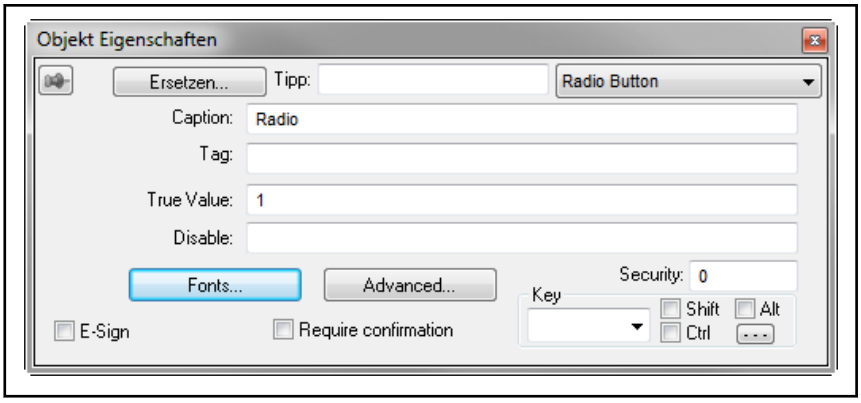


Fig. 8-40: Object Properties: Radio button

To change the default size of the radio button, edit the project file (`project_name.app`) and add the following setting:

```
[Objects]
RadioButtonSize=height_in_pixels
```

This setting changes the size of all radio buttons in the project and is to be used with care.

This is only available for projects running in Project Viewer (Viewer.exe).

It is not available for projects running in CEView or the Web Thin Client.

Use the "Object Properties" dialog to specify the following parameters for the radio button object:

- **Caption**: Specify a caption by entering the text into the text field. Tags can be included in curly brackets (e.g. `{tagname}`)
- **Fonts**: Specify a font style for the caption by clicking the Fonts button
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the command
- **Require confirmation** checkbox: Click (activate) this box to ensure "WinStudio" displays a prompt to confirm the action at runtime
- **Key** selection list: Select a key from the list to associate it with the object or group of objects. This shortcut can then be used to activate/deactivate the radio button
  - Click (enable) the **Shift**, **Ctrl**, and/or **Alt** field to create a combination key, i.e. the <Shift>, <Ctrl> or <Alt> key has to be pressed in combination with the key specified in the selection list
  - Click the browse button **...** to open the "Key modifier" dialog to modify the combination keys. Choose **Left** , **Right** or **Left or Right** to specify the position on the keyboard on page 66 of the Shift, Ctrl, or Alt combination key. If **Left or Right** is selected, the command is executed any time one of these keys is pressed in combination with the key specified in the selection list

If you have defined custom keys for your project, you can select them in this list. For more information, see Define custom keys for selected screen objects.

- **Disable** field: Enter a tag or expression into this field to enable and disable the object. The radio button object is disabled by entering a value different from 0
- **Security** field: Enter a value into this field to specify a security level for the object as defined in the security settings. If a user logs on and does not have the specified security level, "WinStudio" disables the object
- **Tag** field: By clicking on the radio button at runtime, the value of the tag entered here is updated. If no **Feedback** has been specified, the value of this tag is also used to indicate the current status of the object
- **True Value**: Specify a value that is to be used to change the control state to TRUE and to indicate that the control is in TRUE state. For further information on states, please refer to the states table
- **Advanced**: Click this button to open the Advanced dialog:

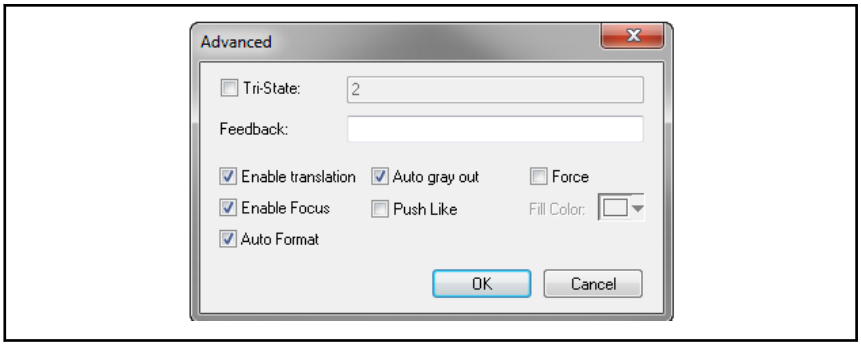


Fig. 8-41: "Advanced" dialog

– **Tri-State**: If enabled, the control element is assigned a third state. The third state is displayed if the tag configured in the **Feedback** field applies the value specified in the **Tri-State** field (selection field grayed out). If the **Feedback** field is left empty, the third state is displayed if the tag configured in the **Tag** field applies the value specified in the **Tri-State** field

NOTICE	**The Tri-State field cannot be configured with the same value as the True Value field. It also cannot be assigned with an empty string**

– **Feedback**: Value that indicates the state of the object (FALSE, TRUE, or TRI-STATE). If the value of the tag configured in **Feedback** is equal to the value of the tag configured in **True Value**, the state is set to TRUE

  If the value of the tag configured in **Feedback** is equal to the value of the tag configured in **Tri-State**, the state is set to TRI-STATE

  If none of these conditions are satisfied, the state is set to FALSE.

  If the **Feedback** field is left empty, the tag configured in the **Tag** field is used as the **Feedback** tag.

– **Enable translation**: If this option is enabled, the text is applied to the translation file.

– **Auto gray out**: Turns the caption of the object to gray if it is disabled by the **Disable** field or by the security system

– **Force**: Click (activate) this checkbox to force the tag database to recognize a tag change when clicking on the object, even if the value of the tag concerned does not change

– **Enable Focus**: If this option is enabled, the object can be activated during runtime with the navigation keys

– **Push Like**: If this option is checked the control element is displayed as a button, instead of a standard radio button

– **Fill Color**: Specify the fill color for the button. This option is only activated if the **Push Like** option is enabled

– **Auto Format**: If this option is enabled and if the caption includes a decimal value in curly brackets (e.g. `{1.2345}`) or a tag of Real on page 111 type (see **Caption** above), the value is formatted according to the virtual table created by the function SetDecimalPoints on page 767

Operating modes

The radio button object can operate in two different modes: Normal and Tri-State.

For further information, refer to Operating Modes for Checkbox and Radio Button objects on page 207.

Operating Modes for Checkbox and Radio Button Objects

The checkbox and radio button objects both have two operating modes: "Normal" and "Tri-State".

These modes essentially work in the same way for each object.

Normal mode

In normal mode, the object has two possible states: FALSE (not selected) and TRUE (selected).

Appearance of objects in normal mode

Object state	Checkbox		Radio button	
	Default	Button style	Default	Button style
FALSE	☐ Check	Check	○ Radio	Radio
TRUE	☑ Check	Check	◉ Radio	Radio

If the object is clicked/selected at runtime, i.e. if the object state is set to TRUE, the tag configured under Object Properties is assigned the value set in the **True Value** field.

When the object is deselected/disabled, i.e. if the object state is reset to FALSE, the value of the tag configured in the **Tag** field is NOT set to the value configured in the **True Value** field. This means in most cases that it is set to "0" or an empty string. However, there are the following exceptions:

- If the **True Value** field is set to "0", "NOT TRUE" corresponds to "1"
- If the **True Value** field is left completely empty, "NOT TRUE" corresponds to the string "UNSELECTED"

Tri-state mode

Tri-state mode is enabled by selecting the **Tri-State** option in the object's "Object Properties" dialog. This adds a third possible state to the object.

Appearance of objects in Tri-State mode

Object state	Checkbox		Radio button	
	Default	Button style	Default	Button style
FALSE	☐ Check	Check	○ Radio	Radio
TRUE	☑ Check	Check	◉ Radio	Radio
TRI-STATE	☑ Check	Check	◉ Radio	Radio

Clicking/selecting the object the first time works the same as it does in Normal mode.

If the object is clicked/selected a second time, the object state is set from TRUE to TRI-STATE and the value of the tag configured in the **Variable** field of the object properties is set to the value configured in the **Tri-State** field.

If the object is clicked/selected a third time, the object change depends on the type of object:

- Checkbox object is reset to FALSE from TRI-STATE

  The tag in the "Variable" field is then automatically assigned the smallest positive Integer value neither configured in state TRUE nor in TRI-STATE

  – **Example 1:**

    Value configured for state TRUE: "TRUE Value" =1

    Value configured for the tri-state: "Tri-State" =2

    Tag value in the "Variable" field for state FALSE = 0 (automatically set at runtime)

  – **Example 2:**

    Value configured for state TRUE: "TRUE Value" =0

    Value configured for the tri-state: "Tri-State" =1

    Tag value in the "Variable" field for state FALSE = 2 (automatically set at runtime)

- The radio button object is reset from TRI-STATE back to FALSE and the value of the tag configured in the **Variable** field is set to the value configured in the **True Value** field. If it is continuously clicked on the object, the object simply toggles between TRUE and TRI-STATE

  To actually clear the object, i.e. to reset the object to FALSE, click/select another radio button in the same group

This functionality is summarized in the following table:

Current state of object	Change to the object on clicking	
	Checkbox	Radio button
FALSE	Object state set to TRUE and value set to **True Value**	
TRUE	Object state set to TRI-STATE and value set to **Tri-State**	
TRI-STATE	The object state is set to FALSE and the tag in the "Variable" field is then automatically assigned the smallest positive Integer value neither configured in state TRUE (True Value) nor in TRI-STATE	Object state set to TRUE and value set to **True Value**

Combo box object

Click on **Combo Box** in the **WinStudio Screen Tools** tool window to insert a selection list with predefined list entries.

If the number of list entries in the runtime environment exceeds the available space (list length), a scroll bar is activated for the list. If an entry is selected from the list at runtime, the combo box is minimized and the selected entry is displayed in the combo box.

Double-click on the combo box object to open the "Object Properties" dialog.

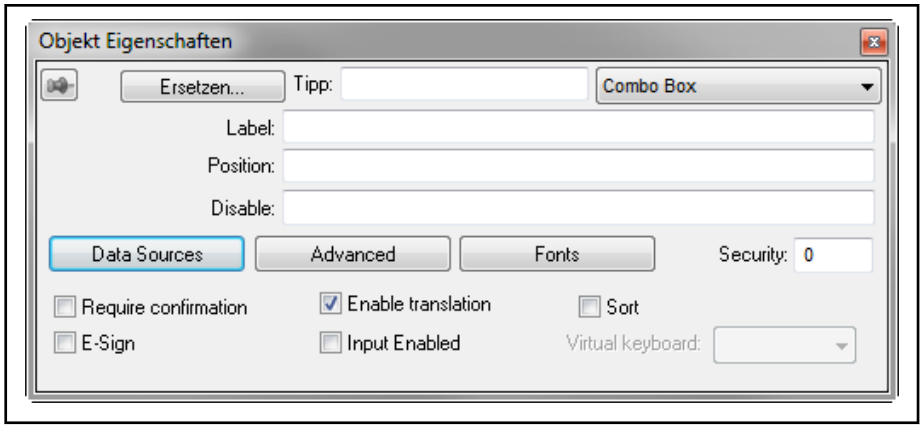


Fig. 8-42: Object Properties: Combo box

In this dialog, the following parameters can be set:

- **Label**: If a string tag is entered, the list entry currently displayed in the combo box is applied to the string tag
- **Position**: Enter a Integer-type tag in which the index of the list position currently selected in the combo box is stored. Changing this tag value changes the displayed list entry
- **Disable**: Enter a tag with a non/zero value to disable this combo box. Enter a zero or leave the field blank (default) to enable the command animation. If the combo box is disabled, it appears grayed out at runtime
- **Data Sources**: Click to open the "Data Sources" dialog (see below)
- **Advanced**: Click to open the "Combo Box - Advanced" dialog:

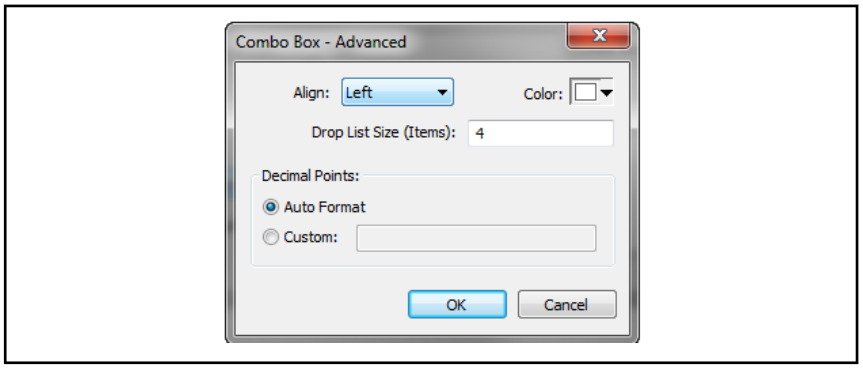


Fig. 8-43: Combo Box - Advanced

  - **Align**: Select the alignment of list entries (**Left**, **Center** or **Right**). This influences the alignment in the combo box as well as in the list
  - **Color**: Click to specify a background color for the combo box
  - **Drop list size (items)**: Enter an integer (or a tag of the Integer on page 111 type) to specify the number of items that are to be displayed at one time if it is clicked on the combo box. The higher the number of items, the longer the drop down list.

☞ If this number is less than the "Total" number of items in the list, the drop down list can be scrolled.

  - "Decimal points": Select how decimal values are displayed on-screen:
    - **Auto Format**: Decimal values are formatted according to the table created by the function SetDecimalPoints on page 767

  - **Custom**: Enter the number of decimal places to display (e.g. 2) for all decimal values

- **Fonts**: To open a fonts dialog click on Fonts on page 280. Use this dialog to change the font of the list entries
- **Security**: Enter a security level for the command (0 to 255). If an operator logs on without the specified security level, the command becomes inactive. If an operator logs on with the specified security level or if this field is left empty, the command element remains active
- **Require confirmation**: Click (activate) to prompt operators to confirm a command at runtime
- **Enable translation**: Click (check) to enable automatic translation of the combo box labels using the translation tool on page 487
- **Sort**: Click (enable) to display the contents of the list in alphabetical order. This parameter is only available if the **Array variable** type has been selected in the "Data sources" pane
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before activating the list selection
- **Input Enabled**: If this option is enabled, available list entries can be selected at runtime and freely selectable values can be entered
- **Virtual keyboard**: For this object, a virtual keyboard is used to enter values. Before configuration of the virtual keyboard for this object, the "Virtual Keyboard" option has to be activated in the **Viewer** tab of the project settings

Data sources

Use the "Data Sources" dialog to configure the elements/labels that are to be displayed in the combo box object.

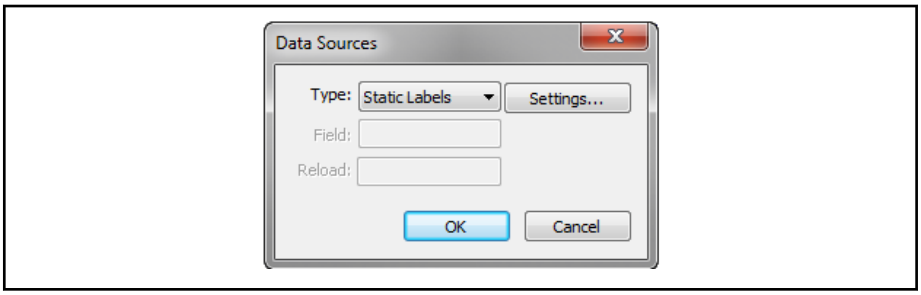


Fig. 8-44: Data Sources dialog

- **Type**: Select the type of data source to be used and click the **Settings…** button to configure the source. Each type of source is described in detail below.
- **Field** (for text file and database only): Specify from which field/column the data source is to read from
- **Reload** (for text file and database only): Enter a tag name. If the value of the specified tag changes, the combo box reloads the labels from the data source

Type: Static labels

If the **type** is set to **Static Labels**, the following "settings" can be configured:

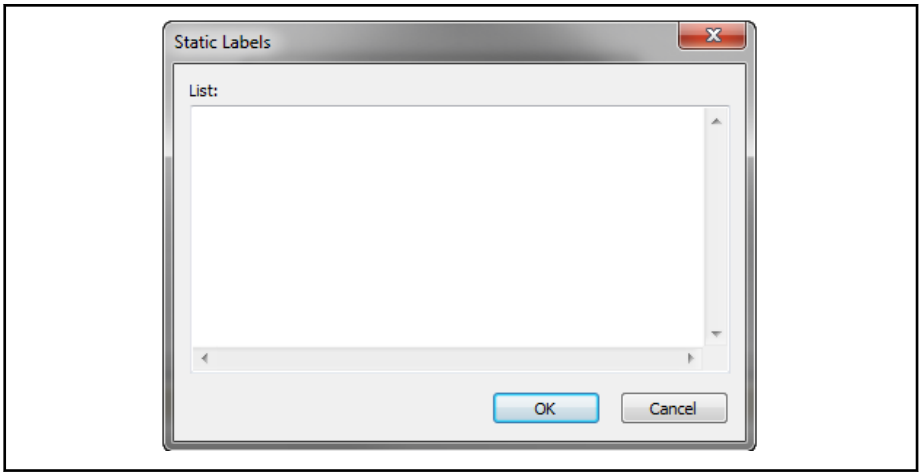


Fig. 8-45: Static Labels dialog

Enter the list entries - with one label per line - just as editing a plain text file. Tags or expressions can be used as list entries by entering them in curly brackets.

Example:

```
Tank#1 in {Array Nametag}.
```

List entries are not sorted and have to be put in the order in which they are to be displayed at runtime. The first line is position 0, the second line is position 1 and so on.

Type: Array variable

If the **type** is set to **Array Variable**, the following settings can be configured:

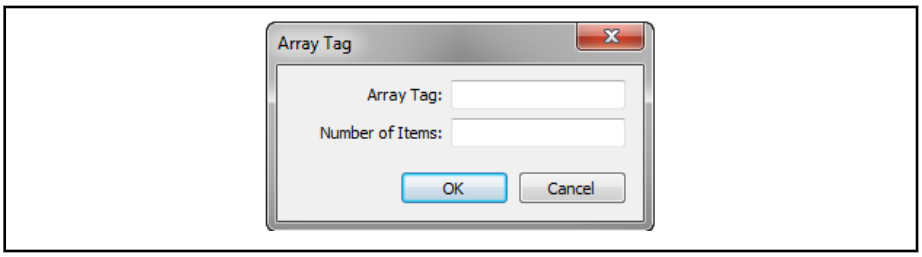


Fig. 8-46: Array tag dialog

- **Array Variable**: Enter the name of an Array variable on page 112 of the String on page 111 type that contains the elements for the combo box
- **Number of Items**: Specify the number of arrays to be displayed in the combo box. "Array Index 0" is counted as the first element. If the value 4 is entered, the combo box displays the array elements 0 to 4

Type: Text file

If the **type** is set to **Text File**, the following "settings" can be configured:

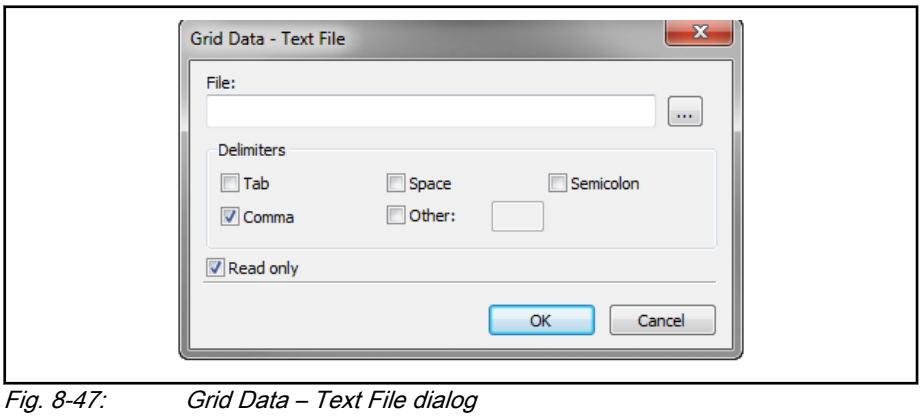


Fig. 8-47: Grid Data – Text File dialog

- **File**: Name of the source text file. Enter the file name and path or click the "…" button to browse for it. (If the file is stored in the project folder, the path in the name is not required.) Tag names can be entered in curly brackets (e.g. **{Tag Name}**) in the **File** field
- **Delimiters**: Set the delimiter(s) used in the data source file. For instance, if the data is to be read from a CSV (comma separated values) file, select the **Comma** option. A user-defined delimiter can be selected by checking the **Other** option and entering the user-defined delimiter in the field to the right

Type: Database

If the **type** is set to **Database**, the following "settings" can be configured:

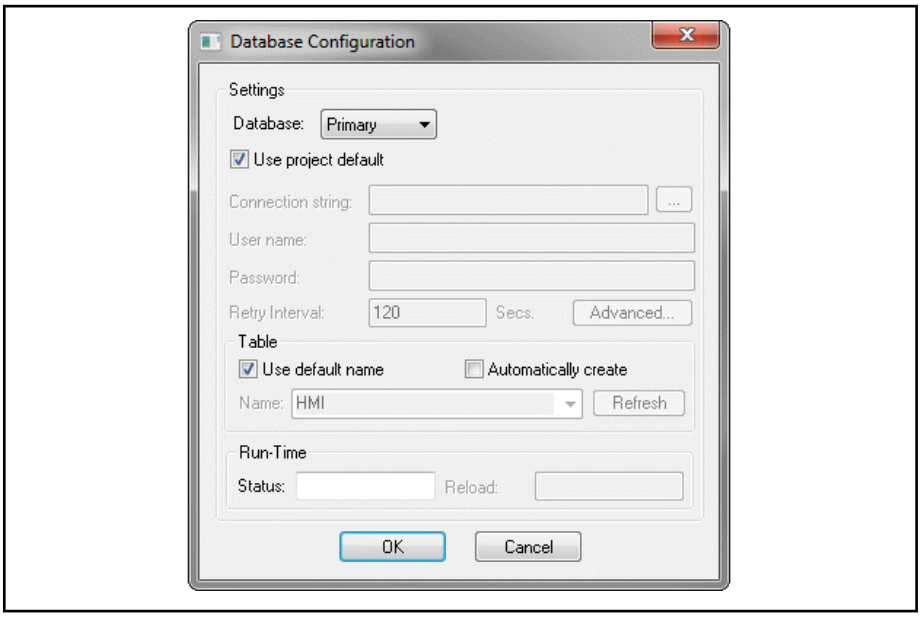


Fig. 8-48: Default Database Configuration dialog

For further information on this dialog, refer to the Database Configuration dialog on page 563.

List box object

To insert a list box, click on **List box** in the **WinStudio Screen Tools** tool window: Generally, if a project is executed, the "active" list box object displays a list of all messages.

On a screen containing only one list box object and no text input boxes, the list box object is activated automatically.

On a screen containing multiple list box objects and text input boxes, a cursor (pointing device) or the tab key can be used to select and activate a list box object.

A message can be selected from the active list box at runtime and the message value can be written to a tag. (If a list is too long to fit in the viewable area of a list box object, the object provides scroll bars.)

Use the option **Require enter for selection** in the "Object Properties" dialog to configure the input behavior of the list:

- Enable the **Require enter for selection** options and browse through the list of messages with the virtual keyboard, list control objects from the **Library**, pointing devices or user-defined keys with the function PostKeys() on page 703. Then, use the Enter key to select the message and apply its value to the write tag. Use the ESC and Tab keys to cancel the input at any time and to return to the previously selected message

- Disable the **Require enter for selection** option to apply the value of a selected (highlighted) message automatically to the write tag

To add list box objects to a WinStudio screen, proceed as follows:

1. To insert a list box, click on **List box** in the **WinStudio Screen Tools** tool window
2. Click in the screen and drag the cursor to create/adjust a rectangle. The height of the rectangle and the font size of the displayed texts define the number of visible messages. The width of the list field defines the visible text per line. After creating the list field, the font size and style can be adjusted to display more messages in the available pane
3. Double-click on the object to open the "Object Properties" dialog.

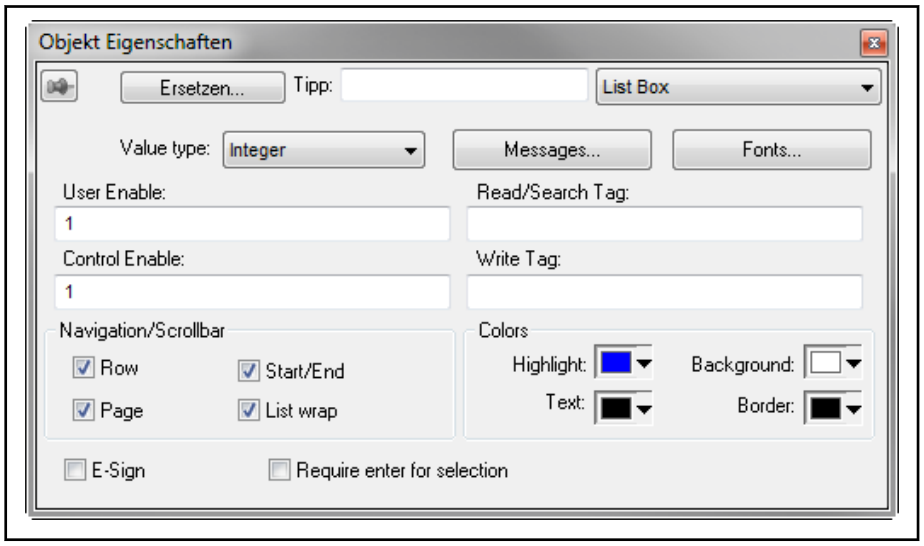


Fig. 8-49: Object Properties: List Box

The "Object Properties" dialog can also be opened, by right-clicking on the list box object or by highlighting the object, pressing the Alt+Enter keys and selecting **Properties** from the resulting context menu.

Use this dialog to specify the following parameters:

- **Value type** selection list: Select one of the following options to indicate the variable types used in the message list
  - **Boolean**
  - **Integer** (default)
  - **LSB** (Least Significant Bit)

For further information, refer to the information in the **Status** column under "Message configuration dialog" on page 215.

- **Messages**: Click to open "Message configuration dialog" on page 215.
- **User enable**: Enter a tag, expression or (non/zero) number to enable selecting messages in the runtime project by the machine operator. The default value is "1" ("true" or "enabled")
- **Control Enable**: Enter a tag, expression or ("non-zero") number to activate the "Text selection" function in the runtime environment

The default value is "1" ("true" or "enabled"). Text is selected by means of a tag in the "Read/Search Tag" field.

WinStudio uses the continuous numbers in the "Value" column under "Message configuration dialog" on page 215 to address the texts. Each number in the "Value" column represents exactly one text in the "Messages" column. If the control function is enabled, the text can be selected directly from the process without user input.

- **Read/Search Tag**: Enter an Integer or a Boolean tag to point to a selected message based on the message **value** field (under "Message configuration dialog" on page 215). Via the "Value" column, the content of the tag refers to a text in the "Messages" column.

  By means of the **Control Enable** and **User Enable** fields, it can be controlled whether the operator or a process can modify this tag.
- **Write Tag** ("optional"): Enter a string tag to receive the message value of the message selected last. If the screen containing a list box object is closed and reopened, "WinStudio" uses this tag value to determine the last message selected in the list box
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature when using this object
- **Row**: Click (enable) to include row up and row down arrows in the list box object scroll bar
- **Page**: Click (enable) to include page up and page down arrows in the list box object scroll bar
- **Start/End**: Click (enable) to include start and end arrows in the list box object scroll bar
- **List wrap**: Click (enable) to enable endless scrolling through the message list
- **Require enter for selection**: Click (enable) this option to enable selecting messages only with the Enter key. It prevents selecting messages with the Tab key
- **Color** fields: Click a color box to open the "Color" dialog or the 16-color "Color Selection" dialog. In both dialogs, colors for the list box object can be specified or changed
  - **Highlight**: Specify a color for highlighting messages (default is blue)
  - **Text**: Specify a color for highlighted text (default is black)
  - **Background**: Specify a color for the list box background (default is white)
  - **Border**: Specify a color for the list box border (default is black)

Message configuration dialog

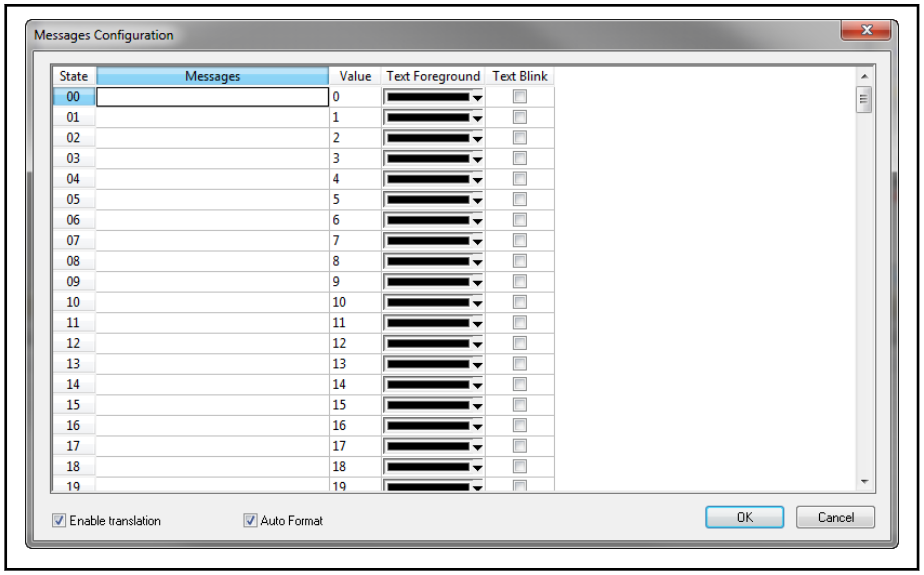


Fig. 8-50: Message configuration dialog

To use the parameters on this tab, proceed as follows:

- **Status** column ("write-protected"): Use this field to view the indexed individual messages

  "WinStudio" automatically numbers these columns depending on the settings selected in the "Value type" selection list:

  – **Boolean**: Provides two valid states, labeled 0 and 1
  – **Integer**: Provides 255 valid states, labeled 1 to 255
  – **LSB**: Provides 32 valid states (32 bits in an integer value) labeled 0 to 31
- **Messages** column: Enter the string to be displayed in the list box object. Tags can be included in messages in curly brackets (e.g. "{"tagname"}")
- **Value** column: Each value entered here corresponds to exactly one text in the "Messages" column. This way, it is possible to address one text via each value. If a suitable value parameterized in the "Read/Search Tag" field is written to the tag during runtime, the corresponding text is automatically selected and highlighted in the "List Object" display. Parallel to this, the value is written to the tag parameterized in the "Read/Search Tag" field. If LSB has been specified in the "Value type" field, "WinStudio" uses the values specified in the "State" column
- **Text Foreground**: Click to specify a color for the message text foreground. The color is displayed only if the message is not selected
- **Text Blink**: Click (enable) to cause a message to blink once per second if it is selected
- **Fonts**: Click to open the "Font" dialog to change the characteristics (style, size, etc) of the message font
- **Enable translation**: Click this option to apply the text to the translation file
- **Auto Format**: If this option is enabled and if the message includes a decimal value in curly brackets (e.g. `{1.2345}`) or a tag of type Real on page 111 (see **Message** above), the value is formatted according to the virtual table created by the function: SetDecimalPoints on page 767

Smart message object

Click on **Smart Message** in the **WinStudio Screen Tools** tool window to create one or more smart message objects to display messages and graphics when executing the project. "WinStudio" provides the following "Smart Message object types":

- **Message display**: Enables displaying any one of multiple messages within a single message object
- **Multistate indicator**: Enables display of one of multiple messages within a single message object while additional bitmap graphics can be displayed in combination with the messages
- **Multistate Pushbutton**: Enables displaying messages and bitmap graphics. This object also resembles a multi-position switch as it enables incremental switching through messages by clicking on the object at runtime

These "Smart Message object types" are distinguished due to their different scope of functions in configuration. However, they have one parameter in common: The variable in the "Read Tag/Expression" field for message selection.

To add a "Smart Message Object" to a WinStudio screen, proceed as follows:

1. Click on **Smart Message** and position the cursor in the screen editor
2. Create a rectangle by clicking and drawing. The size of the rectangle and the font size define how much text can be displayed as well as the size of the bitmap graphic. The rectangle size and font characteristics can be changed later on to allow longer messages to be displayed
3. Double-click on the object to open the "Object Properties" dialog.

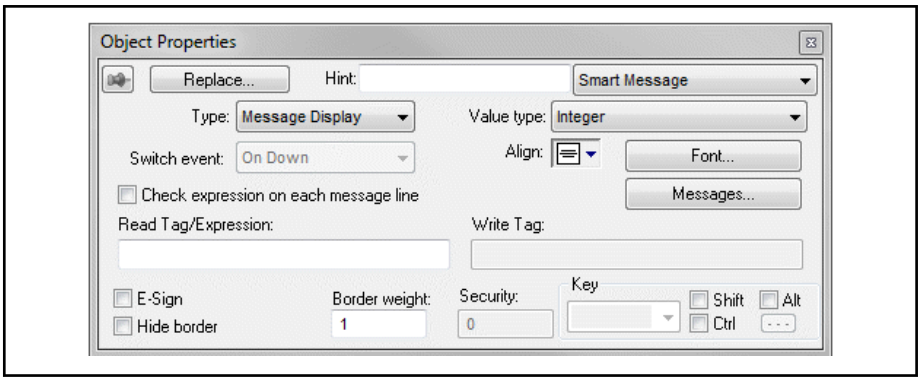


Fig. 8-51: Object Properties: Smart Message

Use this dialog to specify the following parameters:

- **Type**: Click to select the object type. The object type sets the behavior of the object at runtime and the features supported by it:
  - **Message display** (default)
  - **Multistate indicator**
  - **Multistate pushbutton**
- **Value type** selection list: Select the type of values used to index the message list:
  - **Boolean**: Provides two valid states. Use this selection to display one of two different messages, based on a Boolean value (0 or 1)
  - **Integer** (default): Provides 500 valid states. Use this selection to display different messages based on numeric values from an Integer tag

– **LSB** (least significant bit): Provides 32 valid states (32 bits in an Integer value). Use this selection to display different messages based on which bit from an Integer tag is set. If more than one bit from the Integer tag is set simultaneously, the message associated with the lowest value bit is displayed



If **Multistate pushbutton** is selected as **Smart message** object type, only 16 different messages can be associated with the object, even for **Integer** or **LSB** values.

- **Switch event** selection list (available for **Multistate pushbutton** only): Select one of the following options to specify when the message is changed:
  – **On down:** Switch to the next message when clicking on the object (default)
  – **While down:** Switch to the next message continuously while holding the mouse button down on the object
  – **On up:** Switch to the next message if the mouse button on the object is released
- **Align**: Select the alignment of the text displayed by the "Smart message object"
- **Fonts**: Opens the Fonts dialog to configure the font settings for the text displayed in the object
- **Check expression on each message line**: If this option is disabled, only **Read tag/expression** is evaluated during runtime and the resulting value determines which message is displayed. If this option is selected, each message has its own tag/expression
- **Read tag/expression**: Enter a tag or expression. The value determines which message is displayed by the object at runtime
- **Messages:** Displays the Configuration dialog box to configure the messages for the object. Refer to "Configuring messages" on page 218.
- **Write Tag** text box (optional and available for **Multistate Pushbutton** only): Enter the name of a project tag. The value associated with the message currently displayed by the object is written to this tag.
- **E-Sign** (available for **Multistate Pushbutton** only): If this option is selected, a prompt is displayed to enter the electronic signature before executing the animation.
- **Hide border:** If this option is selected, the line border of the object is not visible.
- **Border weight:** Defines the thickness (in pixels) of the line drawn around the object.
- **Security** (available for **Multistate Pushbutton** only): System Access Level required for the object/animation.
- **Key** area (optional and available for **Multistate Pushbutton** only): Shortcut used to go to the next message (step) using a keyboard when the Multistate Pushbutton type is selected. This option is especially useful when creating projects for runtime devices that do not provide a mouse or touch-screen interface — the keyboard is the only physical interface available to interact with your project during runtime.

If you have defined custom keys for your project, you can select them in this list. For more information, see Define custom keys for selected screen objects.

Configuring messages

The behavior of the object and consequently the layout of this dialog box depends on whether **Check expression on each message line** is selected.

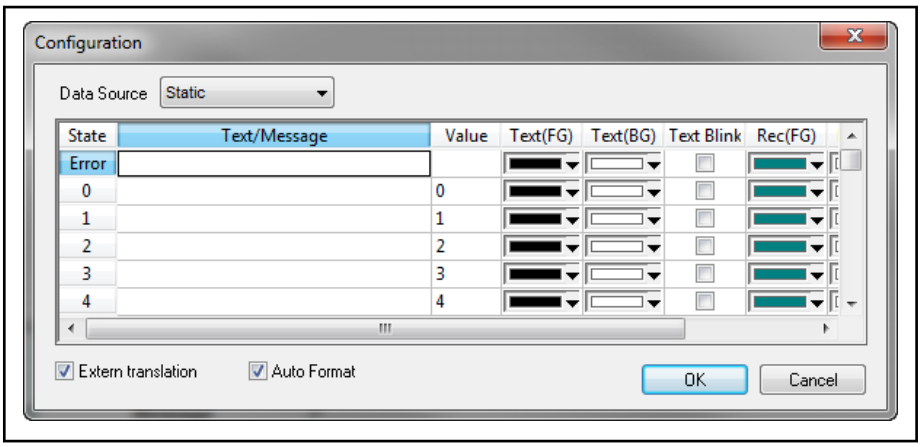


Fig. 8-52: Configuring messages for the resulting values for Read tag/expression

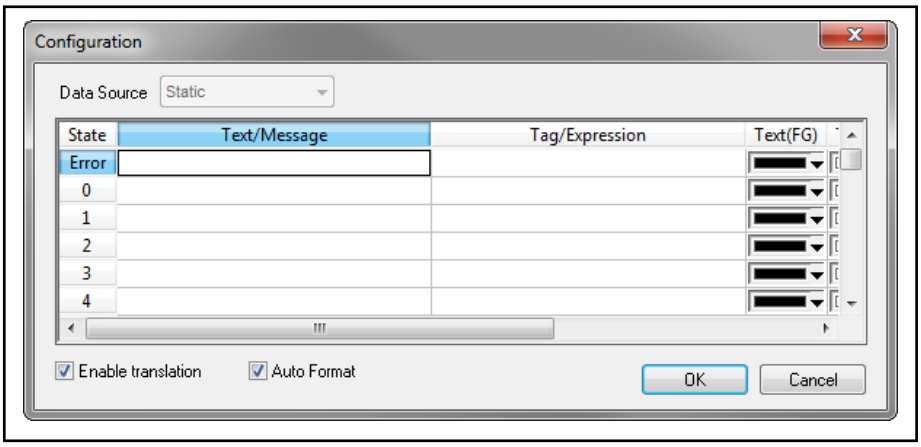


Fig. 8-53: Configuring a different tag/expression for each message line

Data source list field: The source of the messages to be displayed by the object. Select one of the following options:

- **Static:** The messages are configured and saved directly on the object. Configure one worksheet row for each possible status.

  For further information, refer to tab. 8-2 "Properties for data source: Static" on page 219.

- **Text file:** The messages are loaded from an external source file. Enter the path (in your project file) and the name of the file into the **Source file** field.

If the file path is to be changed during runtime, enter the name of a string tag in curly brackets (e.g. `{MyTag}`) and then save the file path in this tag.

Further information on creating and formatting the file can be found in chapter "Source file format" on page 219.

The following table describes the meaning of the properties associated with each message, regardless of the data source:

Property	Description
Text/Message	Message (text) that is displayed when selected during runtime. Tags can be included in messages in curly brackets (e.g. `{tagname}`)
Value (if **Check expression on each message line** is deactivated)	The unique value associated with any **Text/Message**. **Read tag/expression** (in the object properties) is constantly evaluated during runtime and the resulting value determines which message is displayed on the object. If the resulting value is not on this list, the message (**Error**) configured in the first line will be displayed. If **Multistate switch** has been specified as object **type**, the value of the current message is also written to a tag if configured in the **Write tag** field
Tag/Expression (if **Check expression on each message line** is selected)	The Tag/Expression associated with each **Text/Message**. All of them are constantly evaluated during runtime and the first in the list which is evaluated as TRUE (not zero) determines which message is displayed on the object. If none of them is evaluated as TRUE (not zero), the message (**Error**) configured in the first line will be displayed.
Text (FG)	Foreground color for the messages displayed at runtime
Text (BG)	Background color for the messages displayed at runtime
Text Blink	If enabled, the message text blinks at runtime
Rec (FG)	Line color (border) for the rectangle behind the message
Rec (BG)	Background (fill) color for the rectangle behind the message
Rec Blink	If enabled, the rectangle behind the message blinks during runtime
Graphic File	If applicable, path and name of the bitmap file (*.BMP) that is displayed when the assigned message is selected during runtime. If the path is not specified, the bitmap file has to be stored in the project folder
Transparent	Select the color to be transparent in the graphics file if the **En. transparent** checkbox is enabled
Transparent enabled	If enabled, the color selected in the **Transparent** field is set as transparent in the graphics file

Tab. 8-2: Properties for data source: Static

Enable translation: Click (check) to enable translation during runtime using the Translation tool on page 487.

Auto format: If this option is selected and if the message includes a decimal value in curly brackets (e.g. {1.2345}) or a project tag of type Real (see Text/Message below), the value is formatted according to the virtual table created by the function SetDecimalPoints on page 767.



The properties **Graphics file**, **Transparent** and **Transparent Enabled** are not available for the **Message Display** type.



Data can be copied from this dialog and pasted into an Excel worksheet and vice versa.

Source file format

This section describes the format of the text file supported by the "SmartMessage" object if a text file is configured as data source. The main advantage of using an external text file instead of static values is that it provides the flexibility to change the messages during runtime by changing the name of the used text file or even the content of the text file.

The text file has to be created in CSV format (Comma Separated Values), where the comma character (",") is used to divide the columns (data) in each line of the file. Therefore, any CSV editor such as Microsoft Notepad and Microsoft Excel can be used to create the CSV file with the messages and their properties for the "Smart Message object".

For a description of all message properties, refer to Smart Message object on page 216. The order of the data in the CSV file is described in the following table:

Column #	Property	Default value
1	Text/Message	-
2	Value	-
3	Text (FG)	0
4	Text (BG)	16777215
5	Text Blink	0
6	Rec (FG)	8421376
7	Rec (BG)	16777215
8	Rec Blink	0
9	Graphic File	-
10	Transparent	0
11	Transparent enabled	0

When configuring text messages with a comma character as part of the message, the entire message has to be configured between quotation marks (e.g. `"Warning, Turn the motor Off"`). Otherwise, the comma is interpreted as data separator and not as part of the message.

- The first line of this file is equivalent to the **State** = **Error**. In other words: If the configured tag in the "Read tag/expression" field is given a value to which no text has been assigned, the message configured in the first line (**State** = **Error**) is displayed.

  As the error text of the first line is not triggered by a specific value, the entry in the "Value" column is irrelevant. This first line must always be configured, regardless of the object type (even for **Multistate Pushbutton**).
- The **Text/Message** and **Value** columns are mandatory. All other columns are optional. If no values are specified here, the default values are used (refer to the table).
- The fields **Text(FG)**, **Text(BG)**, **Rec(FG)**, **Rec(BG)** and **Transparent** can be configured with the code of the color associated with it. The code can be entered directly in decimal format (e.g. 255) or in hexadecimal format using the syntax `#value` (e.g. `#0000FF`).
- The fields **Text Blink**, **Rec Blink** and **En. Transparent** can be configured with Boolean values "0" or "1" (**0** = **Disabled**; **1** = **Enabled**), or with the keywords **FALSE** or **TRUE** (**FALSE** = **Disabled**; **TRUE** = **Enabled**).

Example:

```
Error Message,,0,16777215,1,8421376,16777215,1,error.bmp,0,0
Message Zero,0,0,16777215,0,8421376,16777215,0,open.bmp,65280,1
Message Ten,10,0,16777215,0,8421376,16777215,0,closed.bmp,65280,1
Message Twenty,20,0,16777215,0,8421376,16777215,0,,0,0
Message Thirty,30,0,16777215,0,8421376,16777215,0,,0,0
```

The editor of the "SmartMessage" object - when selecting

Data source = Static

- can be used to configure messages, values and colors. Optionally, the content of the table can be highlighted and inserted into an Excel worksheet via the Windows clipboard (Copy&Paste).

Then, the Excel worksheet can be saved as a CSV file (**File ▸ Save as**) and integrated again in WinStudio by changing the data source to "Text file".

This procedure provides a user friendly interface for configuring the color codes.

Changing the properties of multiple screen objects (active objects)

This task describes how to select two or more screen objects and how to then change the properties the selected objects have in common.

Before starting this task, a project image must be opened in the image editor.

Which properties can be changed depends on whether several objects of the same type or different object types are selected. If the objects are of the same type, you can change the properties applicable to this type. If, for example, several button objects are selected, the properties applicable to button objects can be changed.

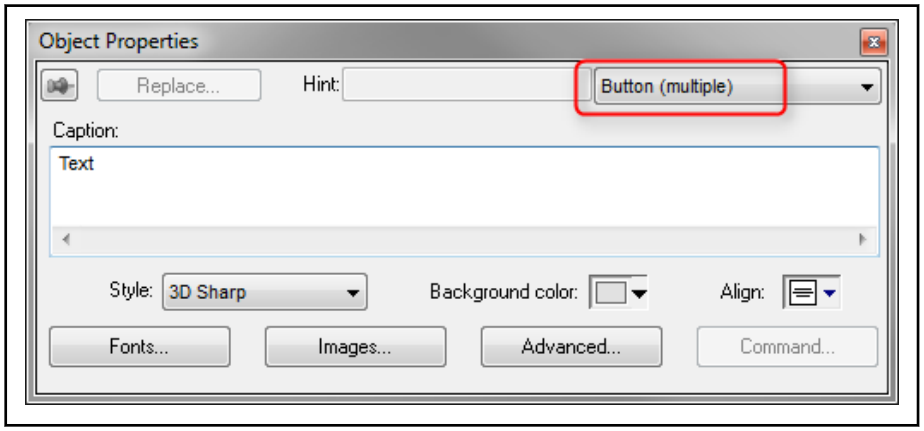


Fig. 8-54: Object properties of multiple selected button objects

For more information on the properties of a specific object type, please consult the documentation for this object

This method can only be used to change the properties of shapes and active objects. The method cannot be used to change the properties of file objects, animations, library elements or objects in a group.

However, if multiple objects of different types are selected, the properties shared by all objects can be changed. This not only includes cosmetic properties such as Border and Background but also functional properties such as Disable, Security, Enable translation and E-Sign. (Some properties may not apply to all objects. Button objects, for example, do not have a border and a rectangle object does not have a security.)

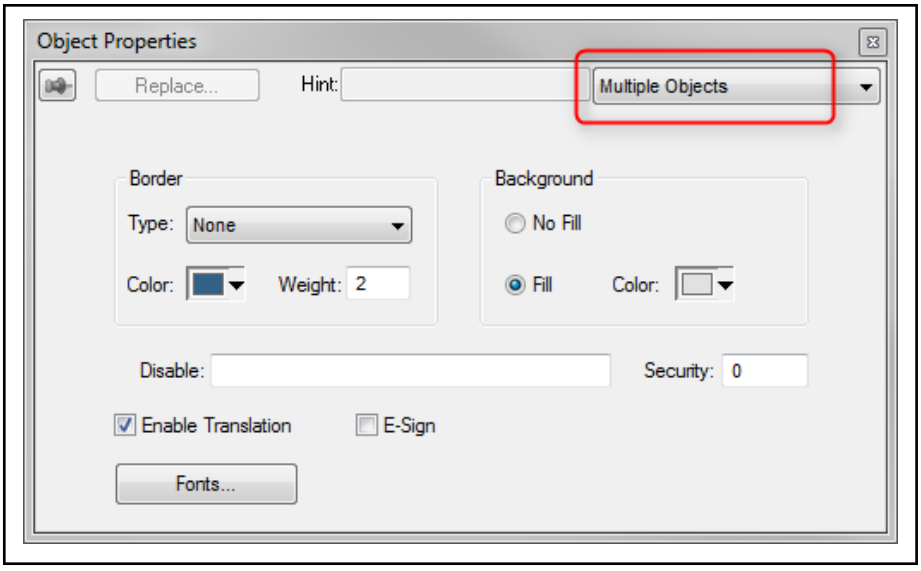


Fig. 8-55: Object properties for multiple objects of different types

In both cases, the dialog box displays the current values of the properties of the most recently selected object.

Only if the value of a property is actually changed, will the change be applied to the selected objects. All other properties remain unchanged independent from the values displayed in the dialog box.

Changing the properties of multiple screen objects:

1. Do one of the following in the screen editor:
   - Either press and hold down <Shift> or <Ctrl> on the keyboard and click on each object that is to be changed; or
   - Use the cursor to draw a selection box around all objects that are to be changed.

   The objects are selected.
2. Do one of the following:

   Right-click on the selected objects and click on **Properties** in the context menu; or

   Press <Alt>+<Enter> on the keyboard.

☞ The Object properties dialog box cannot be opened by a double-click as usual since a double-click cancels the selection.

   The Object properties dialog box is displayed for the selected objects.
3. Change the property values as desired and then close the dialog box.

The changes are applied to all selected objects.

8.2.4 WinStudio libraries

Symbols

Symbols - General information

The **library** is a visual browser for all of the symbols on page 223 that are available to be inserted in a WinStudio screen.

To open the "WinStudio symbols" tool window, select the entry **WinStudio library** in the context menu **WinStudio ▸ Tools** or in the context menu **User symbols**

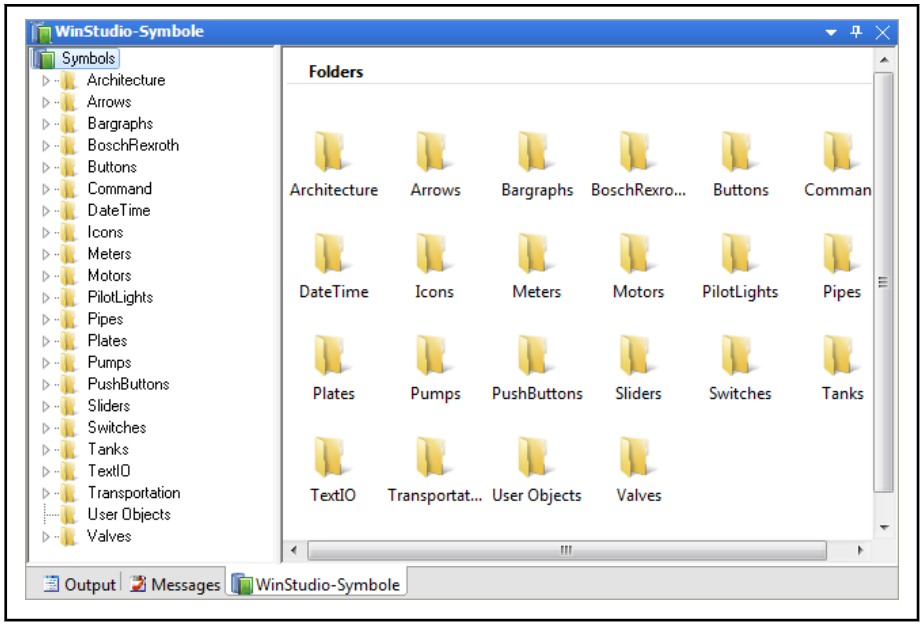


Fig. 8-56: Symbols Library window

The "WinStudio Symbols" tool window offers a comprehensive selection of predefined symbols. These symbols are sorted in folders according to categories (e.g. buttons, counters, tanks).

To select a symbol and insert it into a WinStudio screen, proceed as follows:

1. Find the required symbol in the library and click on it

   The mouse cursor changes to indicate that a symbol has been selected
2. Return to the WinStudio screen in which the symbol is to be inserted
3. Click anywhere in the screen to position the selected symbol
4. If required, configure the object properties on page 178 of the symbol

For further information, refer to Saving own project symbols on page 223:

Providing user-defined symbols to other projects

User-defined symbols are only available in the project where they were initially created and saved, i.e. the "User Symbols" folder in the project tree is project-specific. However, user-defined symbols can be sent to the "User Objects" folder of the WinStudio symbol library to make them available for all projects.

1. Select the symbol in the "User Symbols" folder that is to be made available for other projects

   Select the entry **Apply to WinStudio library** in the context menu of this element
2. By means of this function, the symbol is copied to the `..\Users\All Users\WinStudio\Symbols` directory and now appears in the `User Objects` folder

Saving own project symbols

A **Symbol** is a group of one or multiple objects saved in the "User Symbols" folder that can be reused without limits in the project.

Every time a symbol is used, a copy is made that is linked to the master **Symbol** file in the "Symbols" folder.

These linked copies are also referred to as "instances" of the Symbol.

If any changes are made to the master symbol afterwards, these changes automatically apply to every linked copy in all WinStudio screens.

Each linked copy of the master symbol can be individually extended by **user-defined properties**.

For example, if a gauge to display tank filling levels is created and saved as master symbol, each instance can be supplemented individually:

Each instance of the gauge can display the filling level for a different tank.

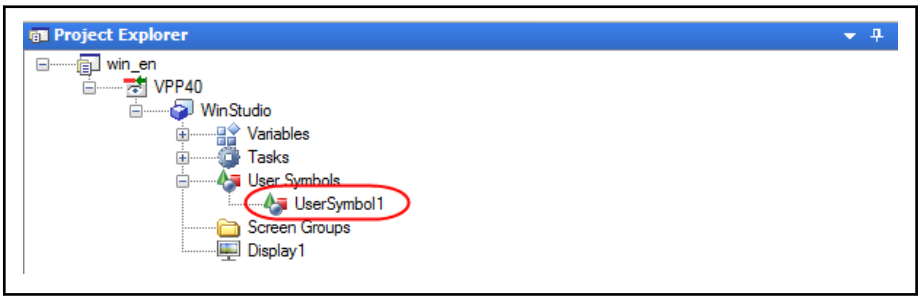


Fig. 8-57: "User Symbols" folder

Creating a master symbol

To create a master symbol and save it to the "User symbols" folder, proceed as follows:

1. Design the symbol just as normally editing a WinStudio screen using any combination of static on page 181 and active objects on page 191.

   For example, three checkboxes on page 201 in a rectangular pane on page 184:

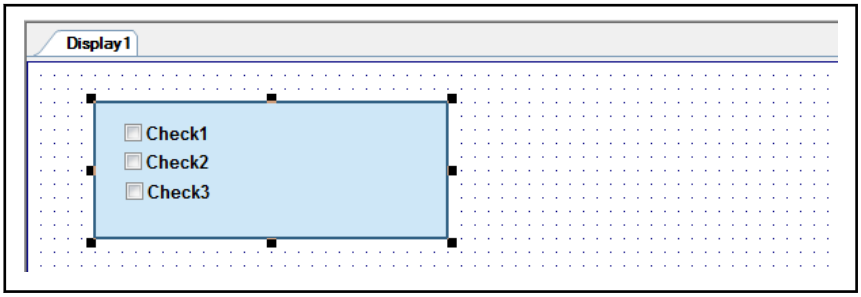


Fig. 8-58: Adding objects into a WinStudio screen

2. Select the object(s) or group on page 270 to be saved as a symbols.

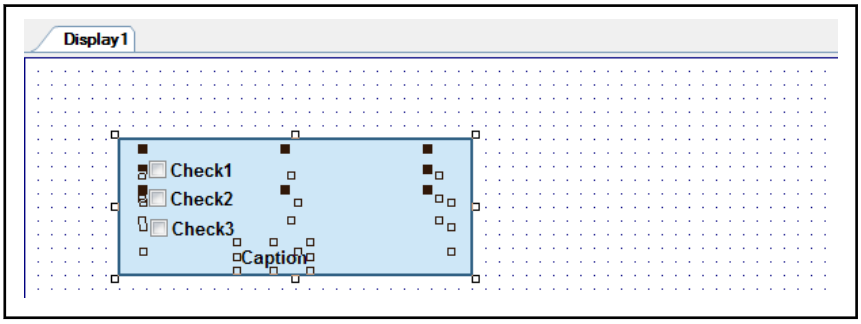


Fig. 8-59: Selecting the objects



It is not necessary to make a group out of two or more objects before saving them as a symbol. Saving the objects together as a symbol effectively groups them.

However, there is one use case requiring grouping the objects:

A symbol can only display a single tooltip.

If a symbol is generated from several objects without previous grouping, existing (if any) tooltips cannot be displayed (under object properties on page 178).

To specify a tooltip for the symbol, the objects have to be grouped first to configure the tooltip for the group. This tooltip is retained if the group is saved as a symbol.

3. Right-click on the selection and click on **Create Linked Symbol** in the context menu.

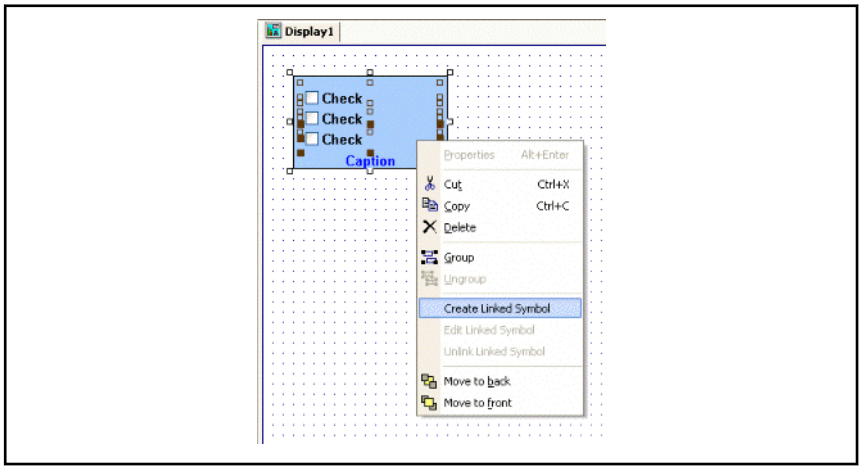


Fig. 8-60: Creating a linked symbol

4. A standard "Save As" dialog is displayed to specify a file name for the new symbol. Symbol files (".sym") are saved in the **\Symbol** folder of the project.

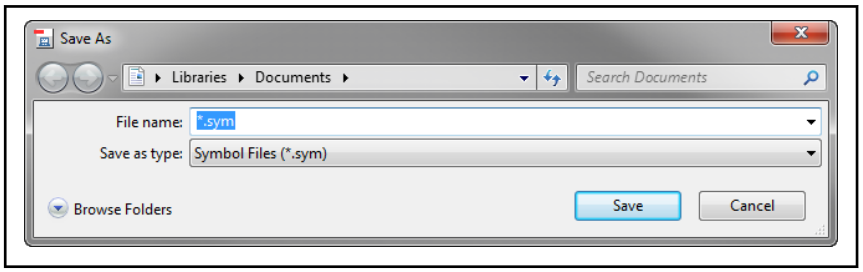


Fig. 8-61: Saving the symbol file test.sym

5. Click **Save** to save the file.

The symbol appears in the "User Symbols" folder, in the Graphics tab on page 170 of the Project Explorer.

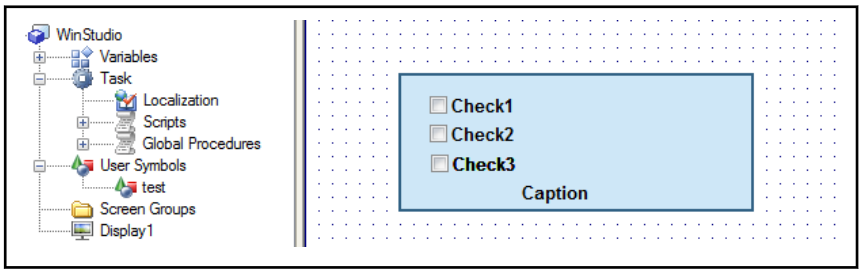


Fig. 8-62: Symbol file in the Project Explorer

The symbol is now ready to be reused in the project. However, the way it is currently saved, every copy has identical properties.

Now, user-defined properties have to be defined for the symbol, i.e. the properties that can be adjusted every time the symbol is used.

Editing the master symbol

Master symbols can be edited after saving it, e.g. by adding or deleting objects in the symbol or by defining user-defined properties.

Remember that any changes to the master symbol are automatically applied to any linked copies (instances) in the entire project. For example, the font color was changed from black to blue...

There is one exception:

A change of the tooltip for a symbol as described in "Creating a master symbol" on page 224, only applies to new instances of this symbol. Existing instances are not changed.

To edit a symbol, proceed as follows:

1. In the "User Symbols" folder, double-click on the symbol file.



Optionally, right-click on any instance of the symbol and choose **Edit Linked Symbol** from the context menu.

The symbol file is opened for editing in a separate window.

This symbol editor works in the same way as the WinStudio screen editor, except that every object in the window is part of a symbol.

If objects are added, moved or deleted in the symbol editor, the size or shape of the symbol may be changed and influence the layout of the WinStudio screens it is used in.

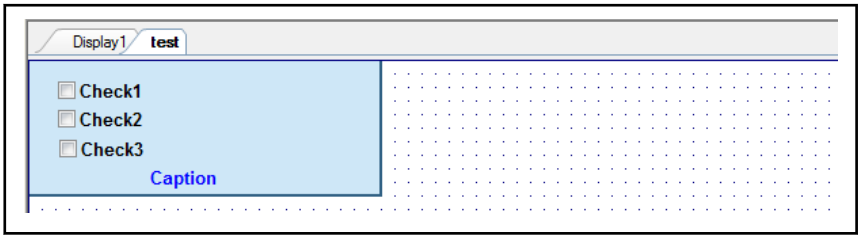


Fig. 8-63: Symbol file opened for editing

Besides adding, moving or deleting objects in the symbol, the object properties on page 178 can be edited just as usual.

It is not always useful that all properties of a symbol are similar in all instances.

In this example, the stored tags on page 104 and the labeling of the three checkboxes are to be freely selectable for each instance.

2. Select the first object of the symbol and open its object properties. For example, the first checkbox:

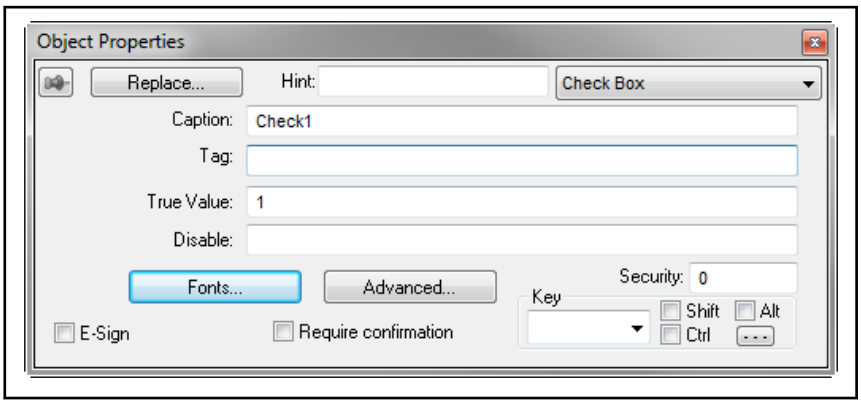


Fig. 8-64: Object Properties for the first check box

3. In any field where usually a tag, expression, or value is to be configured, a user-defined property can be defined using the syntax:

   `# [Category.] Property: [Value]`

   …where:

   - **Category** is an "optional" name for a collection of properties to be grouped, such as all captions or checkbox values. If no **Category** is specified for a property, it is automatically listed under the "Main" category
   - **Property** is the display name to describe a property. A display name is required for each property, and has to be always followed by a colon (":")
   - **Value** is an "optional" default value for the property



All tag/expression syntax rules apply to "value", including tag names on page 109, pointers on page 117, arrays on page 112, strings, numerical and Boolean values and scripting functions.

In the following example, the tag containing the current state of the checkbox is to be individually set for every master symbol instance.

In the **Tag** field, enter **#Option1.Tag:** as illustrated.

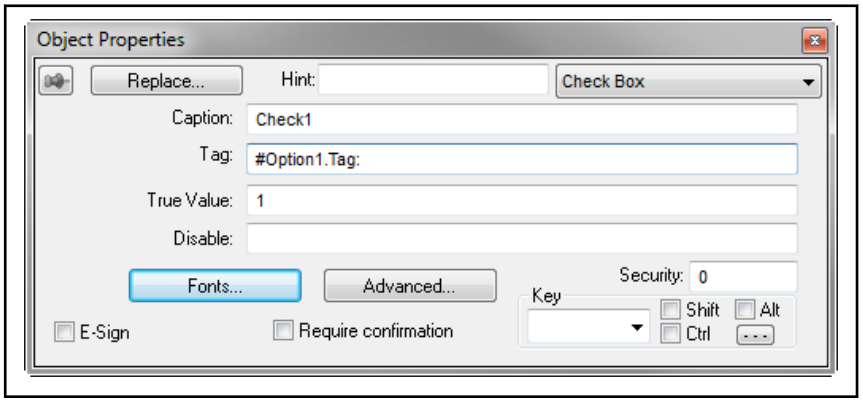


Fig. 8-65: Defining a user-defined property for the tag field

The entry "#Option1.Tag:" in the master symbol corresponds to the following display in the Object Properties dialog of a symbol instance:

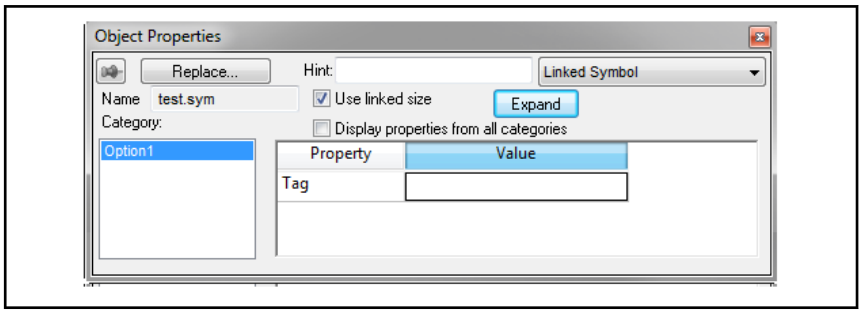


Fig. 8-66: User-defined properties of a symbol

A tag has to be entered in the "Value" column to obtain the status of the checkbox for this instance. For further information, see below.

4. Object property fields usually expecting simple texts (strings) can be replaced by dynamic elements (placeholders) in curly brackets. In this ex-

ample, the **Caption** field requires a string. As illustrated, enter **{#Option1.Caption:"Option 1"}**.

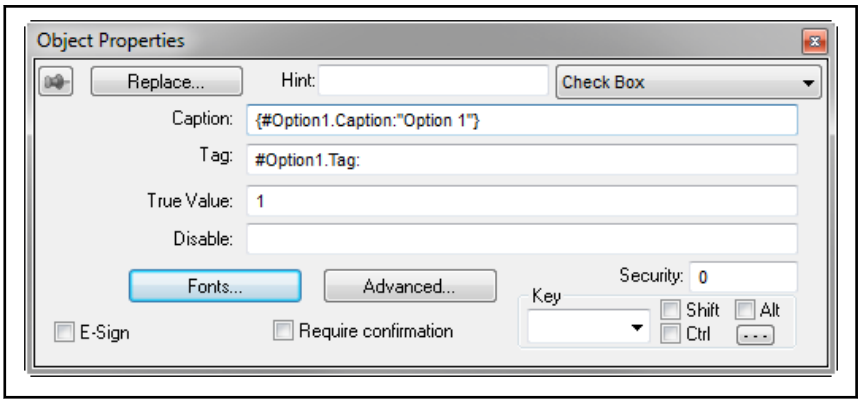


Fig. 8-67: Defining a user-defined property for the Caption field

The entry in the master symbol leads to the following entry in the Object Properties dialog of the symbol instance:

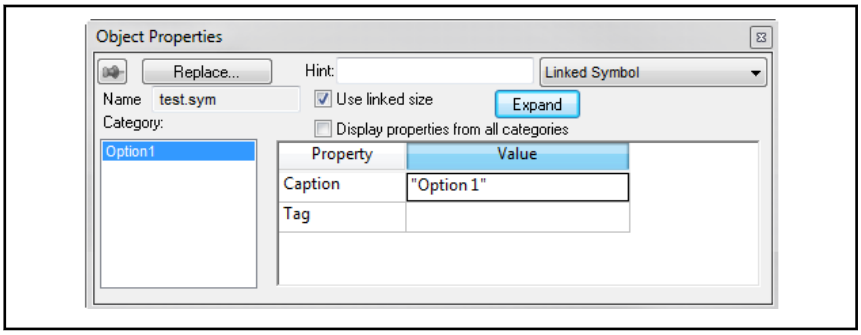


Fig. 8-68: User-defined properties of a symbol

5. Repeat steps 2 to 4 to parameterize the remaining user-defined properties of the master symbol. In this example, the Object Properties dialog of the symbol instance displays the following entries:

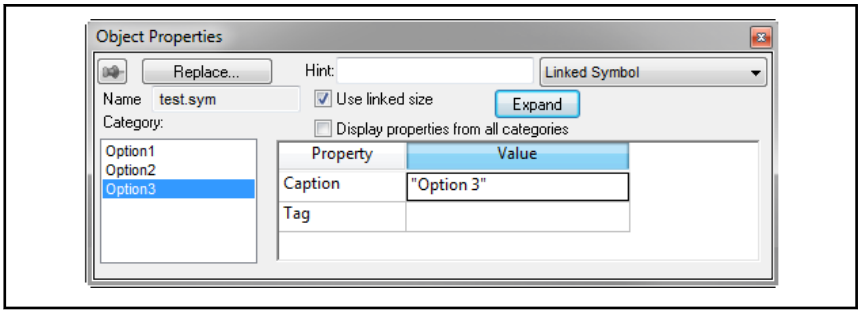


Fig. 8-69: User-defined properties of a symbol

Display properties from all categories enables displaying all object properties of the symbol in a list.

6. Save the symbol and close the symbol editor.
7. Afterwards, execute "Verify Application" in the context menu of the "WinStudio" folder. Thus, all existing instances of the symbol in the project are updated.

Adding tooltips to user-defined properties

For each user-defined property available in the master symbol, a description can be defined simplifying the configuration of the symbol instances.

After creating a symbol, open it with the symbol editor, right-click in the symbol editor (not on the symbol itself) and choose **Edit Symbol Properties** from the context menu.

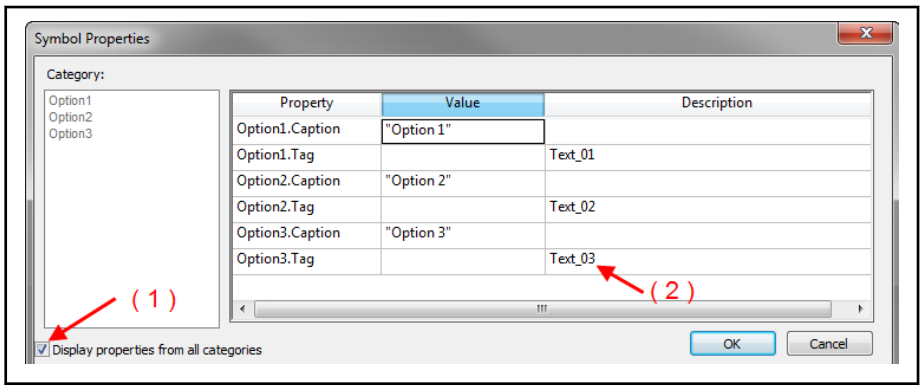


Fig. 8-70: Edit symbol properties

By means of checkbox (1), **Display properties from all categories**, information of all objects can be displayed in a list.

In column (2), **Description**, a text is to be entered that is later used as tooltip.

When inserting and configuring a symbol instance in the screen, the description is displayed as tooltip if the mouse cursor is moved over the name of the property as displayed in the following figure:

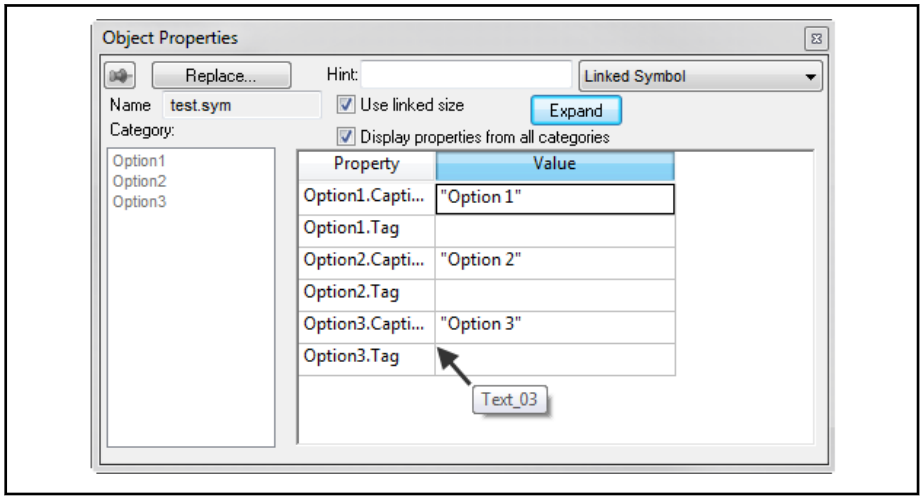


Fig. 8-71: Tooltip indicating the description of the property

Symbol password protection

Passwords can be defined for all user-defined symbols to prevent unauthorized editing or analysis. To protect a symbol, proceed as follows:

1. In the "User Symbols" folder, right-click on the desired symbol file (**.sym**) and choose **Change Password** from the context menu. The "Edit Protection" dialog is displayed
2. Enter the password in the **New Password** field
3. Enter the password again into the **Confirm Password** field
4. Click on **OK**

Once this is done, a prompt for the password is displayed whenever the symbol is to be edited or the link to an instance of the symbol is to be removed.

Adding symbols into a WinStudio screen

To insert a symbol into a screen and complete its user-defined properties, proceed as follows:

1. Open the desired screen from the Screens folder on page 170 or insert a new screen
2. *To open the symbols library, proceed as follows:*
   - Open the **WinStudio** ▸ **Tools** context menu and select "WinStudio library" or
   - Open the **User symbols** context menu and select "WinStudio library" or
   - Right-click on the WinStudio screen editor where the symbol is to be inserted and click **Insert Linked Symbol** in the context menu
3. Select the symbol from the symbols library and click on the WinStudio screen:

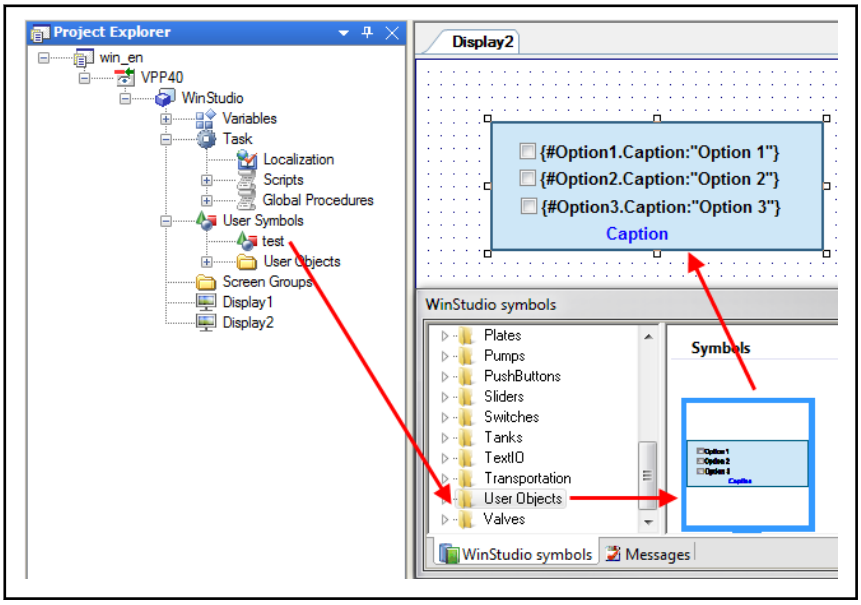


Fig. 8-72: Positioning a symbol in a SCREEN

If the symbol is inserted it can be edited like any other object in the screen. It can be aligned and distributed on page 265 with other objects and animations on page 246 can be applied

However, first of all, the user-defined properties for this instance of the symbol have to be completed

4. Open the Object Properties for the symbol
5. Assign real process values (texts in quotation marks or tag names) to the "Tag" transfer parameters

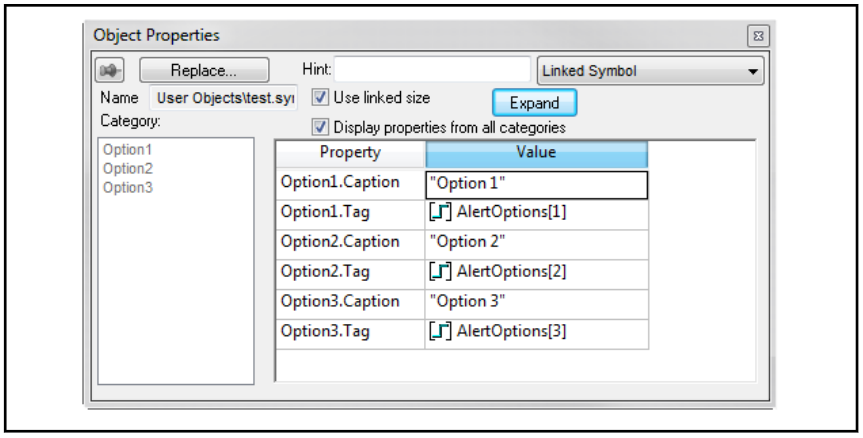


Fig. 8-73: Completed properties for the symbol

6. Click **OK** to close the "Symbol properties" dialog and close afterwards the "Object properties" dialog

At runtime, the placeholders in Engineering {} are replaced by the true (assigned) process values (texts, tag values).

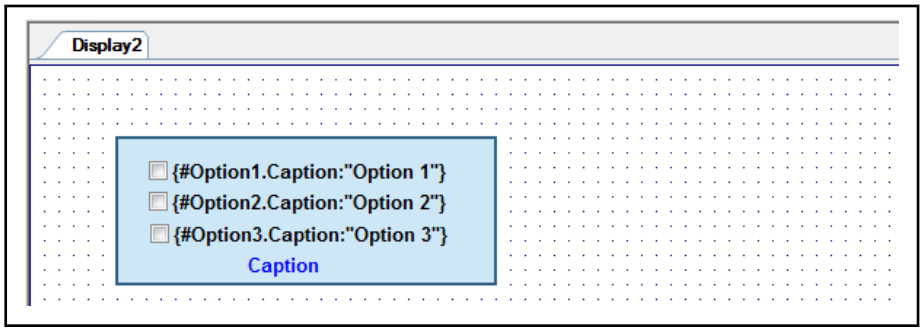


Fig. 8-74: Symbol during editing

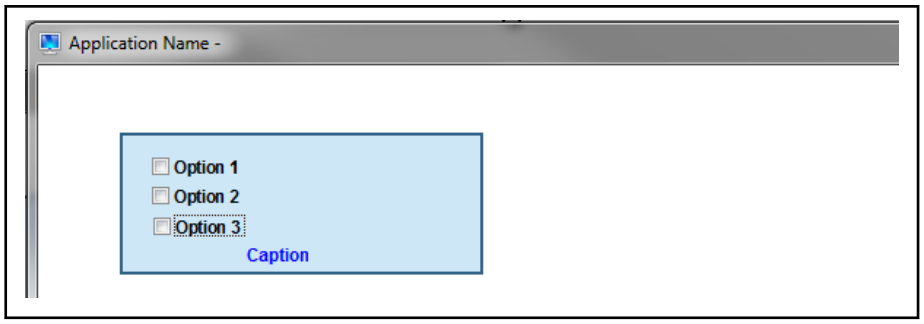


Fig. 8-75: Symbol at runtime

Remember that the assigned process values for each instance of a master symbol are independent of all other instances of this master symbol.

However, if changes are made to the master symbol file, these changes are automatically applied to all instances.

ActiveX control

In the **WinStudio Screen Tools** tool window, click on **ActiveX control**. The "Insert ActiveX control" dialog is displayed to select ActiveX elements and insert them into WinStudio screens.

If the dialog is displayed, it provides a list of all ActiveX components that are registered on the PC.

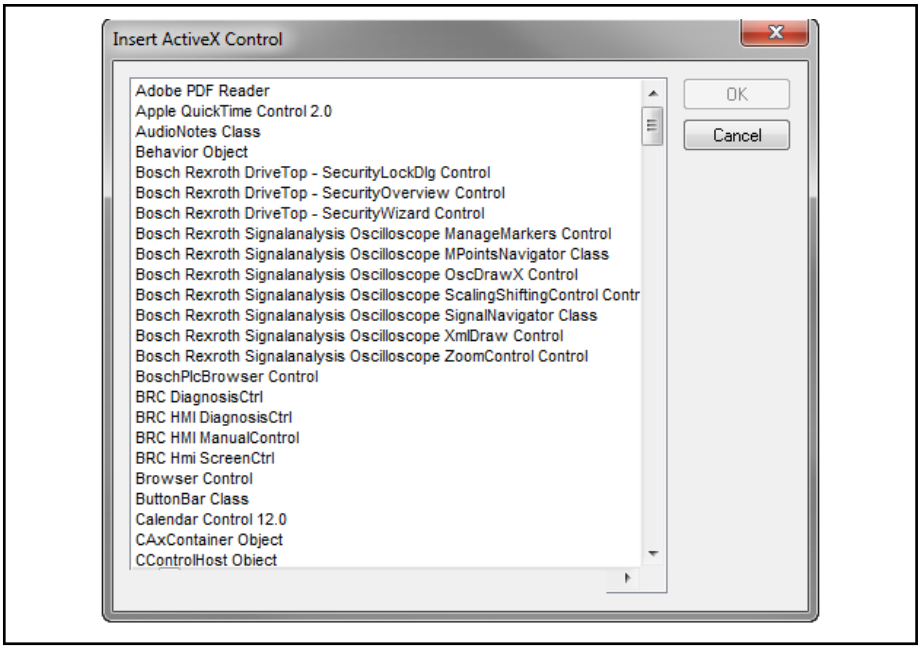


Fig. 8-76: Insert ActiveX control dialog

If the ActiveX controls are to be used in the project, the runtime stations are to have the same controls already installed and registered.

As the "Automatic Download" and "Automatic Registration" functions are mostly deactivated on target devices for safety reasons, ActiveX controls are not always available after project download.

To be able to use ActiveX controls despite these security settings, they have to be transferred to the target device and registered manually. If the ActiveX control is registered for the first time, it can be updated with the project by adjusting the project file (projekt_name.app):

If the automatic download of ActiveX control elements is still to be enabled, this can be achieved by manually editing the project file (`project_name.app`) to add the following settings:

Project file projekt_name.app

```
[UsedControls]
EnableDownload=1
Count=number of controls

[Used Control1]
CLSID= class ID of the ActiveX control
Version = version of the ActiveX control
Codebase= URL of the ActiveX control file, or of the .CAB file that contains the ActiveX control files
RegFile1= name of a specific .OCX or .DLL file within the .CAB file; see below
RegFilen= name of a specific .OCX or .DLL file within the .CAB file; see below
...
[UsedControln]
...
```

The "CLSID" and "Version" settings are required for each ActiveX control and have to match the ID and version of the actual control file(s) to which "Codebase" is linked.

This allows a runtime station to check the ActiveX control against elements that are already registered. If the settings do not match, the runtime station may unnecessarily download the same control element again.

If the "CLSID" and "Version" settings for an ActiveX control are unknown, they can be found in the registry key of an already installed and registered control element. Search for the control file in `HKEY_CLASSES_ROOT\CSLID` in the Windows Registry.

The URL for the "Codebase" setting can be either absolute or relative to the "home" directory of the web server.

Example:

```
Codebase=http://"server_adresse"/AddOns/IndDateTimePick.ocx
      …or…
Codebase=AddOns/IndDateTimePick.ocx
```

The "Regfile" settings are only required if "Codebase" is linked to a .CAB file (container for multiple ActiveX controls). In this case, a "Regfile" entry has to be created for each received ActiveX control.

ActiveX controls are components designed according to a standard. These elements can be implemented into "Rexroth WinStudio" application screens and executed at runtime. ActiveX controls can provide the following interfaces:

- **Properties**: Variables whose values can be read and/or written for the project (e.g. object color, file name, URL, etc.)
- **Methods**: Functions from the ActiveX object that can be triggered by the project (e.g. open dialog, execute calculation, etc.)
- **Events**: Internal messages that can trigger the execution of expressions in the project (e.g. Mouse_Click, Download_Completed, and so forth)

The names of the properties, methods and events supported by each ActiveX object depend on its own implementation.

There are two different ways to interface the project with the ActiveX control elements:

- By using the ActiveX function

  XGet() on page 891,

  XSet() on page 893 and

  XRun() on page 892

  or
- by using the "Object Properties" window to configure the object

Double-click on the ActiveX control to open the "Object Properties" dialog.

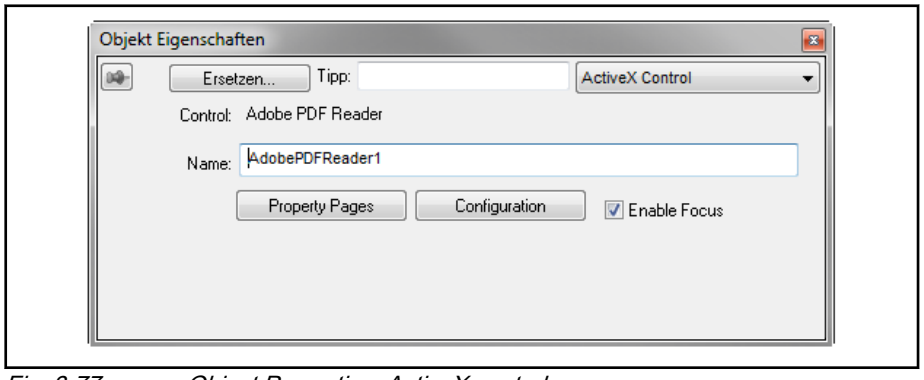


Fig. 8-77: Object Properties: ActiveX control

The "Object Properties" window displays the name of the ActiveX control. Generally, each ActiveX control is either a `*.dll` or a `*.ocx` file registered on the local computer.

Assign the ActiveX control a unique name (alias) in the "Name" field (e.g. `MyControl`). This name is required for function calls from the scripting language.



You should not configure two ActiveX controls on the same screen with the same name. For instance, if two "Windows Media Player" ActiveX controls are inserted on the same screen and assign the name MyMP1 to one object (Name field), you cannot assign the same name to the second object on the same screen. You would have to assign the name MyMP2, for example, to the second object.

The **Property Pages** button opens the standard window for configuring the static properties (if available). The layout and the options in this dialog depend on the implementation of each ActiveX control. Use this interface to set properties that are not to be changed during runtime (fixed properties).

The **Configuration** button on the "Object Properties" window opens a configuration dialog that enables the following tasks:

- Connect tags to properties of the ActiveX control
- Trigger methods from the ActiveX control based on tag change
- Script configuration that are executed at certain events in the ActiveX control

The following sections describe configuration of these interfaces.



Although the Configuration dialog displays the list of all properties, methods and events, only the items that are required for the application need to be configured.

The screenshots used in the following sections depict the "Acrobat 3D Office" ActiveX control.

The names of the properties, methods and events vary for each ActiveX control, but the configuration interface is the same. The procedure described in the following is the same for all controls.

Configuring properties

The "Properties" tab contains a table with one line for each parametrizable property:

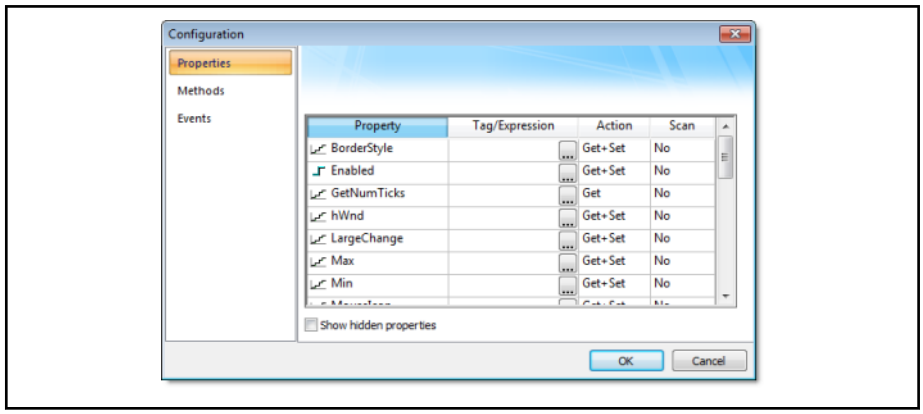


Fig. 8-78: Configuration dialog – Properties tab

- **Property** column: Description of all properties available in this ActiveX control including the respective type ID

Property symbol	Type
	Boolean
	Integer
	Real
	String

- **Tag/Expression** column: The tag configured in this field is associated with the respective property of the ActiveX control. The "Action" column defines whether the value of this tag is written to the ActiveX property or if the value of the ActiveX property is written to this tag (or both).



In this field, an expression can be configured if the result of an expression is to be written to the property of the ActiveX object.

However, in this case, the value of the property cannot be read back to one tag (unless the XGet() function is used).

- **Action** column: In this column it is defined whether the parameter in the "Tag/Expression" column is read from the control or written to the control. In case that the value is to be read and written (Get+Set), the order of the actions has to be set (refer to the following table):

  The "Action" column is automatically configured to "Set" if the expression in the "Tag/Expression" column cannot be written to.

Action	Description
Get	Reads the value of the ActiveX property and writes it to the tag configured in the **Tag/Expression** field
Set	Writes the value from the tag or expression configured in the **Tag/Expression** field into the ActiveX property
Get+Set	Executes both actions (**Get** and **Set**). However, when opening a screen with the ActiveX control, "WinStudio" executes the **Get** command before executing any **Set** commands. This means that when "WinStudio" opens the screen in which the ActiveX control is configured, the tag configured in the **Tag/Expression** field is updated with the value from the ActiveX property.
Set+Get	Executes both actions (**Get** and **Set**). However, when opening a screen with the ActiveX object, "WinStudio" executes the **Set** command before executing any **Get** commands. This means that when "WinStudio" opens the screen in which the ActiveX element is configured, the tag configured in the **Tag/Expression** field is updated with the value from the ActiveX property.



If the value of the property is "read-only" (cannot by overwritten by the project), the **Action** field is automatically set to **Get**.

- **Scan** column: Defines the polling method to get values from the ActiveX properties according to the following table:

Scan	Description
No	The value of the ActiveX property is only read and written to the tag configured in the Tag/Expression field if the screen with the ActiveX object is open and if the ActiveX object sends a message to "WinStudio" to update this tag
Always	"WinStudio" continuously reads the value of the ActiveX property and updates the tag configured in the "Tag/Expression" field with this value

Some ActiveX controls automatically send a message to the container (e.g. to the runtime environment) to indicate that a value has changed and can be read (Get).

However, some ActiveX controls do not support sending messages if values are changed.

In this case, the only way to get the updated values of the ActiveX properties is to keep calling these values from the ActiveX control (Scan=Always).

Configuring methods

The Methods tab provides a table with the following fields:

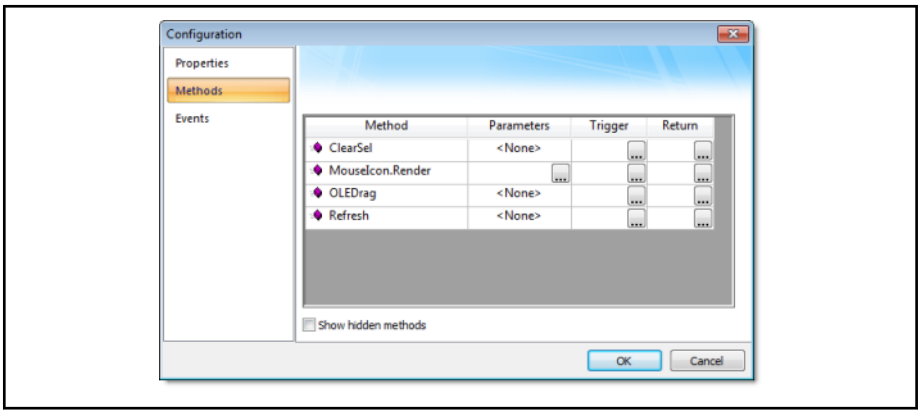


Fig. 8-79: Configuration dialog – Methods tab

- **Method** column: Lists all methods available from the ActiveX object
- **Parameters** column: The tags configured in this column are linked to the transfer parameters of a method.

  If a method does not support any parameters, the text <None> is displayed in the Parameters field.

  If the method expects more than one transfer parameter, a tag separated by a comma (",") can be entered for each parameter.

  For example, `TagA , TagB , TagC`.

  If the method is executed, either the value of the tags are written to the parameters of the method (input parameters), or the ActiveX writes the value of the parameters to the tags (output parameters) after the method is executed.

On clicking the Browse button (...), a list of parameters supported by the method is displayed to associate one tag with each parameter.

- **Trigger** column: If the tag configured in this column changes its value, the respective method of the ActiveX control is executed

- **Return** column: The tag configured in this column receives the return value of the method (if available)

Configuring events

The Events tab provides a table with the following fields:

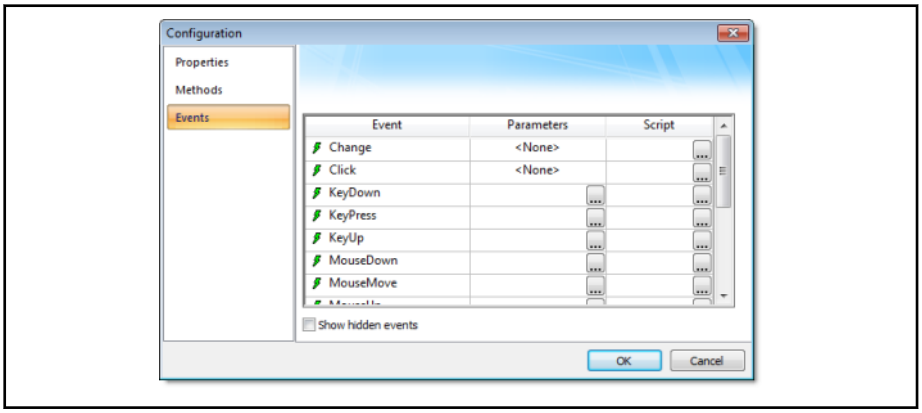


Fig. 8-80: Configuration dialog – Events tab

- **Event** column: Lists all events available from the ActiveX object
- **Parameters** column: The tags configured in this column are linked to the transfer parameters of an event

  If an event does not support any parameters, the text <None> is displayed in the Parameters field.

  If the event expects more than one transfer parameter, a tag separated by a comma (",") can be entered for each parameter

  For example, **TagA , TagB , TagC**.

  If the event occurs, either the values of the tags are written to the parameters of the event (input parameters) or the parameter values are written to the tags (output parameters).



On clicking the Browse button (...), a list of parameters that can be associated with a tag and are supported by the event is displayed.

- **Script** column: The script configured in this field is executed if the event is triggered by the ActiveX control



Click the Browse button (...) to open the program editor. The program can be created in VBScript or in the "Rexroth WinStudio" programming language

.NET controls

.NET components are designed according to the Microsoft .NET framework, which is a standard for modular programming technologies. As WinStudio is a .NET container, .NET components can be integrated and executed in WinStudio screens. The actual functions of .NET components are contained in a **.NET control**, which provides the configuration dialogs.

.NET components include the following interfaces:

- **Properties**: Variables whose values can be read and/or written to the project (e.g. object color, file name, URL, etc.)
- **Methods**: Functions from the .NET component that can be triggered by the project (e.g., open dialog, execute calculation, etc.)

- **Events**: Internal messages that can trigger the execution of expressions in the project (e.g. Mouse_Click, Download_Completed, etc.)

.NET controls can differentiate in representation and function, depending on the configured properties, methods or events.

NOTICE **If .NET components are used in your project...**

If .NET components are used in the project, make sure that the target system (runtime station) can support the same components and that they are properly installed and registered. Your project includes links to the .NET Components; however, the installation of these components on the target system must be done separately.

Furthermore, when .NET components are used on screens open in remote Thin Clients, the .NET components must also be manually installed on the Thin Client stations. The Microsoft Windows operating system installs a large selection of components by default, but additional components are offered by third-party providers. Consult your .NET Component provider for further information on how to install.

Selecting and placing .NET control

To select and insert .NET controls into WinStudio screens, proceed as follows:

1. In the **WinStudio Screen Tools** tool window, click on **.NET control**
2. If the ".NET framework components" dialog opens (as in the following figure), it contains a list of all .NET components that are registered on the computer

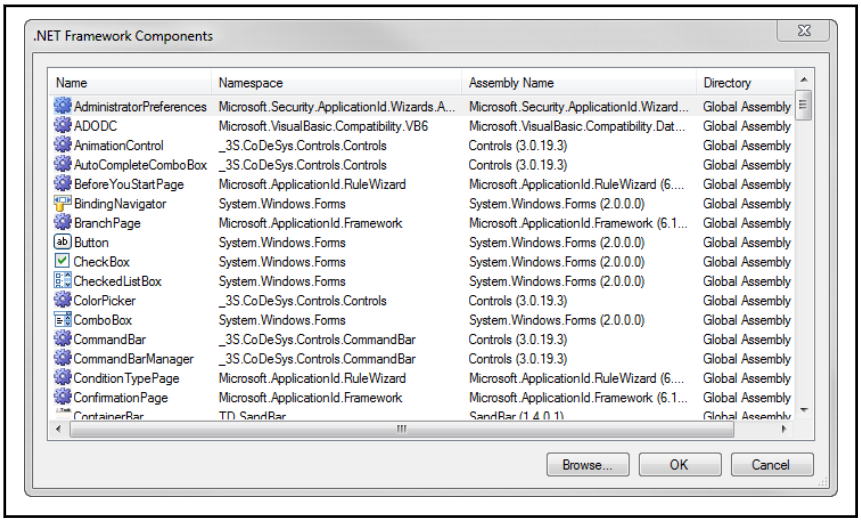


Fig. 8-81: .NET framework components dialog

3. Select a component from the list, and click **OK** to place it in the screen. Click on the **Browse...** button to search for unregistered components on the computer. / Click **Browse...**

Registered .NET components are typically stored in the following directory:

`C:\WINDOWS\Microsoft.NET\Framework\`

Unregistered .NET controls can be added to the selection list in the ".NET framework components" dialog by adding the following entry to the `projekt_name.APP` file:

```
[Execution Environment]
DotNetControlPath=OptionalPath
```

Example:

```
[Execution Environment]
DotNetControlPath=C:\DOTNET CONTROLS BACKUP
```

Thereafter, the .NET framework components dialog lists all registered components "and" all components found in the specified directory

4. By default, a new .NET control object is placed in the upper-left corner of the screen. Click on the object and drag it to the desired location
5. Once the object is placed, double-click on it to open its "Object Properties" dialog

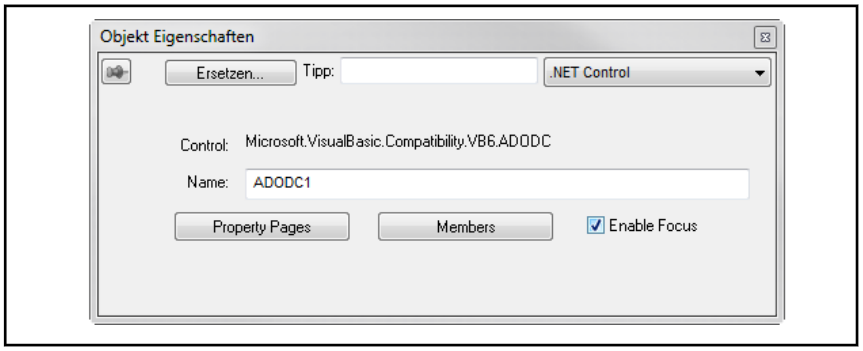


Fig. 8-82: Object properties: .NET control

The "Object Properties" dialog shows the name of the .NET component. A unique name (alias) has to be assigned to the component in the **Name** field (e.g. CheckBox1). This name is used to reference the component when using the scripting languages (VBScript on page 925 and Integrated script language on page 595).

You should not configure two .NET Control objects on the same screen with the same name. For instance, if two Checkbox components are placed on the same screen and the name `Checkbox1` is assigned to one object (**Name** field), the same name cannot be assigned to the second object on the same screen. The name `Checkbox2`, for example, would have to be assigned to the second object.

The **Property Pages** button opens the standard window for configuring the static properties (if available). The layout and the options in this dialog depend on the implementation of each .NET component. Use this interface to set properties that are not to be changed during runtime (fixed properties).

The **Members** button on the "Object Properties" dialog opens additional dialogs that enable the following tasks:

- Associate tags to properties of the .NET component
- Trigger methods from the .NET component by means of a tag change
- Configuration of scripts that are to be executed if events from the .NET component occur

The following sections describe configuration of these interfaces.



Although the "Members" dialog displays a list of all properties, methods and events, only the items that are required for the respective application need to be configured.

The following figures illustrate the .NET control "checkbox". Although the names of properties, methods and events varies by .NET control, the configuration interface is the same for any .NET component. The elements (properties, methods, events) described here apply for all .NET controls.

Configuring properties

The "Properties" tab provides a table with the following fields:

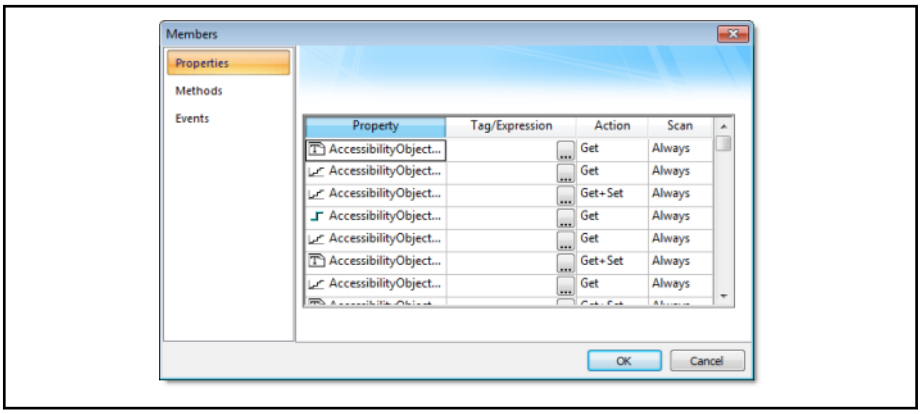


Fig. 8-83: Members dialog - Properties tab

- **Property** column: Lists all properties available from the .NET component, and indicates their types:

Property symbol	Type
	Boolean
	Integer
	Real
	String

- **Tag/Expression** column: The tag configured in this field is associated with the respective property of the .NET component. The Action column defines whether the value of this tag is written to the property or if the value of the property is written to this tag (or both)
- **Action** column: In this column, it is defined whether the parameter in the "Tag/Expression" column is read from the control or written to the control according to the following table:

Action	Description
Get	Reads the value of the property and writes it to the tag configured in the **Tag/Expression** field
Set	Writes the value from the tag or expression configured in the **Tag/Expression** field to the property

Action	Description
Get+Set	Executes both actions (**Get** and **Set**). However: When opening a screen with the .NET component, WinStudio executes the **Get** command before executing any **Set** commands. This means that when "WinStudio" opens the screen in which the .NET component is configured, the tag configured in the **Tag/Expression** field is updated with the value from the property
Set+Get	Executes both actions (**Get** and **Set**). However: When opening a screen with the .NET component, WinStudio executes the **Get** command before executing any **Set** commands. This means that when "WinStudio" opens the screen in which the .NET component is configured, the property is updated with the value in the **Tag/Expression** field

If the value of the property is "read-only" (cannot by overwritten by the project), the **Action** field is automatically set to **Get**.

- **Scan** column: Defines the polling method to get values from the properties. For .NET components, all properties scan **Always** by default. "WinStudio" continuously gets the value of the property and updates the tag configured in the **Tag/Expression** field with this value

Configuring methods

The "Methods" tab provides a table with the following fields:

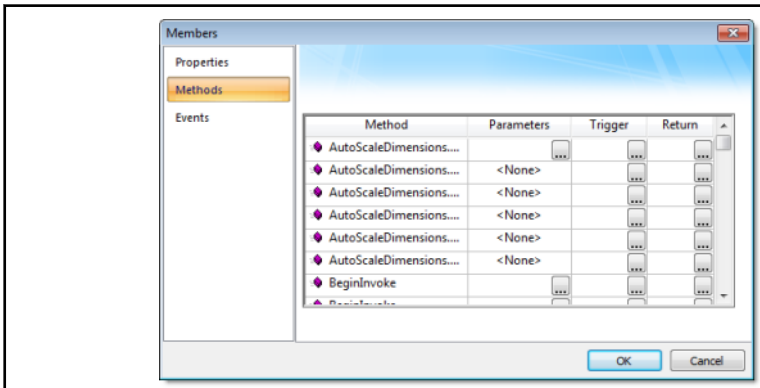


Fig. 8-84: Members dialog – Methods tab

- **Method** column: Lists all methods available from the .NET component
- **Parameters** column: The tags configured in this column are linked to the transfer parameters of a method. If a method does not support any transfer parameters, the text <None> is displayed in the Parameters field. If a method expects more than one transfer parameter, a tag separated by a comma (",") can be entered for each parameter

  Example:

  ```
  TagA , TagB , TagC.
  ```

On clicking the Browse button (...), a list of parameters supported by the method is displayed to associate one tag with each parameter.

If the method is executed, either the value of the tags is written to the transfer parameters of the method (input parameters), or the .NET control writes the value of the transfer parameters to the tags (output parameters) after the method is executed.

- **Trigger** column: If the tag configured in this field changes its value, the respective method of the .NET control is executed
- **Return** column: The tag configured in this field receives the return value of the method (if available)

Configuring events

The "Events" tab provides a table with the following fields:

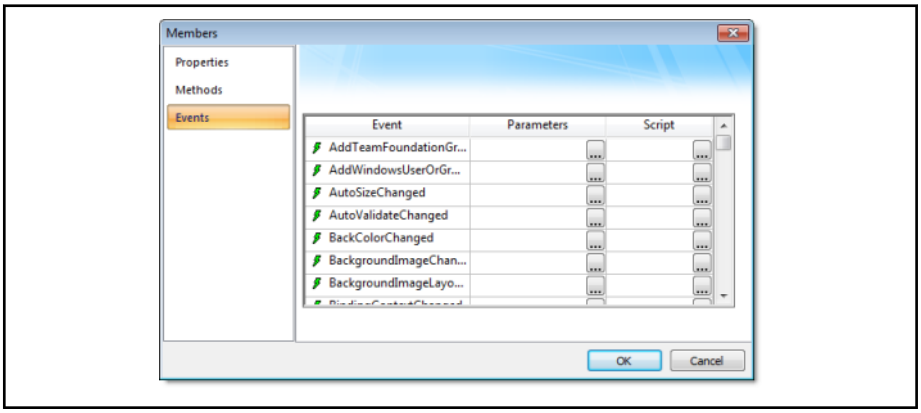


Fig. 8-85: Members dialog - Properties tab

- **Event** column: Lists all events available from the .NET control
- **Parameters** column: The tags configured in this column are linked to the transfer parameters of events. If an event does not support any transfer parameters, the text <None> is displayed in the Parameters field. If the event expects more than one transfer parameter, a tag separated by a comma (",") can be entered for each parameter

  Example:

  ```
  TagA , TagB , TagC.
  ```



On clicking the Browse button (...), a list of parameters that can be associated with a tag and are supported by the event is displayed.

If the event occurs, either the value of the tags is written to the transfer parameters of the method (input parameters), or the .NET control writes the value of the transfer parameters to the tags (output parameters) after the event occurs.

- **Script** column: The script configured in this field is executed if the event is triggered by the .NET control



Click the Browse button (...) to open the program editor. The program can be created in VBScript or in the "Rexroth WinStudio" programming language

Adding a link to an external graphic file

To display graphics in your WinStudio screens, use the object **Graphic** from the static objects (refer to the tool window **WinStudio screen tools**). This object establishes a link to the graphic file and shows it in the WinStudio screen. As it is a link, the identical graphic can be used at different positions in your WinStudio application. Additionally, the graphic file can be edited using an external graphic program without inserting the changed graphic again.

It is recommended to insert the graphic to be used to the WinStudio project first. This ensures that the graphic is also available if the project is applied to a visualization device.

To add the graphic files to the project, proceed as follows:

1. Open the context menu of the "Resources" folder and select "Add file...".
2. Select the file using the file browser and confirm the selection with "Open".

The selected graphic file can be copied to the project and displayed in the "Resources" folder and it can be used anywhere in your WinStudio screens. The file can be subsequently renamed in the “Resource” folder. Double-click on the file to open the editor set for the respective data type in the operating system. Thus, the graphic can be edited and saved again. The changes are then applied to all places of use of the linked graphic.

Inserting a graphic into a WinStudio screen:

1. Click on **Graphic** in the **WinStudio Screen tools** tool window.

   The "Open" dialog is displayed. It shows all files in the directory ..\web \Resources.

☞ Use the file browser to go to any directory and to select a graphic file there. This file is not considered when applying the project to the visualization device. Ensure that the graphic file is located in the same directory on the visualization device.

---

2. Select a graphic file and click on **OK**.

   The following graphic formats are supported in WinStudio projects which run on Windows target systems:

   - Windows bitmap (`*.BMP`)
   - AutoCAD DXF (`*.DXF`), only versions 10 and 11
   - JPEG (`*.JPG, *.JPEG`)
   - PNG (`*.PNG`)
   - TIFF (`*.TIF, *.TIFF`)
   - Windows meta file (`*.WMF, *.EMF`)

   The following graphic formats are supported in WinStudio projects which run on Windows Embedded target systems:

   - Exchangeable Image File Format (`*.Exif`), often used in digital cameras
   - GIF (`*.GIF`)
   - JPEG (`*.JPG`)
   - PNG (`*.PNG`)
   - TIFF (`*.TIFF`)
   - Windows Bitmap (`*.BMP`)

Rexroth WinStudio v7.3+SP4 and higher uses a new graphic system to display graphics on Windows Embedded devices. This new system requires that the new Windows Embedded version runs on your device and that all supported graphic file types are integrated into the runtime image of the device. If a graphic is not correctly displayed on your device, there are two solutions.

2.1 Disable the new graphic system and use the Legacy graphic system instead. However, only .jpg, .png and .bmp images are then supported in your project. To use the Legacy system, open your project file (Projectname.APP) in a text editor and edit the following property:

```
[Objects]
DrawingSystemID=<0 for new system, 1 for legacy system>
```

2.2 It can be required to update the operating system on your Windows Embedded device to ensure that all supported graphic file types are integrated. (The operating system is normally installed by the device manufacturer.) However, this is beyond the scope of this documentation. For more information, please contact the technical Bosch Rexroth support.

The graphic is added to the WinStudio screen as link.

3. Double-click on the graphic object.

   The "Object properties: Linked picture" dialog is displayed.

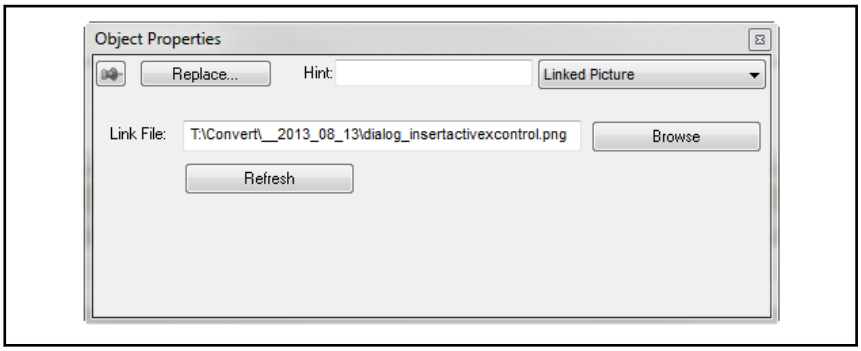


Fig. 8-86: Object properties: Linked picture

4. Check the link in the **Link file** field. If the image file is stored in the web folder, the link is a relative file path. If the graphic file is saved to another directory, the absolute path of the file is displayed.

   It is also possible to enter subfolders in the web folder.

   If, for example, `MyPictures\Picture1.BMP` is entered, the program searches for the image file in

   `projectname\Web\MyPictures\Picture1.BMP`.



The file extension is not always required for a link to work. In projects which run on Windows target systems, the extension PNG is used by default if no file extension is specified. To change this, open your project file (`Projectname.APP`) in a text editor and edit the following property:

[Viewer]

```
DefaultLinkedPictureExtension=<image file extension>
```

In projects which run on Windows Embedded target systems the extension `BMP`is used by default if no file extension is specified. This cannot be changed.

5. To change the link at runtime to dynamically display another graphic, replace the file path by a project tag:
   5.1 In the **Link file** field, select the file path and then copy it to the clipboard.
   5.2 Replace the file path in the “Link file” field with the name of a string tag in curly brackets (e.g. `{MyLinkedPicture}`).
   5.3 If the tag does not exists, you are prompted to create it. It has to be created as string tag.
   5.4 Define the file path copied to the clipboard as the initial value for the tag either by inserting the file path in the initial property of the tag (in the data sheet project tags view) or by configuring the Startup script on page402 to set the tag value when the project is running.

   If the project tag has been defined properly, the link is refreshed every time the tag value changes at runtime. Remember that the tag value has to have the same format as a normal link. Either a relative file path for a file stored in the web folder or an absolute file path for a file stored elsewhere on the target system or in the network.
6. If parts of the graphic are shown as transparent for the background, select a transparent color:
   6.1 In the **Transparent color** group, select either **Color code** or **Tracker** from the list.
   6.2 If a **Color code** has been selected, enter a tag/expression which provides the 24-bit color code of the desired transparent color. For further information on Windows color codes, refer to Color interface on page 277 and WdColor list.
   6.3 If **Tracker** was selected, click and drag the tracker at the screen object until it is for example above the desired transparent color. The tracker is an additional handle at the screen object which appears directly in the bottom right corner of the object in the beginning. Moving the tracker on the object does not move the object itself and does not change its size.



Transparent colors are not supported for Windows meta files (`*.WMF, *.EMF`)

7. Close the "Object properties" dialog.

8.2.5 Applying animations to screen objects

Applying animations to screen objects, general information

Use the "WinStudio dynamic functions" toolbar to apply animations to a screen object or an object group. With animations, tags can be used to change object properties at runtime. Some animations also enable execution of commands or insert values (command values) to the tags.



Fig. 8-87: "WinStudio dynamic functions" toolbar

Dynamic objects are only displayed as active (in color) if the selected screen object or screen group supports this animation (e.g. text input/output for text fields). Move the cursor over an active WinStudio dynamic object to display the function in plain text.

Command animation

Symbol: 

In the **WinStudio Dynamic Functions** toolbar, click on **Command** to add animations to a selected object or object group. This dynamic function enables triggering of actions or execution of program functions by mouse click during runtime. Double-click on the object to view its object properties.



Fig. 8-88: Object Properties: Command

The configuration dialog provides a separate tab for each event supported by the object. Please note that more than one event can be configured simultaneously for the same dynamic function:

- **On Down**: Executes the command/script once when the object is clicked with the left mouse button
- **On While**: Executes the command/script continuously while the mouse button is pressed. The execution period (in milliseconds) for the command/script is defined in the "Rate" field. An exception is the option VBScript that is executed as quickly as possible
- **On Up**: Executes the command/script once when object is released by the left mouse button
- **On Right Down**: Executes the command/script once when the object is clicked with the right mouse button
- **On Right Up**: Executes the command/script once when the object is released by the right mouse button
- **On Double Click**: Use this item to execute the command/script once by double-clicking on the object with the left mouse button

- **On touch, On start of touch, While touched, On end of touch**: These events are used for multi-touch gestures.

  For further information, refer to Touch events on page 299.



The events **On Right Down, On Right Up** and **On Double Click** are not supported in projects running on Windows Embedded target systems.



An asterisk (*) on an Events tab indicates that something is configured for this event. In this way it is easier to see at a glance which events are configured.

Type selection field: This setting defines the type of action that is to be executed if the event occurs. Please observe that for each event (e.g. on pressing or releasing) an individual type selection can be made. This enables different configuration of the same dynamic functions for different events. The following types are supported:

Type	Description
Built-in Language	Enables configuration of a script using the "WinStudio" integrated language. If this type is selected, up to 12 expressions for each event can be configured in the **Expression** column. The expressions are executed sequentially from the first line to the last one if the event is triggered. The result of each expression is written to the tag configured in the **Tag** column (if any). For further information, refer to Appendix: WinStudio-internal scripting language on page 595
VBScript	Enables configuration of a script using the standard VBScript language. If this type is selected, a script can be configured in the VBScript editor for the **command animation**. For further information on VBScript language, refer to Overview of VBScript on page 925
Open screen	Enables configuration of the **Command animation** to open a specific screen if the event is triggered during runtime. This type is equivalent to the function Open on page 666. Enter the screen name into the **Open screen** field or select a screen via the Browse dialog. A string tag can also be entered in this field between curly brackets {TagName}. If the event is executed, the project attempts to open the named screen
Close screen	Enables configuration of the **command animation** to close a specific screen if the event is triggered during runtime. This type is equivalent to the function Close on page 665. Enter the screen name into the **Close screen** field or select a screen via the browser dialog. A string tag can also be entered in this field between curly brackets {TagName}. If the event is executed, the project attempts to close the named screen
Set variable	Allows you to configure the **Command** animation to set a tag when the event is triggered during runtime. Enter the tag name into the **Set variable** field or select a tag via the Browse dialog. If the event is executed, the project writes the value 1 to the configured variable
Reset variable	Enables configuration of the **command animation** to reset a variable if the event is triggered during runtime. Enter the tag name into the **Reset variable** field or select a tag via the Browse dialog. If the event is executed, the project writes the value 0 to the configured variable
Toggle variable	Enables configuration of the **command animation** to toggle a variable if the event is triggered during runtime. Enter the tag name into the **Toggle variable** field or select a tag via the Browse dialog. If the event is executed, the project toggles the value of the tag configured in this field

Config button: Opens an extended view of the object properties with additional configuration settings

Back to button: Click to return to the object properties of the underlying button object

Configuration dialog Extended object properties dialog with additional configuration settings...

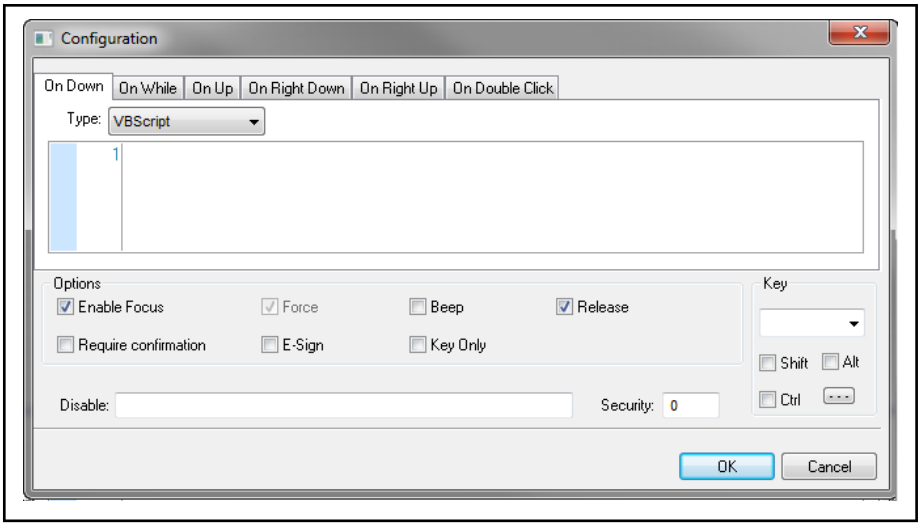


Fig. 8-89: Configuration dialog

The event tabs (e.g. "On Down", "On While", etc.) and the **Type** selection list are the same as in the "Object Properties" dialog described above. The additional configuration settings are shared for all events:

- "Options" pane:
  - **Enable Focus**: If this option is selected, the object on which the command animation has been executed, can receive the input focus during runtime and can thereby be selected by navigating with the <Tab> key
  - **Force**: If this option is selected, each project tag which receives a value triggers events as if the tag had changed even if the new value is the same as the old value. If a tag for example has a value of 0, the command animation executes a process which writes 0 to this tag. All other tasks in the project runtime environment recognize that the tag has changed even if this is not the case. This option is useful to make sure that events triggered by tag changes (e.g. **Write on tag change** on a communication driver) are always triggered after the command animation is executed.

    Please note that if the value of the tag does not really change, the time stamp of the tag (`tagname->Timestamp`) is not changed.

    Force applies to the process executed by the command animation itself and to all global procedures which are called during this process provided that they run on the project runtime client on which the command animation is used (i.e. on the device on which the button is pressed).

    Force does not apply to global procedures which are executed on the project runtime server via the RunGlobalProcedureOnServer on page 711 function, even if the function is called during the process which is executed by the command animation. If tag changes in global procedures which are executed on the server are to be forced, use the ForceTagChange on page 812 function.

☞ For projects created with Rexroth WinStudio v7.0 or earlier, the Force option is enabled by default and cannot be disabled.

For projects created with "Rexroth WinStudio v7.x" or earlier, the **Force** option is enabled by default and cannot be disabled

  - **Beep**: If this option is enabled, a short beep is returned when the command is executed. This option is useful to provide an audio feed-back indicating that the command was executed. It does not indicate, however, if the action triggered by the command animation was successful or not
  - **Release**: If this option is enabled, the `On Up` event is executed when the cursor (or finger) is dragged out of the object area (whether the button was released or not). This option is useful to make sure that the `On Up` event is always executed after an `On Down` event, even if the mouse cursor is moved out of the object area before the button is released.
  - **Require confirmation**: If this option is enabled, a confirmation is required before executing the command. This option is useful for decreasing the accidental triggering of critical events during runtime
  - **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the command
  - **Key Only**: If this option is enabled, "only" the keyboard shortcuts (configured in the "Key" pane described below) can be used to execute commands
- **Disable**: Disables user action if the result of the expression configured in this field is TRUE (value different from 0)
- **Security**: Security on page 461, an access level required to execute the command animation during runtime
- **Key** pane: Keyboard shortcut to trigger the events "On Down", "On While" and "On Up" using the keyboard. (In other words, pressing this keyboard shortcut is the same as clicking the left mouse button.) This option is particularly useful if the keyboard is the only available physical interface to interact with the user interface during runtime
  - **Shift**, **Ctrl** or **Alt**: Click to create a combination key, meaning the Shift, Ctrl and/or Alt key has to be pressed with the key specified in the drop-down list
  - Click the browse button (**...**) to open the "Key modifier" dialog, to modify the combination keys. **Left**, **Right** or **Left or Right** can be selected to specify the position on the keyboard of the Shift, Ctrl or Alt key in the key combination. If **Left or Right** is selected, the command is executed any time one of these keys is pressed in combination with the key specified in the selection list



If you have defined custom keys for your project, you can select them in this list. For more information, see Define custom keys for selected screen objects.

Hyperlink animation

Symbol: 

In the **WinStudio Dynamic Functions** toolbar, click on **Hyperlink** to add a hyperlink to a selected object or object group.

Hyperlinks enable opening a standard browser with a predefined URL during runtime.

Double-click on the object to open the "Object Properties" dialog.

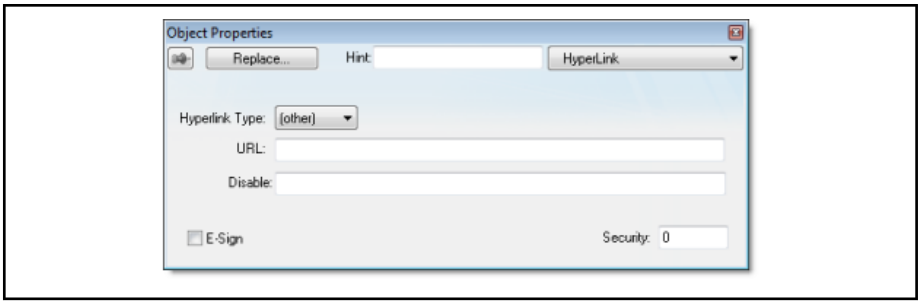


Fig. 8-90: Object Properties: Hyperlink

Use this dialog to specify the following parameters:

- **Hyperlink Type**: Select a URL protocol from the selection list. The project uses this protocol when loading a URL
- **URL**: Enter the URL address to be loaded

It is not necessary to enter the protocol type in the URL field.

When selecting a protocol type from the **Hyperlink Type** list, the project automatically adds the prefix of the protocol to the URL address.

- **Disable**: Enter a value greater than zero into this field to disable the hyperlink command animation for the selected object(s)
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the animation
- **Security**: Type a value in this field to specify a security level for the object(s), as defined under Project security on page 461: If a user logs on but does not have the required security level, the project disables the hyperlink command for the object(s)

Bargraph Animation

Icon: 

In the **WinStudio Dynamic Functions** toolbar, click on **BarGraph** to add a bargraph property to a selected object or object group. Double-click on the object to open its "Object Properties" dialog.

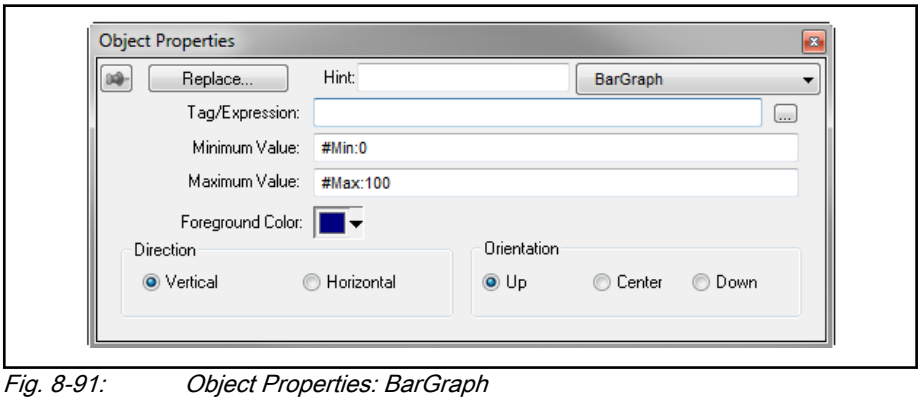


Fig. 8-91: Object Properties: BarGraph

Use the "Object Properties" dialog to specify the following parameters:

- **Tag/Expression**: Enter a tag or expression that is to be displayed as bar height or width. Click on the  button to browse for existing tags or expressions
- **Minimum Value**: Enter a numeric constant or a tag value into this field to define the minimum value used to calculate the height (if vertical) or width (if horizontal) of the bars
- **Maximum Value**: Enter a numeric constant or a tag value into this field to define the maximum value used to calculate the height (if vertical) or width (if horizontal) of the bars

  If no value is specified for this field, the application opens a dialog for selecting or creating a tag



"WinStudio" also allows to enter constants in tag/numeric value fields. Constant values (defined by the # character) are equivalent to numeric values. The text constant can be used as description field as well as to specify generic values.

For example: `#Name:100`.

Where the value ("100") following the colon (:) is the generic value and "Name" is only a mnemonic for the constant and is not added to the database.

- **Foreground color:** To specify a filling color for the bar, click on the combo box
- **Direction**: Click the **Vertical** or **Horizontal** radio button to specify the direction of the bar graph
- **Orientation**: Click the **Up**, **Center**, or **Down** radio button to specify the orientation of the maximum and minimum values when drawing the bar

Text data link animation

Symbol: 

Click on **Text Data Link** in the **WinStudio dynamic function** toolbar to add the properties for input or output texts for the animation to a selected text object on page 192.

The dynamic function "Text Data Link" enables representation and change of tag values via the user interface during runtime.

This animation can only be applied to text objects that include one or more **#** characters.

Each # represents an input or output character. # characters can be combined with regular text in the same text object, for example, `MyLabel#####` or `$###.##`.

It is important to remember that the runtime project always displays the highest digits of a numeric value, regardless of the number or placement of # characters in the text. That means, if not sufficient # characters are available to display the value, then it is transformed depending on the format of the value (as set by the **Fmt** option described below):

- In decimal format, the number of decimal places is determined by the position of the decimal separator in the ### text. However, if not sufficient # characters are provided to the left of the decimal separator to display the whole value, the integer overwrites the decimal value

  For example, if a value of 112.64 is to be displayed in #.##, only 112 is displayed
- In hex and binary formats, if more # characters are provided than are required to display the value for Hex and Binary formats, the runtime project fills these in with leading zeroes. If less characters than required are available, the value simply is shortened
- In Auto format, the runtime project ignores the number of # characters and displays the entire numeric or string value

  Decimal values are displayed with all pre-decimal and decimal places, regardless of the position of the decimal separator in the ### text. Exceptionally large values or long strings may disrupt the layout of the screens.

Double-click on the object to open the "Object Properties" dialog. Use this dialog to specify the following parameters:

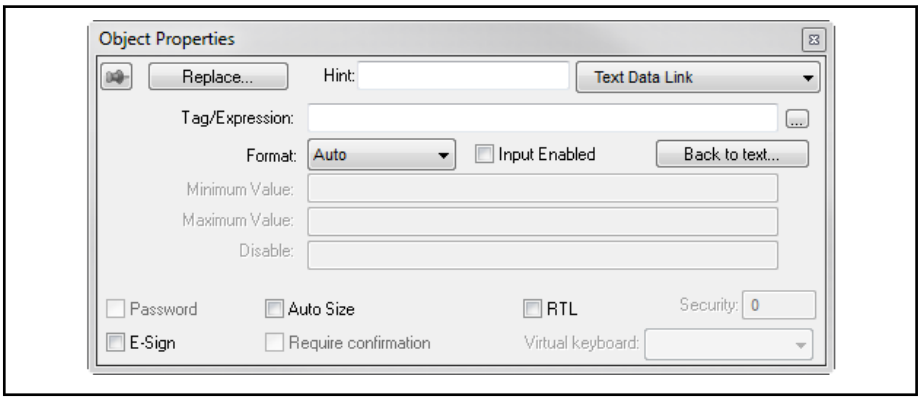


Fig. 8-92: Object Properties: Text Data Link

- **Tag/expression**

  Enter the following into the text field:

  – Name of a tag whose value is to be displayed or entered
  – Expression whose values are to be displayed

  Open the "Object finder" with the ... button to select an existing tag or expression.



If the configured tag/expression is invalid, the placeholder characters (###) are displayed during runtime.

- **Format**: Selection of how a numeric value of the specified tag or expression is to be formatted and displayed on the screen. Available options include **Decimal**, **Hexadecimal**, **Binary** and **Auto**. If **Auto** is selected, the value is formatted according to the virtual table created by the SetDecimalPoints on page 767 function. The format settings do not change the storage format of the displayed tag or expression

  Example: **Tag/Expression** is set to a tag of Integer on page 111 type, **Input Enabled** is checked and **Fmt** is set to **Hexadecimal**. A new value can be entered in hexadecimal format but it is saved in the project database as Integer
- **Input Enabled**: Click (enable) this option to allow user input for the specified tag. Disable (deactivate) this option if the value is only to be displayed at runtime without changing it
- **Back to text**: Click to return to the object properties of the underlying text object
- **Minimum Value**: Enter a minimum value for the tag associated with this text object. It is not possible to enter lower values
- **Maximum Value**: Enter a maximum value for the tag associated with this text object. It is not possible to enter higher values
- **Password**: Enable (activate) this option if the input field is used for a password input, for example, and all characters are to be replaced by a "*"
- **Require confirmation**: Enable (activate) this option to require users to confirm any new values set during runtime
- **Auto Size**: Enable (activate) this option to automatically resize the text object to fit the text length. This option is not available if **Input Enabled** is enabled (see above)
- **E-Sign**: If this option is enabled, a prompt is displayed to enter the electronic signature before changing the tag value
- **Virtual keyboard**: For this object, a virtual keyboard is used to enter values. Before configuration of the virtual keyboard for this interface, the option "virtual keyboard" in the "Viewer" tab in the properties of the "WinStudio" folder is to be activated
- **Disable**: Enter a value above zero in this field to deactivate the input function
- **Security** Enter a value into this field to specify the security level that is required to edit a value at runtime. (also refer to Security on page 461)

Color Animation

Icon: 

In the **WinStudio Dynamic Functions** toolbar, click on **Color change** to add the animation to a selected object.

The color animation enables modification of the color of an object during runtime based on the value of a tag or expression.

Double-click on the object to open the Object Properties dialog.

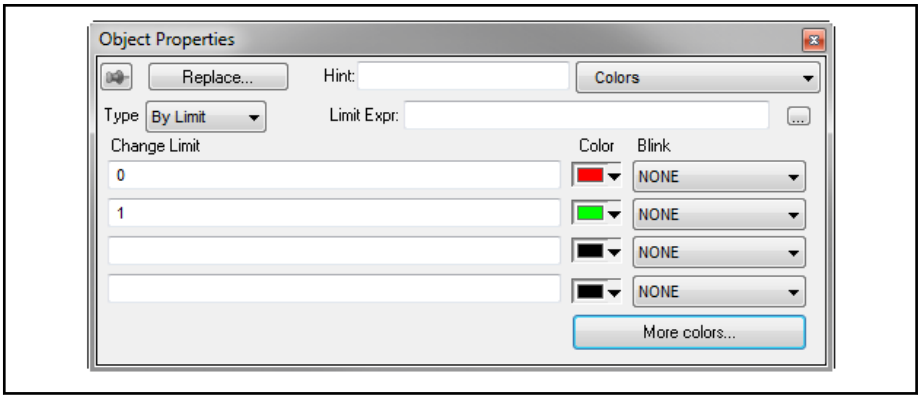


Fig. 8-93: Object Properties: Colors

Use this dialog to specify the following parameters:

- **Type**: Specifies the mode of the color change:
  - **By Limit**: If this type is selected, up to four limit values (Change Limit) and a color for each limit value can be defined. If the value of the tag or expression configured in the Tag/Expression field reaches the limit value, the color associated with the respective limit value is applied to the object
  - **By Color**: If this type is selected, any color code to be displayed is provided to a tag in the field Tag/Expression. Using this code, any color supported by the device can be applied to the object



The function RGB Color on page 763 can be configured in the Tag/Expression field if the entry "By Color" is selected under **Type**. This enables configuration of colors by its RGB codes. Refer to Color Interface on page 277 for a table with the codes for the most commonly used colors.

- **Tag/Expression**: Enter the name of a tag or expression to be monitored. If the "By Limit" type is selected, "WinStudio" compares the result of the tag/expression with the defined limit values to determine the correct color for the selected object. If "By Color" is selected, the value of the tag parameterized here defines the color of the object (color code)
- **Change Limit**: Enter a limit value (a numeric constant or tag) for the color change. The numbers have to be configured in ascending order (from top to bottom). Click on "More colors" to configure up to 16 different limit values for the color animation

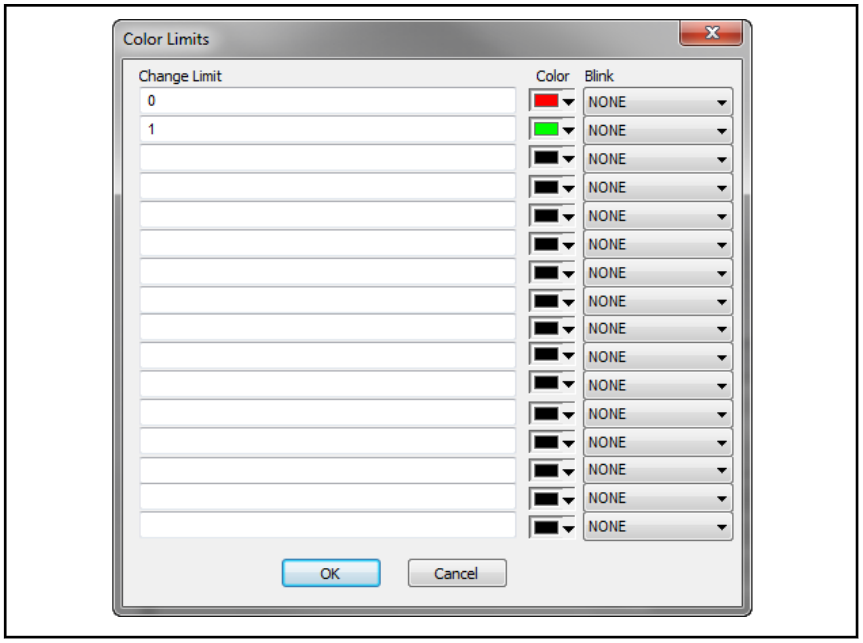


Fig. 8-94: Color Limits dialog

- **Color**: Click on the button to assign a color to each color change limit value
- **Blink**: Select whether the color is to blink on reaching the limit value and how fast

The following fields are automatically disabled (grayed out) if "By Color" is selected under **Type**: **Change Limit**, **Color** and **Blink**.

Position Animation

Symbol: 

The position animation enables moving an object horizontally and/or vertically at runtime.

In the **WinStudio Dynamic Functions** toolbar, click on **Position** to add the animation to a selected object.

Double-click on the object to open the "Object Properties" dialog.

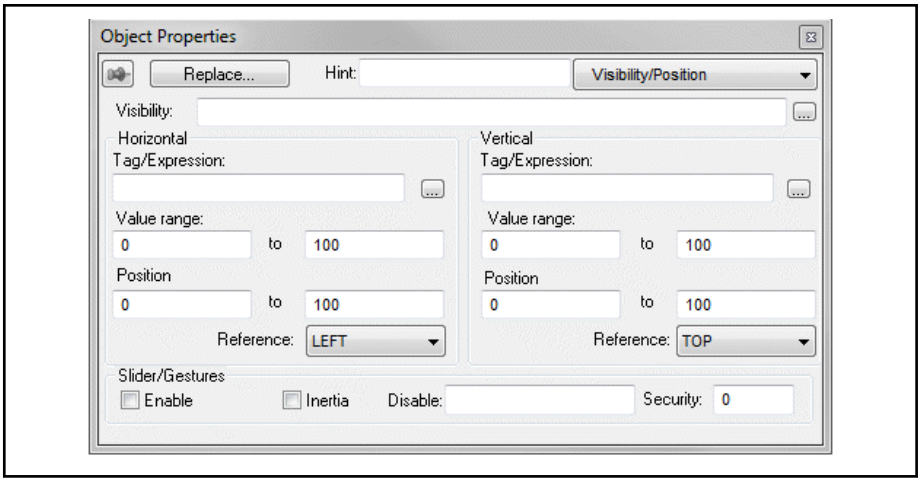


Fig. 8-95: Object Properties: Position.

Use the dialog to configure the following properties:

- **Visibility**: Configure a tag/expression in this field to define when the object is shown. If the field is blank or the tag/expression is TRUE (i.e. all values different from zero), the object is shown. If the value is FALSE (i.e. 0), the object is not shown. Thus, the field can be left blank to always show the object

  Hidden objects cannot be clicked/tapped and therefore cannot execute any command animations assigned to them.

  For some types of screen objects (i.e. all shapes, standard buttons and graphics links), the **Visibility** does not only control the visibility but also the opacity of the object and the value (from 0 to 1) of the Tag/Expression defines the opacity percentage. A value of 0.8 would, for example, give the object an opacity of 80%. This value can be changed during runtime. It can be used to display and hide objects. Please note that for this function to be available **Enable advanced graphics functions** must be selected in the project settings.

  For further information, refer to Project settings: Viewer, on page 61.

- **Horizontal** and **Vertical**: Configure these settings to determine how the object moves on the screen:

  Tag/expression fields: Configure a tag/expression which determines the position of the object during runtime. If the value changes, the object is moved on the screen If a tag or an expression can be configured depends on whether the **Slider/Gestures** option (see below) is selected:

  – If the **Slider/Gestures** option is not enabled, configure either a tag or an expression in this field.

  – If the **Slider/Gestures** option is enabled, only configure a project tag (Integer or Real type) in this field. If the end user manually moves the object, the new value is written on the tag again.

  For the horizontal position, the value increases as the object moves to the right and it decreases as the object moves to the left. For the vertical position, the value increases as the object moves to the bottom and it decreases as the object moves to the top.

- **Value range** fields: Enter the minimum and maximum values for the tag/expression.

  If the actual value goes outside of its range, then the value is ignored and the limit is used instead.

- **Position** fields: Enter values to specify how far (in pixels) the object can move from its starting position. The starting position is "0.0". Values greater than 0 allow the object to move right and down, and values less than 0 allow the object to move left and up.

  During runtime, the object's position is proportional to the tag/expression value within its range.

  Example:

  If **Position** is `0` to `100` and **Range** is `0` to `10`, each increment in the value moves the object by 10 pixels. This applies to **Horizontal** and **Vertical**.

- **Reference** drop-down lists: Select a reference point on the object. The following table illustrates how the selections for **Horizontal** and **Vertical** work in combination:

	LEFT	CENTER	RIGHT
TOP			
CENTER			
BOTTOM			

The reference point is only important if the Resize animation on page 257 has been added to the same object. The position of the object is always based on this reference point, regardless of the size or shape of the object.

- **Slider/Gestures**: If multi-touch gestures are activated, the end user can move this object during runtime with one or two fingers. The changes in position are then written on the project tags which are configured in the **Tag/Expression** fields above.
  - **Enable** option: Select this option to enable gestures for this object.

    Please note that multi-touch must also be activated for the project and the screen.
  - **Inertia** option: Select this option to enable inertia for this object so that it naturally slows down instead of stopping abruptly when the end user releases it.
  - **Disable** field: Configure a tag/expression If the value is TRUE (i.e. not 0), the gestures for this object are deactivated.
  - **Security** field: Enter the minimum Security level on page 461 an end user needs to be able to use gestures for this object.

For further information, refer to Using multi-touch in your project on page 285.

Resize Animation

Symbol: 

The "Resize" animation enables increasing or decreasing the size of an object during runtime.

In the **WinStudio Dynamic Functions** toolbar, click on **Resize** to add the animation to a selected object.

Double-click on the object to open the "Object Properties" dialog.

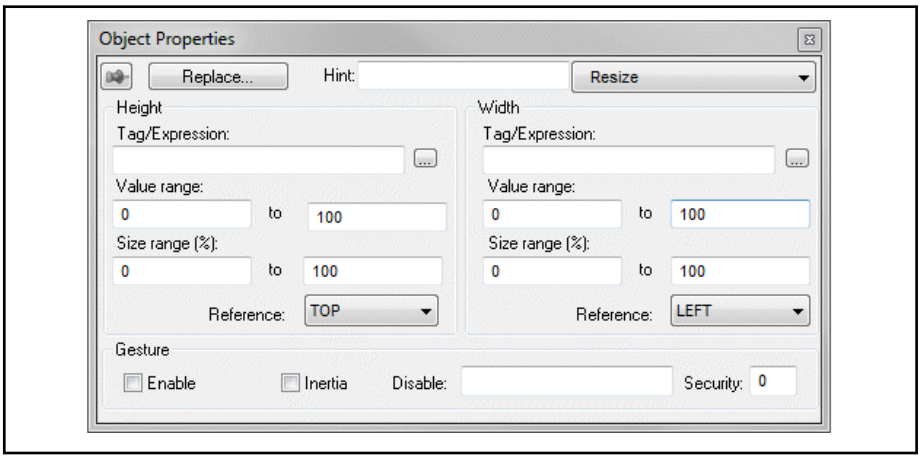
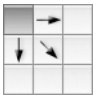
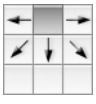
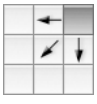


Fig. 8-96: Object Properties: Size

Use the dialog to configure the following properties:

- **Height** and **Width**: Configure these settings to determine how the object moves on the screen:
  - **Tag/expression** fields: Configure a tag/expression which determines the size of the object during runtime. If the value changes, the size of the object is changed on the screen. If a tag or an expression can be configured depends on whether the **Gestures** option (see below) is selected:
    - If the **Gestures** option is not enabled, configure either a tag or an expression in this field.
    - If the **Gestures** option is enabled, only configure a project tag (Integer or Real type) in this field. If the end user manually changes the object, the new value is written on the tag again.
  - **Value range**: Enter the minimum and maximum values for the specified tag(s). If the actual value goes outside of its range, then the value is ignored and the limit is used instead.
  - **Size range (%)**: Enter the minimum and maximum values for the size of the object. The minimum value can be as low as 0% (making the object effectively invisible), and the maximum value can be as high as required. The original size of the object is 100% if it is drawn in the screen editor. Thus, 200% is twice its original size, etc.

    During runtime, the size of the object is proportional to the tag value within its range. For example, if the **Size (%)** is "0" to "100" and the **Range** is "0" to "10", then each increment in the value increases the object size by 10%. This applies for both height and width
  - **Reference**: Select a reference point to determine the directions in which the object changes its size. The following table illustrates how the selections for height and width work in combination:

	LEFT	CENTER	RIGHT
TOP			

CENTER			
BOTTOM			

- **Gestures**: If multi-touch gestures are enabled, the end user can "pinch" and "spread" this object during runtime with two fingers. The changes in size are then written on the project tags which are configured in the **Tag/ Expression** fields above.
  - **Enable** option: Select this option to enable gestures for this object.

    Please note that multi-touch must also be activated for the project and the screen.
  - **Inertia** option: Select this option to enable inertia for this object so that it naturally slows down instead of stopping abruptly when the end user releases it.
  - **Disable** field: Configure a tag/expression If the value is TRUE (i.e. not 0), the gestures for this object are deactivated.
  - **Security** field: Enter the minimum Security level on page 461 an end user needs to be able to use gestures for this object.

For further information, refer to Using multi-touch in your project on page 285.

Rotation animation

Symbol: 

In the **WinStudio dynamic functions** toolbar, click on **Rotation**, to add the animation to an object Line on page 181, Open polygon on page 182, Closed polygon on page 183 or Bitmap on page 188 or Linked picture on page 242.



The rotation animation does not work for Group of objects on page 270. If the animation is added to an object and the object is then grouped with others, it is disabled.

In addition, the rotation animation cannot be added to bitmap objects and linked images in projects which run on Windows Embedded devices.

For further information, please refer to Target platforms and product types on page 86.

Double-click on the object to open the "Object properties" dialog.

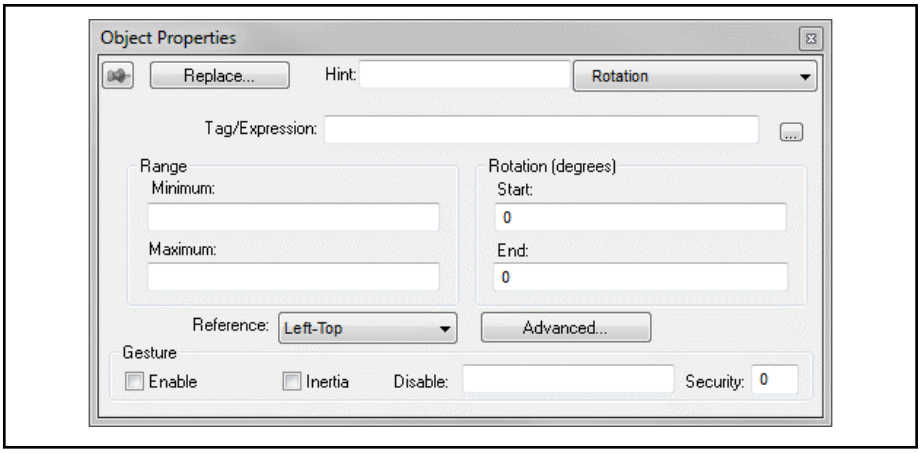


Fig. 8-97: Object Properties: Rotation animation

Use this dialog to specify the following parameters.

- **Tag/Expression** field: Configure a tag/expression which determines the angle of the object at runtime. If the value changes, the object is rotated on the screen. Whether a tag or an expression can be specified depends on whether the “Gesture” option (see in the following) is selected:
  - If the **Gesture** option is not enabled, specify either a tag or an expression in this field.
  - If the **Gesture** option is enabled, only specify a project tag (integer or real type) in this field. If the end user manually rotates the object, the new value is written on the tag again.
- "Value range": Enter the **Minimum** and **Maximum** values allowed for **Tag/Expression**. Values less than the minimum and greater than the maximum are ignored.
- "Rotation (degree)": Enter the **Start** and **End** positions (in degrees) of the object. The actual rotation is proportional to the value of **Tag/Expression** within the "Range". An object can rotate up to 360 degrees and it rotates clockwise by default.

For example, a rotation animation has the following settings: **Minimum** is 0, **Maximum** is 100, **Start** is 0 und **End** is 180. If the current value of **Tag/Expression** is 50 (i.e. halfway between “Minimum” and “Maximum”), the actual rotation of the object is 90 degrees (i.e. halfway between “Start” and “End”). A value of 25 is equal to 45 degrees, a value of 75 is equal to 135 degrees, etc.

- **Reference**: Select one of the following points as pivot point to rotate the object:
  - **Left-top**: Upper-left corner of the object
  - **Left-bottom**: Lower-left corner of the object
  - **Center**: Center of the object
  - **Right-top**: Upper-right corner of the object
  - **Right-bottom**: Lower-right corner of the object

  The pivot point can be exactly positioned via offset (“Advanced” button) as described in the following.
- **Advanced**: Click on the button to open the "Advanced" dialog to configure the following settings:

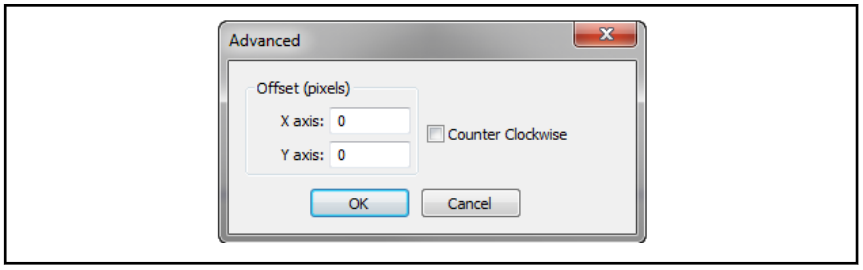


Fig. 8-98: Object properties: Rotation animation – "Advanced" dialog

- "Offset (pixels)": Enter the number of pixels by which the **Reference** (i.e. pivot point) is to offset on the **X-axis** and/or **Y-axis**.
- **Counterclockwise**: Select this option to rotate the object counterclockwise.

- **Gesture**: If multi-touch gestures are enabled, the end user can "rotate" this object with two fingers at runtime. The angle changes are again written on the project tag configured in the above-mentioned “Tag/ Expression” field.
  - **Enable** option: Select this option to enable gestures for this object.

    Please note that multi-touch also has to be activated for the project and the screen.
  - **Inertia** option: Select this option to enable inertia for this object so that it naturally slows down instead of stopping abruptly when the end user releases it.
  - **Disable** field: Configure a tag/expression. If the value is TRUE (i.e. not 0), the gestures for this object are disabled.
  - **Security** field: Enter the minimum Security level on page 461 an end user needs to be able to use gestures for this object.

For further information, refer to Using multi-touch in your project on page 285.

8.2.6 Using user-defined property values when opening image objects

General information

When opening image objects, user-defined values, variables or expressions can be dynamically assigned to image object properties or animation properties. This function is based on the use of user-defined properties that can be configured as aliases to display animations and image objects.

The `$Open()` function is used for dynamic assignment.

Consequently, an image object can be used to represent different values, depending on the context it has been opened in.

Syntax of a user-defined property

`#<CustomPropertyName>:<CustomPropertyValue>`

`<CustomProperty-Name>`	**Identifier** (alias) of the user-defined property
`<CustomProperty-Value>`	**Actual value** of the user-defined property Valid: • literal values (numerical or alphanumerical) • Variables • Expressions in brackets [...] The value can also be omitted so that no default value is specified for the user-defined property. In this case, the value can be assigned dynamically when opening the image object using the `$Open()` function.

Only `<CustomPropertyValue>` is used during runtime and the remaining text of the syntax mentioned before is ignored.

Examples:

User-defined property (complete syntax)	**User-defined property name** (alias used as identifier)	**Property value** (used at runtime)
`#MyNumValue:10`	`MyNumValue`	`10`
`#MyTextValue:"ABC"`	`MyTextValue`	`"ABC"`
`#MyTag:Second`	`MyTag`	`Second`
`#MyExpression:(Minute*10)`	`MyExpression`	`(Minute*10)`

In most cases, the value of the user-defined property is completely replaced by the value that is transmitted by the embedded function `$Open()`.

Dynamic property value assignment

In the following example, the dynamic assignment of a property value via the "`$Open()`" function is illustrated:

In the `MyScreen` *object, an object property with the syntax* `#MyCustomProperty:` *is configured but does not contain any property value yet.*

1. The property value can be dynamically assigned via the following expression:

   ```
   $Open("MyScreen",-1,-1,-1,-1,0,0,"#MyCustomProperty:Second")
   ```

   The expression opens the screen object `MyScreen` and the alias `#MyCustomProperty:` is replaced by the `Second` tag during runtime.

More examples:

User-defined property configured in objects or animations	**User-defined property** that is transmitted by the `$Open()` function	**Proprety value** executed during runtime
`#MyCustomProperty:Minute`	`#MyCustomProperty:Second`	`Second`
`#MyCustomProperty:Minute`	`#MyCustomProperty:Second- >Quality`	`Second->Quality`

#MyCustomProperty:Minute	#MyCustomProperty:Mytag[1].MyMember	Mytag[1].MyMember
#MyCustomProperty:Minute	#MyCustomProperty:Mytag[1].MyMember->Quality	Mytag[1].MyMember->Quality

Partial assignment of user-defined properties using tag arays

Assuming that the following properties are configured for different objects in an image object:

- #MyCustomProperty:Minute
- #MyCustomProperty:Minute->Min
- #MyCustomProperty:Minute->Max
- #MyCustomProperty:Minute->Unit

The user-defined property #MyCustomProperty is linked to different values (Minute, Minute->Min, Minute->Max and Minute->Unit). This is valid if all properties are assigned to different arrays of the same tag. Use the $Open() function to dynamically replace the Minute tag by another tag.

Example:

```
$Open("MyScreen",-1,-1,-1,-1,0,0,"#MyCustomProperty:Second")
```

The following table illustrates the property value change after the function has been called:

User-defined property configured in objects or animations	**User-defined property** that is transmitted by the $Open() function	**Proprety value** executed during runtime
#MyCustomProperty:Minute	#MyCustomProperty:Second	Second
#MyCustomProperty:Minute->Min	#MyCustomProperty:Second	Second->Min
#MyCustomProperty:Minute->Max	#MyCustomProperty:Second	Second->Max
#MyCustomProperty:Minute->Unit	#MyCustomProperty:Second	Second->Unit

Partial assignment of user-defined properties by means of class tags

Assuming that the following properties are configured for different objects in an image object:

- #MyCustomProperty:MyTagA.MemberX
- #MyCustomProperty:MyTagA.MemberY
- #MyCustomProperty:MyTagA.MemberZ

The user-defined property #MyCustomProperty is linked to different values (MyTagA.MemberX, MyTagA.MemberY and MyTagA.MemberZ). This is valid if all properties values are assigned to different members of the same class tag. Use the $Open() function to dynamically replace the "MyTagA" tag by another tag.

Example:

```
$Open("MyScreen",-1,-1,-1,-1,0,0,"#MyCustomProperty:MyTagB")
```

The following table illustrates the property value change after the function has been called:

User-defined property configured in objects or animations	**User-defined property** that is transmitted by the `$Open()` function	**Proprety value** executed during runtime
`#MyCustomProperty:MyTagA.MemberX`	`#MyCustomProperty:MyTagB`	`MyTagB.MemberX`
`#MyCustomProperty:MyTagA.MemberY`	`#MyCustomProperty:MyTagB`	`MyTagB.MemberY`
`#MyCustomProperty:MyTagA.MemberZ`	`#MyCustomProperty:MyTagB`	`MyTagB.MemberZ`

☞ The partial assignment is only valid if the original class tag `MyTagA` and the new class tag `MyTagB` correspond to the same class type.

Partial assignment of user-defined properties by means of array tags

Assuming that the following properties are configured for different objects in an image object:

- `#MyCustomProperty:MyArray[1]`
- `#MyCustomProperty:MyArray[2]`
- `#MyCustomProperty:MyArray[3]`

The user-defined property `#MyCustomProperty` is linked to different values (`MyArray[1]`, `MyArray[2]` und `MyArray[3]`). This is valid if all properties values are assigned to different array positions of the same tag. Use the $Open() function to dynamically replace the "`MyArray`" tag by another tag by retaining the array positions.

Example:

```
$Open("MyScreen",-1,-1,-1,-1,0,0,"#MyCustomProperty:NewArray")
```

The following table illustrates the property value change after the function has been called:

User-defined property configured in objects or animations	**User-defined property** that is transmitted by the `$Open()` function	**Proprety value** executed during runtime
`#MyCustomProperty:MyArray[1]`	`#MyCustomProperty:NewArray`	`NewArray[1]`
`#MyCustomProperty:MyArray[2]`	`#MyCustomProperty:NewArray`	`NewArray[2]`
`#MyCustomProperty:MyArray[3]`	`#MyCustomProperty:NewArray`	`NewArray[3]`

☞ The partial assignment is only valid if the original class tag `MyArray` and the new tag `NewArray` correspond to the "Array" tag type.

8.3 Formatting screen objects

8.3.1 Formatting screen objects, general information

The "WinStudio Alignment Functions" and "WinStudio Object Properties" toolbars offer numerous functions to align, group and design screen objects.



Fig. 8-99: "Alignment Functions" toolbar



Fig. 8-100: "Object Properties" toolbar

Some functions in the toolbars are only available if one or multiple objects have been selected in the WinStudio screen.

Move the cursor over an active "tool" to display the function in plain text.

The tools are organized into the following groups:

- **Arrange**: Arranging objects in a WinStudio screen including
  - Move to foreground and Move to background on page 269
  - To the foreground and To the background, by one level on page 269
  - Group and ungroup objects on page 270
  - Align, Center and Distribute on page 271
  - Rotating and mirroring on page 273
- Adjusting position on page 274
- **Size**: Same width / same height for selected objects on page 275 of a screen object in a WinStudio screen
- **Style**: Changing the color of screen objects
  - Fill color on page 276 of a screen object
  - Line color on page 276 of a screen object
- **Edit the color of a component**: Color Interface on page 277
- **Fonts**: Changing Fonts on page 280 of a screen object

8.3.2 Changing the properties of multiple screen objects (formatting)

This task describes how to select two or more screen objects and how to then change the properties the selected objects have in common.

Before starting this task, a project image must be opened in the image editor.

Which properties can be changed depends on whether several objects of the same type or different object types are selected. If the objects are of the same type, you can change the properties applicable to this type. If, for example, several button objects are selected, the properties applicable to button objects can be changed.

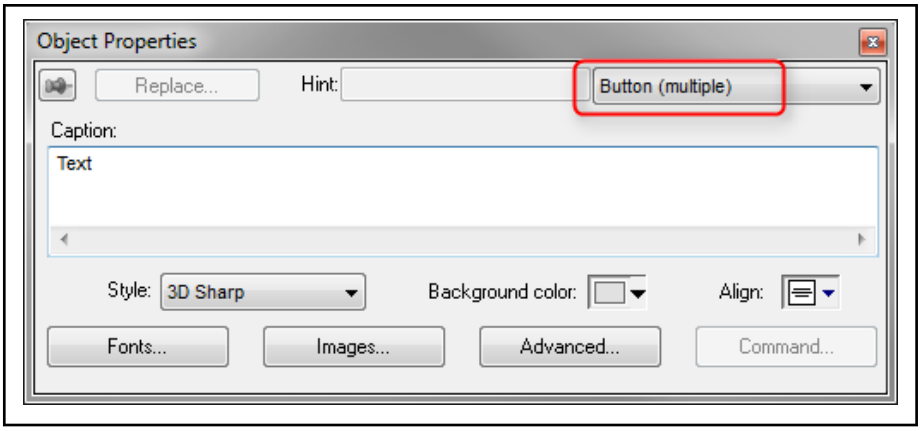


Fig. 8-101: Object properties of multiple selected button objects

For more information on the properties of a specific object type, please consult the documentation for this object



This method can only be used to change the properties of shapes and active objects. The method cannot be used to change the properties of file objects, animations, library elements or objects in a group.

However, if multiple objects of different types are selected, the properties shared by all objects can be changed. This not only includes cosmetic properties such as Border and Background but also functional properties such as Disable, Security, Enable translation and E-Sign. (Some properties may not apply to all objects. Button objects, for example, do not have a border and a rectangle object does not have a security.)

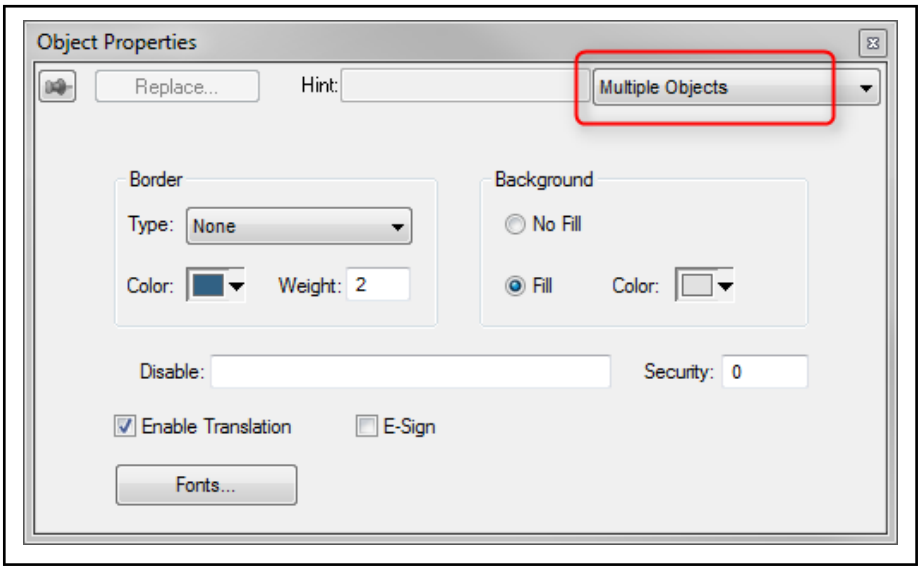


Fig. 8-102: Object properties for multiple objects of different types

In both cases, the dialog box displays the current values of the properties of the most recently selected object.

Only if the value of a property is actually changed, will the change be applied to the selected objects. All other properties remain unchanged independent from the values displayed in the dialog box.

Changing the properties of multiple screen objects:

1. Do one of the following in the screen editor:
   - Either press and hold down <Shift> or <Ctrl> on the keyboard and click on each object that is to be changed; or
   - Use the cursor to draw a selection box around all objects that are to be changed.

   The objects are selected.
2. Do one of the following:

   Right-click on the selected objects and click on **Properties** in the context menu; or

   Press <Alt>+<Enter> on the keyboard.



The Object properties dialog box cannot be opened by a double-click as usual since a double-click cancels the selection.

   The Object properties dialog box is displayed for the selected objects.
3. Change the property values as desired and then close the dialog box.

The changes are applied to all selected objects.

8.3.3 Determining the tab order of screen objects

Determine the tab order of screen objects to create a screen which can be used more easily via a physical keyboard/keypad instead of a mouse or a touchscreen.

If Tab is pressed on the keyboard/keypad during runtime, the focus moves to the next object or field on the screen. (Also referred to as: The object becomes active.) Using the focus the object or field can be controlled via the keyboard/keypad alone.

Example:

- If the focus is on a button, press the Enter key to click on this button;
- if the focus is on a text field, enter a value in this field and then press **Enter** to confirm the value; and
- If the focus is on a list or a menu, use the arrow keys to navigate through the menu and then confirm your selection using the **Enter key**.

If **Tab** is pressed repeatedly, the focus switches through all objects on a screen according to the tab order of the screen. By default, the tab order corresponds to the layer order. It starts with the layer which is furthest back (ID: 0) and ends with the one furthest at the front (ID: n). It is, however, possible to draw objects everywhere, to re-arrange them and to adjust their layers while developing the screen. This is why it may seem as if the default tab order of a finished screen jumps back and forth at random instead of moving from left to right or from top to bottom as the user would expect.

Determine the tab order to ensure it moves through the screen exactly as desired independent from the layer order.

Determining the tab order:

1. Open an image for editing.
2. Right-click on the screen background and click on **Tab order** in the context menu.

If **Tab order** is not available in the context menu, the right click may have been on a screen object or a screen object has already been selected. Please ensure that no screen object is selected and the right click is made on the screen background.

The screen opens the tab order mode and the number of each screen object is displayed in an orange box in the top right corner of the object.

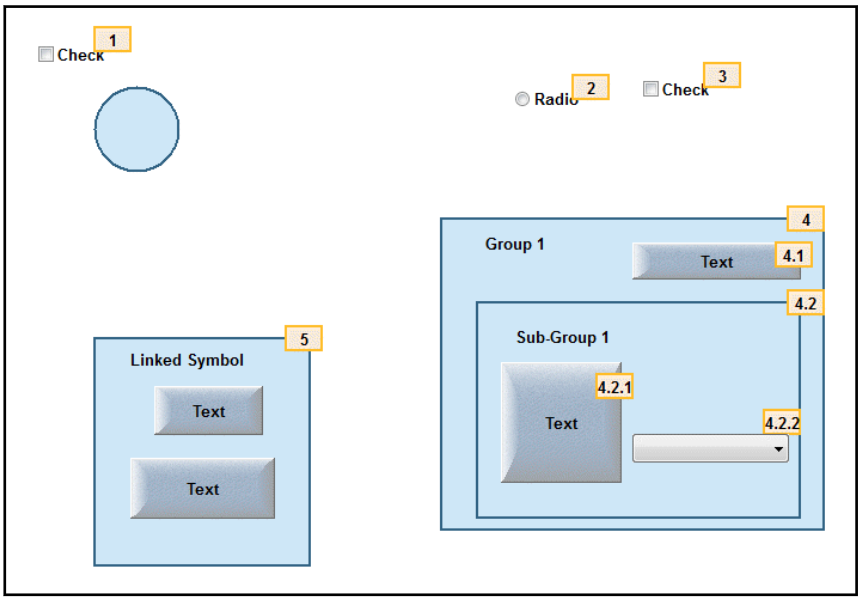


Fig. 8-103: Example for the tab order mode

A group of objects has an individual number for the entire group (e.g. 4) and the objects in the group have their own sub-number (e.g. 4.1 and 4.2). This is continued downwards as far as required to also represent nested groups. If the sub-numbers are not displayed, right-click again on the screen and click **Expand all groups** in the context menu.

3. To quickly set the tab order for all objects just click on the objects in the desired order. By clicking on the objects the numbers are updated.
4. Determining the number of a specific object:
   - 4.1 Double-click on the box at the object. The box becomes a text input field.
   - 4.2 Enter the number for this object: Make sure to enter all required sub-numbers.
   - 4.3 Confirm the change with the **enter key**.

The sub-numbers of objects within a linked symbol cannot be changed since it is only a copy of the master symbol which is used by the entire project. Instead, the master symbol needs to be changed. For further information, refer to Saving your own Project symbols on page 223.

5. After determining the tab order, right-click on the screen and click on **Tab order** in the context menu to exit the tab order mode.

8.3.4 "Move to foreground" and "Move to background"

"WinStudio" assigns a unique identification number (ID#) to every screen object. These ID#s always start at zero and range up to the total number of objects on the WinStudio screen.

Click on an object to display its ID number in the Status Bar on page 84:

"WinStudio" uses ID#s to determine whether an object is displayed in front of or behind another screen object. Objects with lower ID#s are displayed behind objects with higher ID#s.

With the following object layer tools, selected objects can be moved behind or in front of other screen objects.

These tools can be used only with a single selected object or a group on page 270 of objects. If several individual objects are selected and moved behind or in front of another object, the selected objects maintain their original display order.



In this chapter, "Object" refers to individual screen objects as well as object groups but not to several objects which were selected but not grouped.

Move object to the background

In the **WinStudio Alignment Functions** toolbar, click on  , **Move object to the background** to move one or multiple selected objects behind all other objects in the screen. "WinStudio" assigns the object the lowest ID# and moves it behind all other objects on the screen.

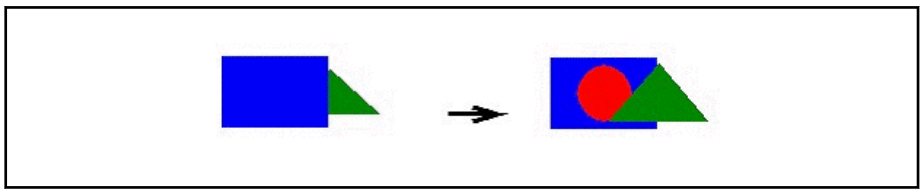


Fig. 8-104: Moving objects to the background

Alternatively, right-click on an object and select **Move to background** from the context menu of the object.

Move object to the foreground

Click on the  tool, **Move object to the foreground**, to move a selected object or objects in front of all other objects on the screen. "WinStudio" assigns the object the highest ID# and moves it in front of all other objects on the screen.

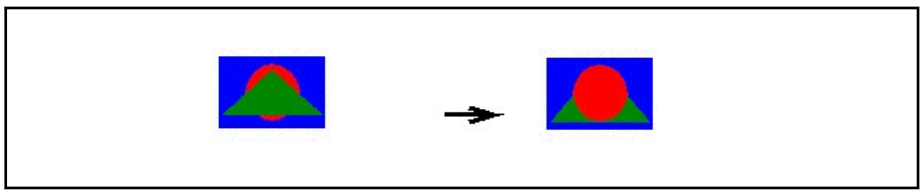


Fig. 8-105: Moving objects to the front

Alternatively, right-click on an object and select **Move to foreground** from the context menu.

8.3.5 "To the foreground" and "To the background", by one level

"WinStudio" assigns a unique identification number (ID#) to every screen object. These ID#s always start at zero and range up to the total number of objects on the screen. Click on an object to display its ID number in the Status Bar on page 84:

"WinStudio" uses ID#s to determine whether an object is displayed in front of or behind another object in the screen. Objects with lower ID#s are displayed behind objects with higher ID#s. With the following object layer tools, selected objects can be moved behind or in front of other screen objects.



These tools can only be used with a single selected object or group on page 270 of objects. These tools cannot be used with multiple objects selected. If several individual objects are selected and moved behind or in front of another object, the selected objects maintain their original display order.

Open the context menu of a selected object and click on **To the back** to move the selected object/group of objects by one level behind the next object in the screen. "WinStudio" assigns the selected object the next available ID# less than the object behind which it was moved.

Open the context menu of a selected object and click on **To the front** to move the selected object/group of objects by one level in front of the next object in the screen. "WinStudio" assigns the selected object the first available ID# greater than the object in front of which it was moved.

8.3.6 "Group " and "ungroup objects"

Symbols of the "Alignment Functions" toolbar:

Use the following tools to group and ungroup two or more selected objects.



All objects with animations on page 246 and groups of symbol objects (which includes most symbols and library objects) have multiple object and dynamic properties.

The selection list in the "Object Properties" dialog, provides access to these various dialogs and properties.

Grouping objects

In the **WinStudio Alignment Functions** toolbar, click on  , **Group objects**, to combine several selected objects to a single object for easier handling. (Each part of the group can be accessed in the "Object Properties" dialog.)



Alternatively, right-click on an object and select **Group** from the context menu.

Ungrouping objects

In the **WinStudio Alignment Functions** toolbar, click on  , **Ungroup**, to divide a grouped object into its individual components.



Alternatively, right-click on an object and select **Ungroup** from the context menu.



A complex grouped object can consist of several sets of grouped objects ("subgroups"). Consequently, it may be necessary to ungroup all of the subgroups to completely ungroup a complex object.

Editing the object properties of a group

To edit the object properties of a group use the Object properties dialog box on page 178 as for an individual object. However, a group has more than one set of object properties like an object with added Animations on page 246. Use the list in the top right corner of the dialog box to select a set of object properties.

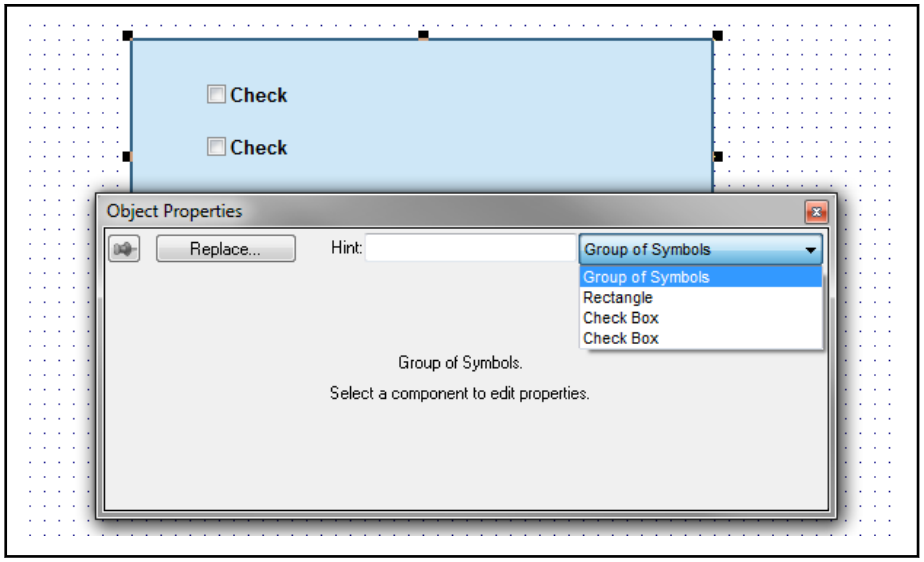


Fig. 8-106: Selecting a set of object properties in a group

When a group - also referred to as a symbol - has been created, a master of it can be saved in your Symbol library on page 222 and the group can also be used elsewhere in your project. Each copy is linked to the master. If the master is changed, all linked copies are also changed. For further information, refer to Saving your own Project symbols on page 223.

To cancel the grouping of objects, select the group and do one of the following:

- Right-click on the selected group and click on Cancel grouping in the context menu.

A complex group can have several sub-groups. This means that first the grouping of the group and then the grouping of the sub-group has to be cancelled.

8.3.7 "Align", "Center" and "Distribute"

If a series of objects (two or more) is selected, those objects can be aligned based on the location of the last object selected. The last object selected is assigned a special mark (filled check icon).

In all of the figures, the rectangle represents the last object selected.

Use the following tools from the **WinStudio Alignment Functions** toolbar to align a series of objects.

Examples are provided in the following figures:

Left-aligned

Click on , **Left-aligned**, to align all selected objects to the left edge of the last object selected.

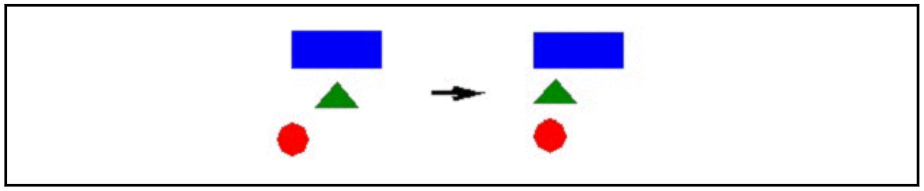


Fig. 8-107: Aligning objects left

Right-aligned

Click on , **Right-aligned**, to align all selected objects to the right edge of the last object selected.

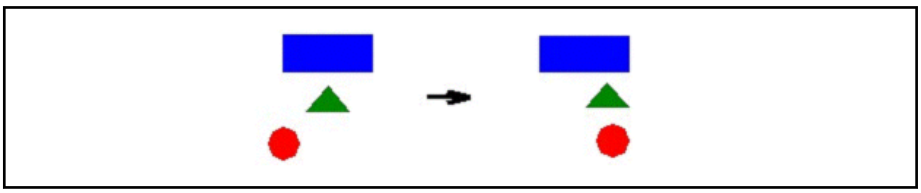


Fig. 8-108: Aligning objects right

Align top

Click on , **Align top**, to align all selected objects to the top edge of the last objected selected.

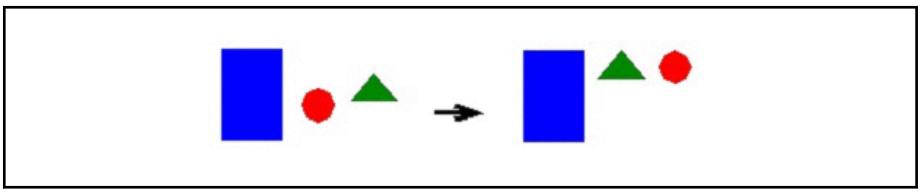


Fig. 8-109: Aligning object to the top

Align bottom

Click on , **Align bottom**, to align all selected objects to the bottom edge of the last object selected.

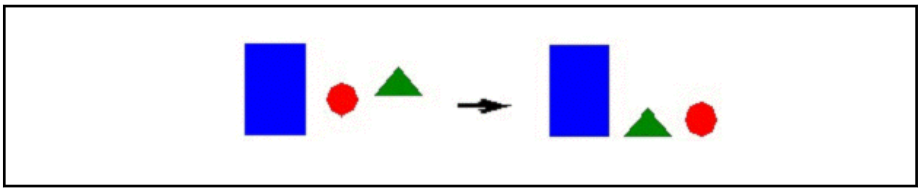


Fig. 8-110: Aligning object to the bottom

Center vertically

Click on , **Center vertically**, to align all selected objects to the vertical center of the last object selected.

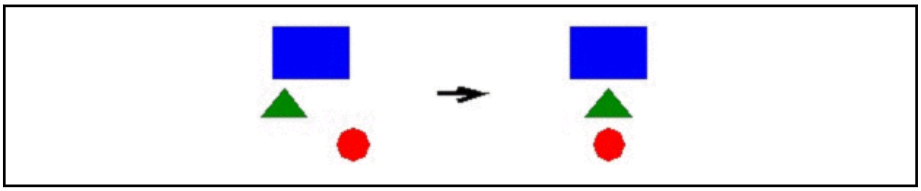


Fig. 8-111: Centering objects vertically

Centering horizontally

Click on , **Center horizontally** to align all selected objects to the horizontal center line of the last object selected.

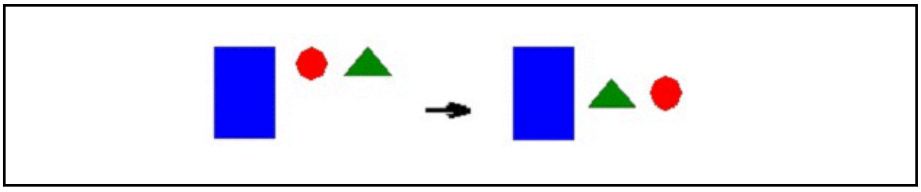


Fig. 8-112: Centering objects horizontally

Distributing horizontally

Click on , **Distributing horizontally** to put an equal amount of horizontal space between a series of objects (three or more).

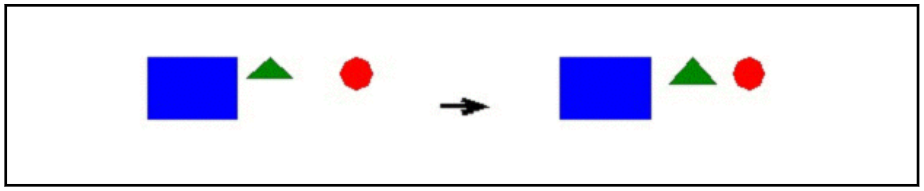


Fig. 8-113: Distributing objects horizontally

Distributing vertically

Click on , **Distributing vertically** to put an equal amount of vertical space between a series of objects (three or more).

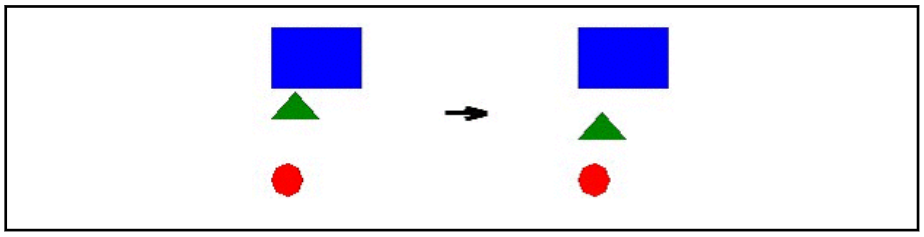


Fig. 8-114: Distributing objects vertically

The distribution tools may move the last object selected (with filled handles) by no more than a few pixels to equally distribute all of the objects.

8.3.8 "Rotating" and "mirroring"

"Rotating"

Use the following tools from the **WinStudio Alignment Functions** toolbar to rotate and mirror objects.

Rotate

Click on the tool , **Rotate**, to rotate the selected object by 90 degrees (a quarter turn) clockwise:

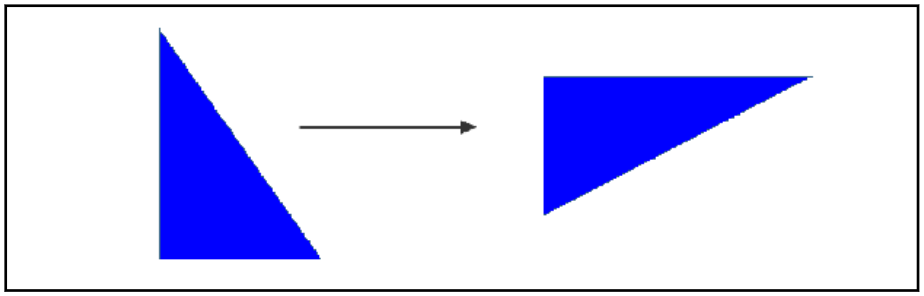


Fig. 8-115: Rotating objects

This tool can only be used with a single selected object or a group on page 270 object. This tool cannot be used with multiple objects selected.

"Mirroring vertically"

Click on the tool , **Mirror vertically**, to invert the selected object vertically. The object rotates around an imaginary vertical center line until it is a mirror image of the original object:

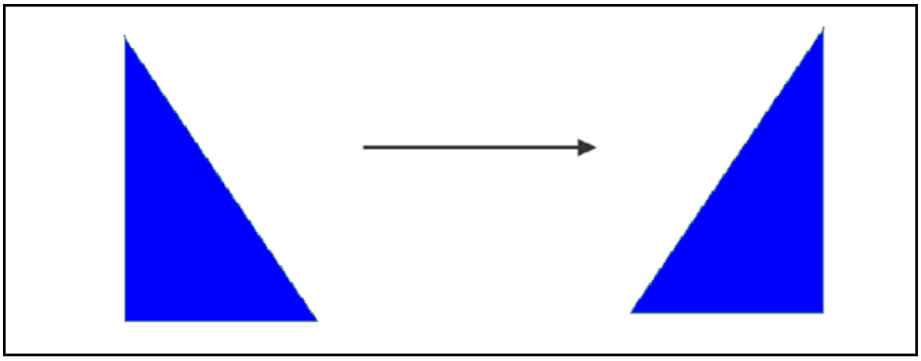


Fig. 8-116: Mirroring objects vertically

This tool can only be used with a single selected object or a group on page 270 object. This tool cannot be used with multiple objects selected.

"Mirroring horizontally"

Click on the tool , **Mirror horizontally**, to invert the selected object horizontally. The object rotates around an imaginary horizontal center line until it is a mirror image of the original object.

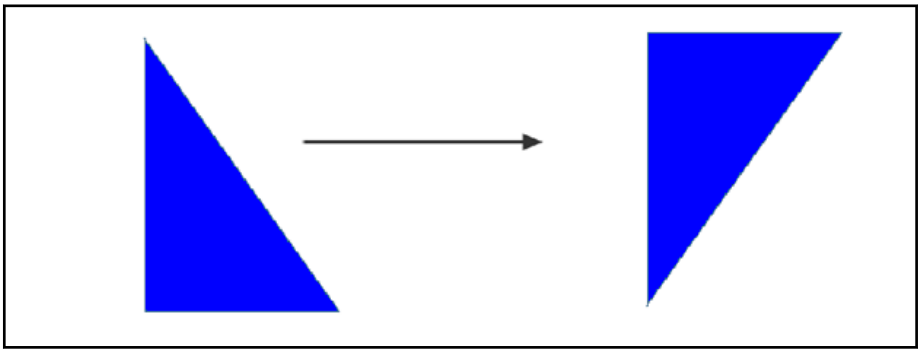


Fig. 8-117: Mirroring objects horizontally

This tool can only be used with a single selected object or a group on page 270 object. This tool cannot be used with multiple objects selected.

8.3.9 Adjusting position

Moving a screen object by selecting and dragging it can be imprecise, particularly if "Snap to grid" is activated. For this reason, positioning tools can be used to exactly adjust (readjust) the position of objects.

1. Click on a screen object to select it.
2. Keep the <SHIFT> key pressed and use the arrow keys to adjust (readjust) the horizontal and vertical position of the object. The step width is derived from the grid width of the grid lines.

Pixel-accurate positioning of screen objects: The position can also be changed by entering the x- and y-coordinates manually. Enable the toolbar WinStudio Alignment Functions Toolbar on page 41 and use the keyboard to enter a valid numerical value either into the field "X" or the field "Y". To apply the new values, confirm them with the Enter key. The position of the screen object is immediately updated.

Also refer to:

- Status Bar on page 84
- Grid Settings on page 181

8.3.10 "Same width" / "same height" for selected objects

Use the following functions in the "WinStudio Alignment Functions" toolbar to adjust the size of screen objects:

Same width for selected objects

Click on  , **Same width for selected objects**, to increase/decrease the width of the object selected first to the width of the secondly selected object.

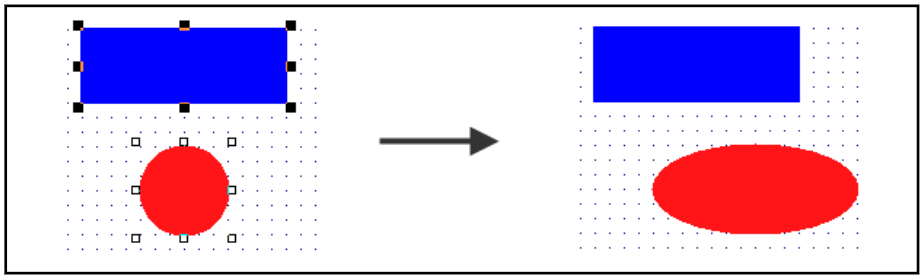


Fig. 8-118: Elongating objects to the same width

Same height for selected objects

Click on the tool  , **Same height for selected objects**, to increase/decrease the height of the object selected first to the height of the secondly selected object.

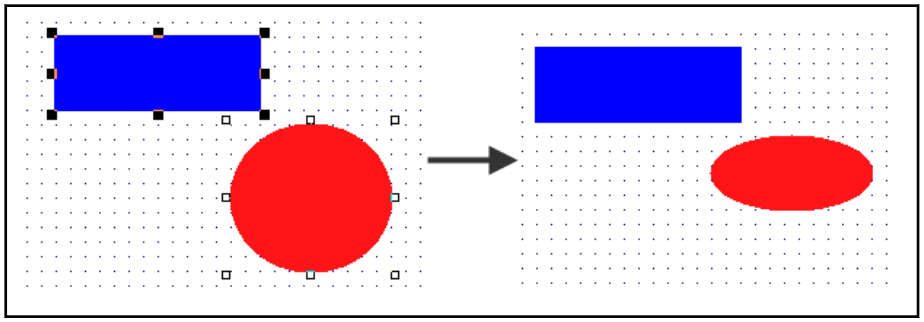


Fig. 8-119: Compressing objects to same height



By means of **Adjust width** and **Adjust height**, an ellipse can be turned into a circle or a rectangle into a square. However, only one object may be selected.

The mouse and arrow keys can also be used to resize objects. When selecting an object (or group of objects) with the cursor, handles are displayed at each corner and at centers of each edge. These handles can be used as follows:

- **To enlarge an object**, drag a handle into the respective direction. Dragging a side handle resizes the object in one direction only (height only or width only). Dragging a corner handle resizes the entire object (height and width). When dragging a corner handle, the proportion of sides remains unchanged by default. To freely resize the object, hold down the SHIFT key while dragging the handle.
- **To adjust the size of an** open polygon on page 182 **or** closed polygon on page 183, select and group on page 270 the polygon (Group context menu). The size of the polygon can then be adjusted similar to any object.

If the function to adjust the size is used for grouped objects (e.g. linked symbols, group of screen elements), all individual elements in the group are adjusted in the same direction and in the same proportion.

Pixel-accurate change in size: Pixel-accurate change of screen object size is possible. Show the WinStudio toolbar WinStudio Alignment Functions Toolbar on page 41. The input fields for width and height of the selected object are located there. The displayed value can be changed with the keyboard and applied with “Enter”.

Also refer to:

- Status Bar on page 84

8.3.11 Fill color

Click on the tool  , **Fill Color**, on the "WinStudio Object Properties" toolbar to specify a default fill color for the following objects:

- Closed polygon object on page 183
- Ellipse Object on page 187
- Rounded Rectangle Object on page 186
- Rectangle object on page 184

To save development time, several objects can be selected (of any type specified in the preceding list) to use **Fill Color** to specify a default fill color for all of them at once.

Also refer to:

- **Edit the color of a component**: Color Interface on page 277

8.3.12 Line color

Click on the tool  , **Line Color**, on the "WinStudio Object Properties" toolbar, to specify a line color for selected objects or to set a default color for new objects, including the following:

- Line Object on page 181
- Open polygon object on page 182
- Closed polygon object on page 183
- Rectangle object on page 184
- Rounded Rectangle Object on page 186
- Ellipse Object on page 187

Click the **Line Color** tool, to display the "Line Selection" dialog. Use this dialog to specify line styles and colors for the selected objects.

To save development time, several of the preceding objects can be selected simultaneously to use the **Line Color** tool to specify a line color for all of the objects at once.

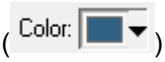
Also refer to:

- **Edit the color of a component**: Color Interface on page 277

8.3.13 Color Interface

The color interface can be used to edit the color of a component.

Color picker

1. Click on the symbol () in the "Properties" menu item of a graphic object
2. A popup box is displayed to select one of ten default colors or sixty co-ordinated design colors

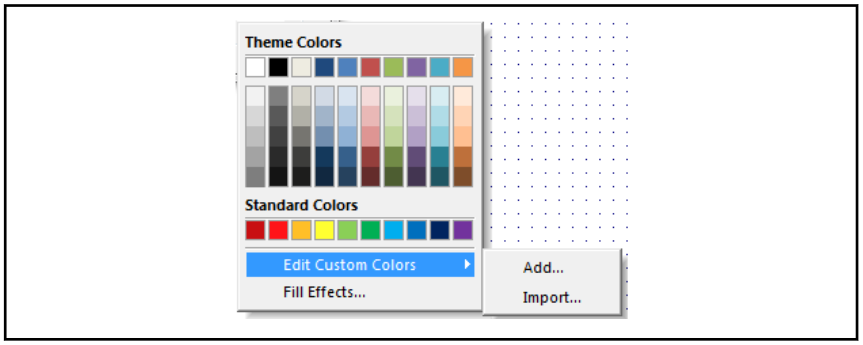


Fig. 8-120: Pop-up box with main colors

The selected color is applied to the edited component

The box enables transition to

- Editing user-defined colors: "Adding standard and user-defined colors" on page 277
- Editing user-defined colors: "Import" on page 278
- "Fill effects" on page 278

Adding standard and user-defined colors

1. Click on **Edit Custom Colors** to use a different color. The color can be selected on the **Standard** tab by clicking on the required color hexagon or by using the eyedropper (Select button)



The eyedropper enables applying any color from the IndraWorks environment.

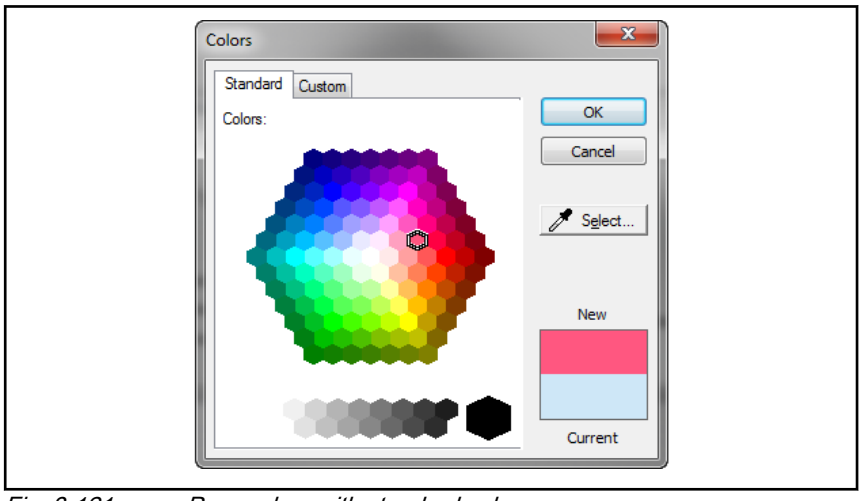


Fig. 8-121: Pop-up box with standard colors

2. Click on the **Custom** tab.

The custom color can be defined by specifying the HSL (hue, saturation, lightness) color code or the RGB (red, green, blue) color code (also refer to "RGB code table" on page 279)

As an alternative, practically any color can be selected by clicking with the mouse or using the eyedropper

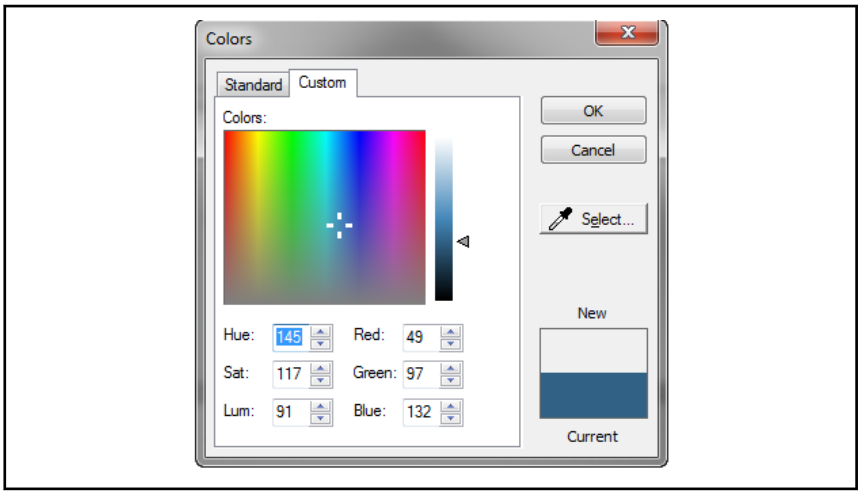


Fig. 8-122: Pop-up box to create a custom color

3. Click on **OK** to apply the selected color to the component that is being edited

Import This option enables import of a "user-defined color" text file.

Fill effects

1. Depending on the component that is edited, the **Fill Effects** option is available on the pop-up interface. Click this option to apply gradient colors with different styles and variants

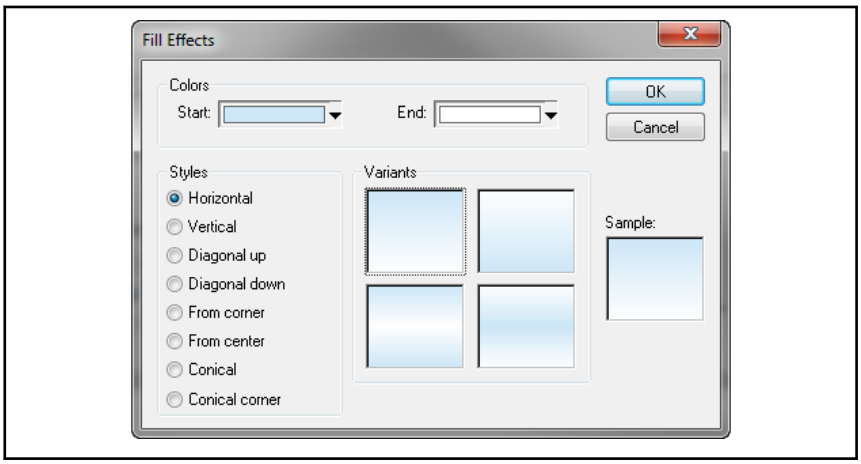


Fig. 8-123: Fill Effects dialog

2. Select two colors in the **Start** and **End** fields, select the "Style" and click on the desired "Variant". Finally, click **OK** to apply the fill effect to the component which is being edited

For projects on Windows CE target systems, the **Fill Effects** interface is available only for **Rectangle** objects.

Although **fill effects** are a useful tool for enhancing the look and effect of screens, the operating system takes longer to fill objects with fill effects than with plain colors. Particularly when creating Win CE user interfaces, this additional resource requirement is to be taken into consideration.

RGB code table

By means of the color animation on page 253, the color of objects can be modified during runtime. When this animation is configured with type = "By Color", the color to be displayed in the object during runtime can be set by the color code. The following table provides the code values as well as the RGB values for the most commonly used colors:

Name	RGB code			Code value
	R (Red)	G (Green)	B (Blue)	
Black	0	0	0	0
Dark red	128	0	0	128
Red	255	0	0	255
Pink	255	0	255	16711935
Rose	255	153	204	13408767
Brown	153	51	0	13209
Orange	255	102	0	26367
Light orange	255	153	0	39423
Gold	255	204	0	52479
Light brown	255	204	153	10079487
Olive green	51	51	0	13107
Dark yellow	128	128	0	32896
Lime	153	204	0	52377
Yellow	255	255	0	65535
Light yellow	255	255	153	10092543
Dark green	0	51	0	13056
Green	0	128	0	32768
Sea green	51	153	102	6723891
Bright green	0	255	0	65280
Light green	204	255	204	13434828
Dark teal	0	51	102	7877376
Teal	0	128	128	8421376
Aqua	51	204	204	13421619
Turquoise	0	255	255	16776960
Light turquoise	204	255	255	16777164
Dark blue	0	0	128	8388608
Blue	0	0	255	16711680
Light blue	51	102	255	16737843
Sky blue	0	204	255	16737843
Pale blue	153	204	255	16764057
Indigo	51	51	153	10040115

Name	RGB code			Code value
	R (Red)	G (Green)	B (Blue)	
Blue-gray	102	102	153	10053222
Violet	128	0	128	8388736
Plum	153	51	102	6697881
Lavender	204	153	255	16751052
Gray-80%	51	51	51	3355443
Gray-50%	128	128	128	8421504
Gray-40%	150	150	150	9868950
Gray-25%	192	192	192	12632256
White	255	255	255	16777215

The functions RGB Color on page 763 and RGB Component on page 763 can be used to adjust color codes during runtime.

The number of colors available when developing the project depends on the color settings configured on the operating system of the development station.

The number of colors available when running the project depends on the color settings configured on the operating system of the runtime station.

8.3.14 Fonts

Click on the tool  , **Fonts**, on the "WinStudio Object Properties" toolbar to specify the font and color for selected text objects on page 192 or to specify a default font and color for new text objects.



To save development time, several text objects can be selected simultaneously to apply the **Fonts** tool to specify font and color settings for all of the objects at once. (However, this function cannot be used for grouped on page 270 text objects.)

"WinStudio" supports any UNICODE font available on the operating system. Thus, it is possible to configure user interfaces for languages that do not use standard western characters, such as Japanese, Chinese, Arabic, Cyrillic, etc.

The font style of several objects can be changed simultaneously by selecting them all (press Shift while selecting the objects) and using the **Fonts** tool in the WinStudio toolbar.

The font of individual screen objects can also be adjusted by using the fonts button in the "Object Properties" window:

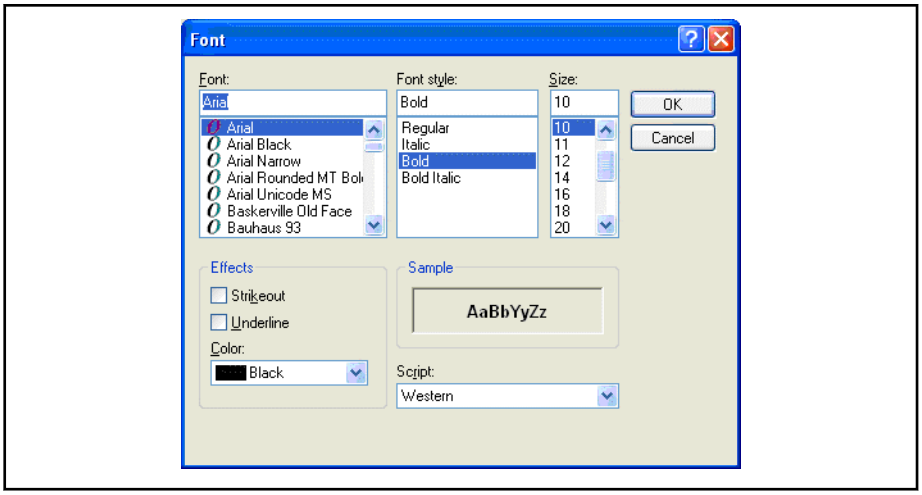


Fig. 8-124: Standard Fonts dialog

Font type, style, size, effects and script can be set.

The icon displayed to the left of the font name indicates the font format (technology)

Symbol	Technology	Information	
T	TrueType	Outline font format	TrueType and OpenType fonts are outline fonts that consist of line and curve commands. OpenType is an extension of TrueType. Both can be scaled and rotated. Both can be displayed in all sizes and on all output devices supported by Windows Windows provides a selection of OpenType fonts, including Arial, Courier New, Lucida Console, Times New Roman, Symbol, and Wingdings The font Type1 by Adobe Systems, Inc. is an outline font especially designed for PostScript printers. The "Outlines" can be scaled and rotated. With OpenType technology, Windows fully supports Type1 fonts
O	OpenType		
N/A	Vector		Vector fonts are supported as a number of programs still use this font format Vector fonts are rendered from a mathematical model. They are used primarily with plotters. Windows supports three vector fonts: Modern, Roman and Script
N/A	Grid		Raster fonts are supported as a number of programs still use this font format Raster fonts are stored in files as bitmap images and are composed of a series of dots both on the screen and on paper

⚠ CAUTION **It is strongly recommended to only use TrueType or OpenType fonts.**

Fonts designed with other technologies (e.g. Courier) cannot be scaled properly and can cause issues during runtime.

When designing screens, the fonts available in the operating system of the development station are provided.

The fonts on the runtime station, however, may look different (e.g. different size in pixels), even if all settings are the same on both stations.

These situation occurs more frequently if projects are executed on CE operating systems where font sizes are not always displayed identically. Thus, it is crucial to test the created screens during project development on the target platform.

8.4 Data input

User interfaces are often displayed on HMI screen. The HMI screen can only be operated using the touch screen instead of real keyboards. In this case, the user has to be able to enter data such as numerical values or texts via the touch screen.

This chapter describes how to configure data input options in the project.

The following screen objects, animations and functions accept data input via the user interface.

- **Text object with animation "Text data link"**

  The option "Input active" has to be selected in the object properties.

  See: chapter "Text data link animation" on page 251
- **Text box object**

  The option "Input active" has to be selected in the object properties.

  See: chapter "Text object" on page 192
- **Combobox object**

  The option "Input active" has to be selected in the object properties.

  See: chapter "Combo box object" on page 208
- **Alarm/event control object**

  To add comments and to filter the list.

  A virtual keyboard has to be configured in the object settings.

  See: chapter 9.3 "Alarm/event control object" on page 331
- **Trend control object**

  To change time and period of the trend graph.

  See: chapter 9.5 "Trend control object" on page 349
- **Grid object**

  In each column activated for input.

  See: chapter 9.6 "Grid object" on page 381
- **Function "KeyPad"**

  See: chapter 19.14.15 "KeyPad" on page 700
- **Function "ShowInplaceInput"**

  See: chapter 19.12.5 "ShowInplaceInput" on page 671



If the project safety system is enabled for your project, the user has to be able to enter user name and/or password: To login, to acknowledge an event or to change the password.

See: chapter 12 "Project security" on page 461

If data input is required in the previously mentioned examples, e.g. when an object is tapped or a function is called, a virtual keyboard can be displayed on the regular project screen.

The virtual keyboard type depends on the "Thin Clients" type the user accesses the project with.

Data input in Secure Viewer and Web Thin Client

The virtual keyboard is activated in separate settings for each Thin Client type:

- The Viewer settings are configured in the WinStudio project settings for activation in the Secure Viewer (including the server runtime environment).
- Configuration takes place in the project web settings for activation in the Web Thin Client.



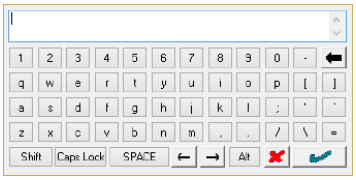
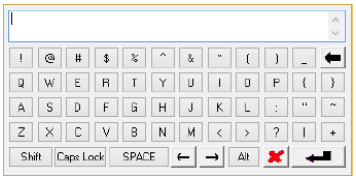
If the virtual keyboard is enabled for each Thin Client type, the keyboard is displayed on all client stations using the same client software. Irrespective of whether the stations are equipped with a touch screen or a real keyboard.

Possible effects on the operation have to be taken into account, in particular in case of mixed client stations (virtual/real keyboard).

Virtual keyboard types and options

When activating the virtual keyboard, the keyboard type as well as additional options can be configured. The following table describes the keyboard types and options:

Table type and options	Description	Representation
Keypad	To enter numerical values	
Keypad with enabled option "Min/max array"	To enter numerical values The minimum/maximum limit value configured for the variable is displayed.	
EnhKeypad	To enter numerical values, in particular for small operating displays Use -> to switch between the different keypads.	

AlphaNumeric	Keyboard to enter alphanumerical values.	
AlphaNumeric with activated option Enable multi-line text input	Keyboard for multi-line input of alphanumerical characters with line breaks (CR+LF). When Caps Lock is enabled, the OK/Accept key in the lower right corner of the keyboard changed to a Return key.	

As soon as the virtual keyboard has been enabled an a standard keyboard type is configured, this keyboard type is used for data input in the user interface.

Additionally, an individual keyboard type can be configured for individual image objects or function calls.

In case of image objects, the keyboard type can be configured in the object properties.

The keyboard type for the function call is determined in the optional function parameters.

For more information, refer to:

- chapter 8 "Screens and graphics" on page 169
- chapter 19.2 "List of Available Functions" on page 595

Modifying the size, position, language

By default, the virtual keyboard is displayed close to the object it is connected to. The keyboard size and position can be specified using the settings in the "HMI.app" WinStudio project file.

HMI.app

```
[Keypad]
VKType=<0-3>
// 0 (Default) = The Virtual Keyboard will work in
//               default mode Screens and Graphics
// 1 (Auto Size) = The Virtual Keyboard will automatically
//                 resize according to the screen size
// 2 (Manual Size/Position) = The entries below will be used

//Properties for AlphaNumeric
AlphaNumeric-PosX= // The TOP coordinate (in pixels) where the
//                    Virtual Keyboard must be displayed
AlphaNumeric-PosY= // The LEFT coordinate (in pixels) where the
//                    Virtual Keyboard must be displayed
AlphaNumeric-Width= // Virtual Keyboard width (in pixels)
AlphaNumeric-Height= // Virtual Keyboard height (in pixels)

//Properties for EnhKeypad
EnKeyPad-PosX=
EnKeyPad-PosY=
EnKeyPad-Width=
EnKeyPad-Height=

//Properties for Keypad with Min/Max fields enabled
KeyPadMinMax-PosX=
KeyPadMinMax-PosY=
KeyPadMinMax-Width=
KeyPadMinMax-Height=
```

```
//Properties for Keypad with Min/Max fields not enabled
KeyPad-PosX=
KeyPad-PosY=
KeyPad-Width=
KeyPad-Height=
```

The language setting of the virtual keyboard can be changed by calling the SetKeyboardLanguage function during runtime.

Refer to chapter 19.19.41 "SetKeyboardLanguage" on page 802

8.5 Multi-touch in your project

8.5.1 Using multi-touch in your project

Activate multi-touch gestures in your project screens to provide the end user with additional interface options.

On a conventional touch screen device which only supports one touch point, the touching or tapping by the user corresponds to a single mouse click. The end user cannot perform any actions he could not perform by connecting a mouse to the device.

With a newer touch screen device which supports several touch points, the end user can, however, manipulate project screens and screen objects using more than one finger at the same time. The additional touch points provide options which are not available with one single touch point. With two fingers at once, various gestures can be performed and different gestures cause different actions on the screen.

Example:

- Swipe two fingers in the same direction to quickly move windows or scroll through a list;
- Spread or pinch two fingers to change the size of a screen object or to zoom in or out of a project screen; or
- “Turn” a screen object using two fingers to rotate it like a dial.

With some experience with VBScript, it is possible to use touch events to customize the multi-touch behavior beyond the standard gestures described in this chapter. These touch events are actually VBScript sub-routines which receive touch input raw data from Windows API.

Requirements and considerations

These requirements do not apply to the computer which is used to develop your Rexroth WinStudio project. If it is possible to execute the development application and to create a project, the required settings can be configured in a way that ensures that multi-touch is integrated in your project. These requirements, however, do apply to the client devices your end users use to access your project.

To support multi-touch gestures, a client device must meet the following requirements: Microsoft Windows 7, Microsoft Windows 8, or Microsoft Windows Embedded Compact 7;

a touch screen display which is able to read one or more touch points and

either Web Thin Client or Secure Viewer for Windows, EmbeddedView for Windows Embedded Standard or CEView for Windows Embedded Compact must be installed.

To ensure that the client device supports multi-touch gestures, open the control panel (**Start** ▸ **Control Panel** ▸ **System and Security** ▸ **System**) and look for Pen and Touch. It must say that a touch input with at least two points is available.

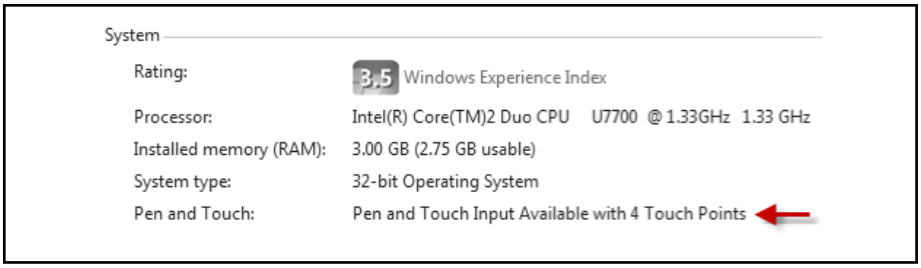


*Fig. 8-125: **Pen and Touch** in the Control Panel*

If your project is to be used in various environments - i.e., if your end users will use different kinds of client devices including ones which do not support touch input - consider carefully how to use multi-touch gestures in your project. Always ensure that there is a second option to manipulate a screen or object by using a keyboard or mouse or by tapping it with a finger.

Support in Windows Embedded Compact

The support of multi-touch gestures on Windows Embedded Compact devices is limited. The exact limitations are described in the respective chapters but the following table provides a summary.

Function	Windows 7 / Windows 8	Windows Embedded Compact
Zoom and scroll gestures on project screens	Supported	Not supported
Gestures with alarm/event control object	Supported	“Slide to scroll” supported; “Swipe to select” not supported
Gestures with trend control object	Supported	Supported
Gestures with grid object	Supported	“Slide to scroll/pan” supported; “Swipe to select” not supported
Gestures with position animation	Supported	Supported
Gestures with resize animation	Supported	Supported
Gestures with rotation animation	Supported	Not supported
Touch events for up to 10 touch points	Supported	Not supported

Tab. 8-3: Support of multi-touch gestures according to operating system

8.5.2 Multi-touch settings for project screens

Multi-touch settings for project screens, General information

The multi-touch settings determine which impact multi-touch gestures have in each project screen.

It is possible to configure standard settings for your entire project so that each new project screen has exactly the same settings as all the other screens of the same type and to customize the settings for individual screens if the standard settings are unsuitable.

Configuring multi-touch settings for all project screens

Configure the standard multi-touch settings for all screens in your project so that all screens behave in the same way during runtime.

The multi-touch settings are pre-configured for most conventional actions so that they only have to be configured further if:

- the runtime behavior is not satisfactory; or
- project tags are to be used to change the settings in the program during runtime.

Please note that this concerns the standard settings for all screens in your project. To configure the settings for an individual screen, refer to Configuring the multi-touch settings for an individual screen, page 289.

Configuring the standard multi-touch settings for all screens:

1. To open the **Project settings** dialog, right-click on the open WinStudio screen and select the "Properties" entry in the context menu. Open the **Viewer** dialog.

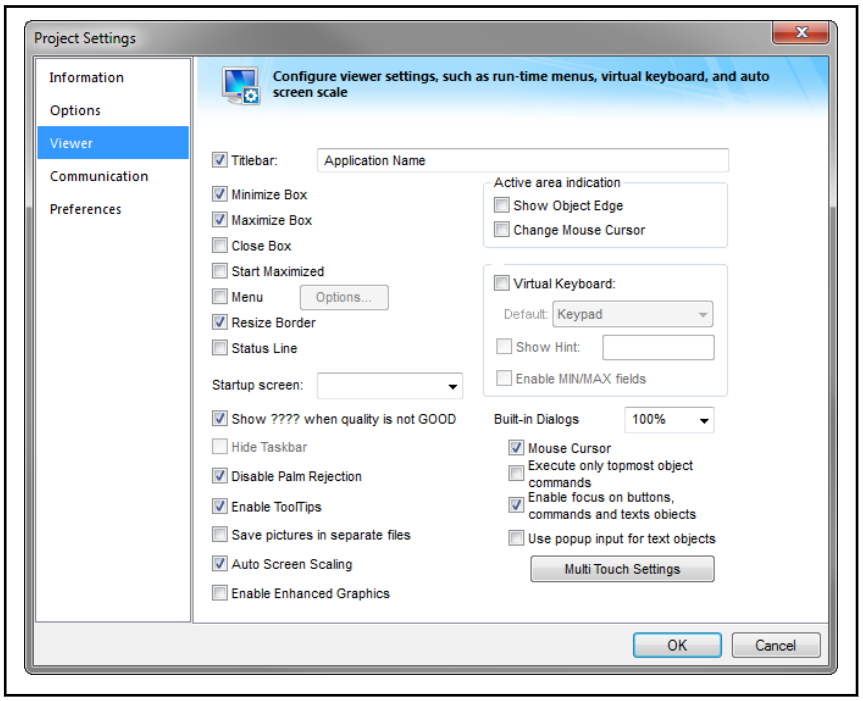


Fig. 8-126: Viewer settings

2. Click on **Multi touch settings**. The Multi touch settings dialog is displayed.

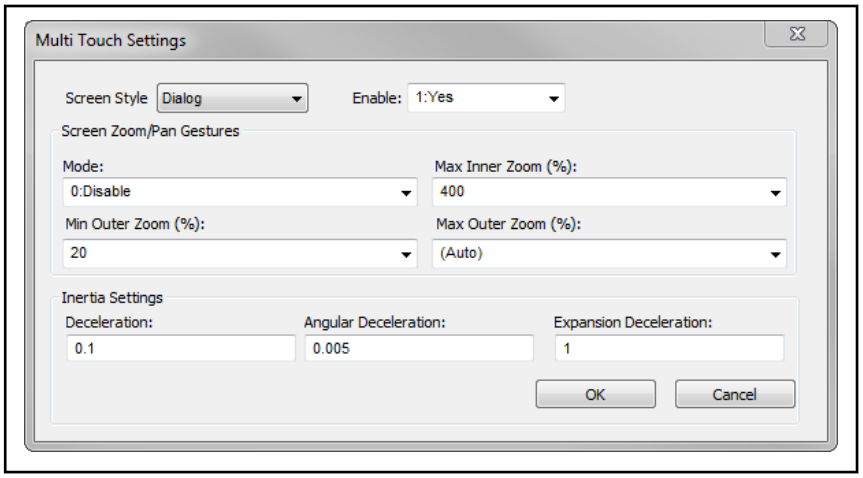


Fig. 8-127: Multi touch settings dialog

3. In the Screen style list, click on the style for which the standard settings are to be configured. Each style has individual settings so that the following steps need to be repeated for each style. For more information on the different styles of project screens, refer toScreen attributes dialog on page 170.
4. In the **Enable** field, either click on an option or enter the name of a project tag (Boolean or Integer type). This setting determines if the general multi-touch functions (e.g. gestures, screen zoom/pan gestures, inertia, touch events) are enabled for the selected screen style. When the name of a project tag is entered, this tag value controls the setting during runtime.
   - **0:No** The multi-touch functions are disabled for the selected screen style.
   - **0:Yes** The multi-touch functions are enabled for the selected screen style.
5. In the **Screen zoom/Pan gestures** section, configure the zoom settings.
   5.1 In the **Mode** field, either click on an option or enter the name of a project tag (Integer type). This setting determines the zoom/pan mode for the selected screen style. When the name of a project tag is entered, this tag value determines the mode during runtime.
      - **0:Disabled** Zoom/pan is disabled for the selected screen style.
      - **0:Inner** The inner zoom/pan is enabled for the selected screen style. Zooming changes the size of the visible area and panning moves the visible area in the screen frame. In most cases, the screen itself does not change its size or position.
      - **0:Outer** Zoom/pan is enabled for the selected screen style. In practice, it is more about resizing/panning; Zooming changes the size of the entire screen (and thereby enlarges what is displayed on the screen) and panning moves the screen in relation to the other open screens.

      For further information, refer to Using multi-touch gestures in project screens on page 292.

   5.2 In the **Max. inner zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the maximum admissible enlargement of the visible area in the project screen if the **Mode** is set to **Inner**. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 100 (i.e. 100%) and above 1000 (i.e. 1000%) are ignored.

   5.3 In the **Min. outer zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the minimum admissible size for the project screen (as a percentage of the size specified in the screen settings) if the **Mode** is set to **Outer**. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 20 (i.e. 20%) and above 100 (i.e. 100%) are ignored.

   5.4 In the **Max. outer zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the maximum admissible size for the project screen (as a percentage of the size specified in the screen settings) if the **Mode** is set

to **Outer**. The default option is **(Auto)**. This means that the maximum screen size corresponds to the size of the display on which the project is displayed. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 100 (i.e. 100%) and above 1000 (i.e. 1000%) are ignored.



These settings only apply to projects which run on Windows since multi-touch gestures in project screens are not supported by Windows Embedded Compact.

6. In the **Inertia settings** section, configure the deceleration values for the different types of movement. All values are indicated in pixels per second.
   6.1 In the **Deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Slide to pan” and “Slide to move” gestures.
   6.2 In the **Angular deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Rotate” gesture.
   6.3 In the **Expansion deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Pinch/stretch to resize” and “Pinch/stretch to zoom” gestures.

   For a detailed explanation on the use of inertia in multi-touch gestures, refer to “Inertia Mechanics” on the Microsoft Developer Network website:

   msdn.microsoft.com/library/dd562169.aspx



These settings only apply to projects which run on Windows since inertia for touch inputs is not supported by Windows Embedded Compact.

7. Repeat these steps from step 3 for each screen style you want to configure.
8. Click <OK> to apply the changes and close the dialog.

Configuring the standard multi-touch settings for an individual project screen

Configure the multi-touch settings if the standard settings of the project are not suitable.

Before proceeding with this task, first open the selected screen worksheet for editing.

Please note that the default settings of the project ensure that all screens behave in the same way during runtime. Consistency leads to a more user-friendly project. (For further information, refer to Configuring the standard multi-touch settings for all screens, page 286.) Only change the settings for a specific screen if it is absolutely necessary for the purpose of that screen.

Steps to configure multi-touch settings for a specific screen:

1. Right-click anywhere in the screen worksheet and then click **Screen attributes** in the context menu.

   The Screen Attributes dialog is displayed.

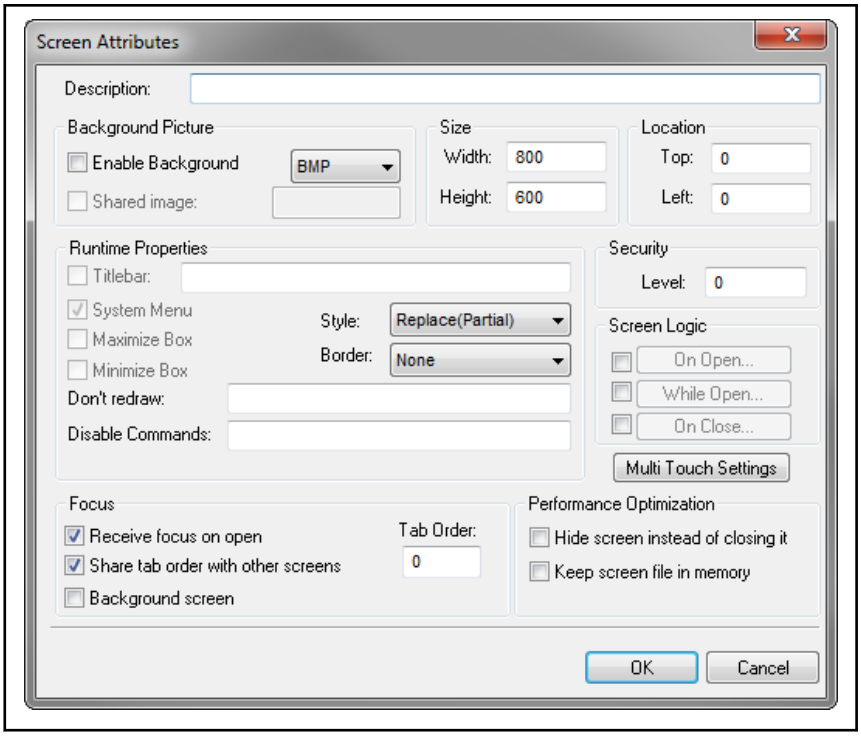


Fig. 8-128: Screen Attributes dialog

2. Click on **Multi touch settings**. The Multi touch settings dialog is displayed. Most settings are disabled since the screen uses the default settings of the project.
3. Disable the **Use project default** option.

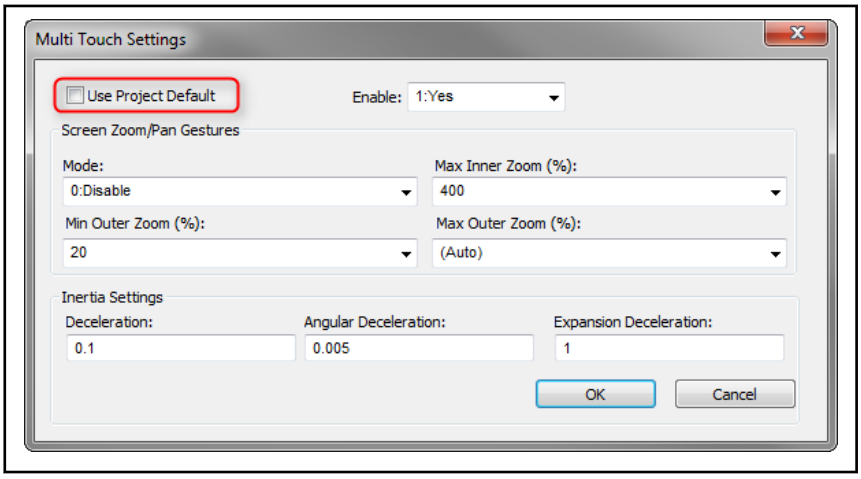


Fig. 8-129: Multi touch settings dialog

Now, it is possible to configure the remaining settings.

4. In the **Enable** field, either click on an option or enter the name of a project tag (Boolean or Integer type). This setting determines if the general multi-touch functions (e.g. gestures, screen zoom/pan gestures, inertia, touch events) are enabled for the selected screen style. When the name of a project tag is entered, this tag value controls the setting during runtime.
   - **0:No** The multi-touch functions are disabled for the selected screen style.

- **0:Yes** The multi-touch functions are enabled for the selected screen style.

5. In the **Screen zoom/Pan gestures** section, configure the zoom settings.

   5.1 In the **Mode** field, either click on an option or enter the name of a project tag (Integer type). This setting determines the zoom/pan mode for the selected screen style. When the name of a project tag is entered, this tag value determines the mode during runtime.

   - **0:Disabled** Zoom/pan is disabled for the selected screen style.
   - **0:Inner** The inner zoom/pan is enabled for the selected screen style. Zooming changes the size of the visible area and panning moves the visible area in the screen frame. In most cases, the screen itself does not change its size or position.
   - **0:Outer** Zoom/pan is enabled for the selected screen style. In practice, it is more about resizing/panning; Zooming changes the size of the entire screen (and thereby enlarges what is displayed on the screen) and panning moves the screen in relation to the other open screens.

   For further information, refer to chapter "Using multi-touch gestures in project screens" on page 292

   5.2 In the **Max. inner zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the maximum admissible enlargement of the visible area in the project screen if the **Mode** is set to **Inner**. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 100 (i.e. 100%) and above 1000 (i.e. 1000%) are ignored.

   5.3 In the **Min. outer zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the minimum admissible size for the project screen (as a percentage of the size specified in the screen settings) if the **Mode** is set to **Outer**. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 20 (i.e. 20%) and above 100 (i.e. 100%) are ignored.

   5.4 In the **Max. outer zoom** field, either click on an option or enter the name of a project tag (Integer type). This determines the maximum admissible size for the project screen (as a percentage of the size specified in the screen settings) if the **Mode** is set to **Outer**. The default option is **(Auto)**. This means that the maximum screen size corresponds to the size of the display on which the project is displayed. When the name of a project tag is entered, this tag value determines the zoom at runtime. Values below 100 (i.e. 100%) and above 1000 (i.e. 1000%) are ignored.



These settings only apply to projects which run on Windows since multi-touch gestures in project screens are not supported by Windows Embedded Compact.

6. In the **Inertia settings** section, configure the deceleration values for the different types of movement. All values are indicated in pixels per second.

6.1 In the **Deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Slide to pan” and “Slide to move” gestures.

6.2 In the **Angular deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Rotate” gesture.

6.3 In the **Expansion deceleration** field, either enter a value or the name of a project tag (Real type) that contains the value. This value controls the deceleration of the “Pinch/stretch to resize” and “Pinch/stretch to zoom” gestures.

For a detailed explanation on the use of inertia in multi-touch gestures, refer to “Inertia Mechanics” on the Microsoft Developer Network website:

msdn.microsoft.com/library/dd562169.aspx



These settings only apply to projects which run on Windows since inertia for touch inputs is not supported by Windows Embedded Compact.

7. Click <OK> to apply the changes and close the dialog.

8.5.3 The different multi-touch gestures

The different multi-touch gestures

This section is about the different types of multi-touch gestures and how to use them in your project.

The gestures - swipe, slide, pinch, stretch and so on - are standard gestures in Windows 7 and Windows 8. This means that you are possibly already familiar with their use on a smart phone or tablet. If not, figures of the gestures are included.

This chapter describes which gestures can be used in which parts of your project. The "Pinch" and "Stretch" gestures, for example, can be used to resize a project screen or a screen object using the resize animation or even to navigate through a trend graph.

Please note that the term "multi-touch gesture" almost always refers to a gesture with two fingers in this documentation. There are some exceptions like, for example, sliding with one finger to select cells in a grid object, but these exceptions will be described in detail in the respective topics.

Using multi-touch gestures in project screens

Depending on the configuration of the screen, multi-touch gestures can either be used to zoom and pan or to resize and move a project screen during runtime.

This depends in particular on whether the **Zoom/Pan mode** for the corresponding screen is set to **Inner** or **Outer**.

For further information, refer to Multi-touch settings for project screens on page 286.



For these gestures to work, at least two fingers have to touch the same project screen. If only one finger is on the screen and the others are outside of it, the touch input is ignored.

Inner zoom/pan

If the inner zoom/pan is enabled for a project screen, it is possible to use gestures with two fingers to zoom and pan the visible area within the screen border. Simply touch any part of the screen with your fingers during runtime and either pinch them or spread them to zoom or pan. In most cases, the size of the screen itself is not changed and the screen is not panned.



Fig. 8-130: Inner zoom/pan gestures (1)

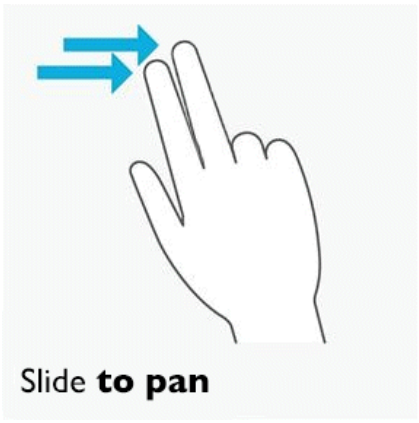


Fig. 8-131: Inner zoom/pan gestures (2)

These gestures are normally used if a diagram or a work area is to be viewed which is larger than your display. It is possible to load the large diagram in a smaller project screen and then use the zoom and pan function to view it.

If the zoom is 100% and the end user tries to pan, the entire screen moves and not the area visible on screen.

These gestures are not supported by Windows Embedded Compact.

Outer zoom/pan

If Outer zoom/pan is activated for a project screen, gestures with two fingers can be used to resize the screen and to pan it on the display as for resizing and moving windows on the desktop. Simply touch any part of the screen with your fingers during runtime and either pinch them or spread them to resize or move the screen.

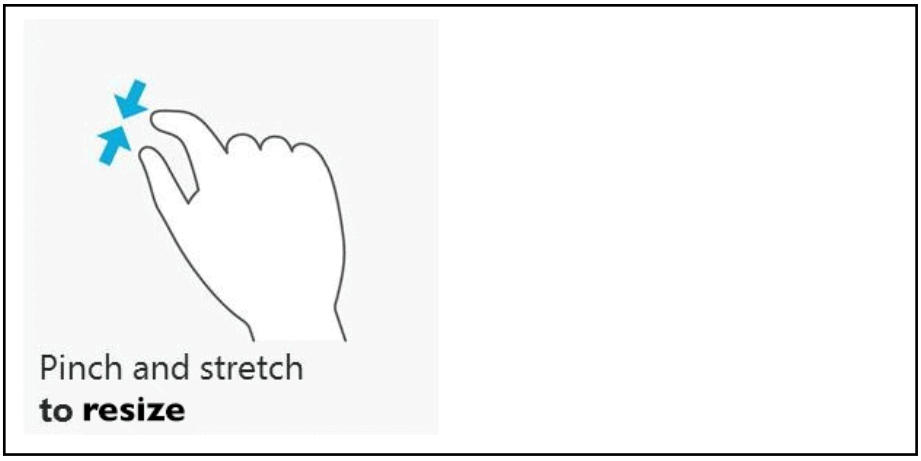


Fig. 8-132: Outer zoom/pan gestures (1)

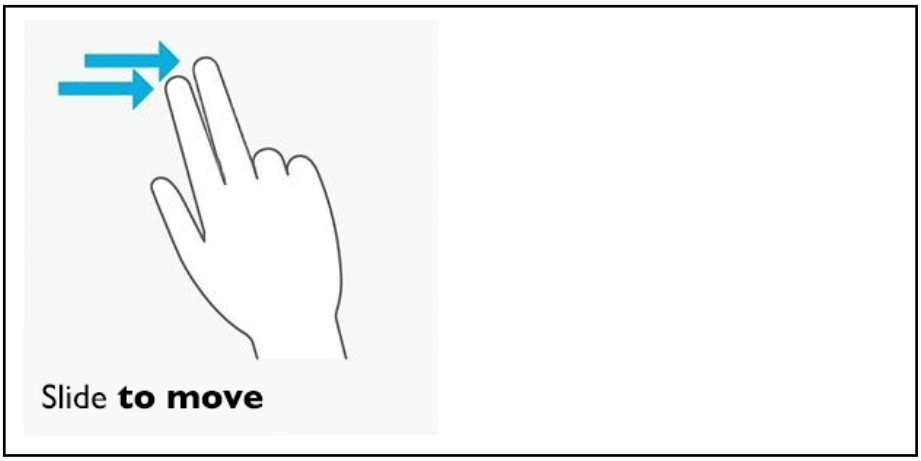


Fig. 8-133: Outer zoom/pan gestures (2)

This is essentially the same as using gestures with one finger (or a connected mouse) to resize or pan the screen. However, gestures with one finger only work for a screen with a border with resizing function (as configured under Screen attributes on page 170) and only if the border itself is "gripped".

These gestures are not supported by Windows Embedded Compact.

Using multi-touch gestures in data objects

You can use multi-touch gestures to manipulate an alarm/event, trend or grid object during runtime.

☞ For these gestures to work, all fingers have to touch the same screen object. If only one finger is on the object and the others are outside of it, the touch input is ignored.

Alarm/event control object

If multi-touch is enabled for a project screen, every Alarm/event control object on page 331 on this screen can be manipulated by multi-touch gestures. This means that it is possible to swipe with one finger to select elements in the list of alarms/events and to swipe with one or two fingers to scroll through the list.

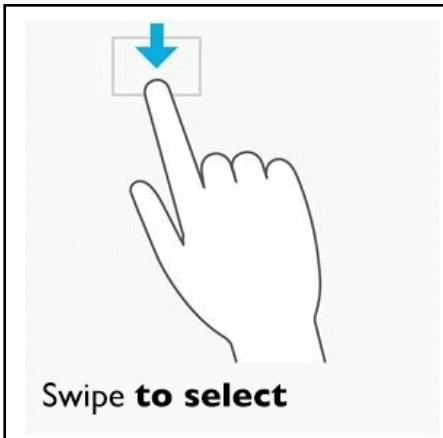


Fig. 8-134: Alarm/event control object gestures (1)

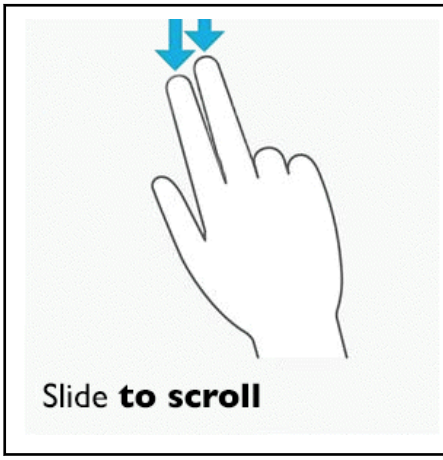


Fig. 8-135: Alarm/event control object gestures (2)

The “Slide to scroll” gesture (with only one finger) is supported by Windows Embedded Compact. The “Swipe to select” gesture is not supported.

Trend control object

If multi-touch is enabled for a project screen, every Trend control object on page 353 on this screen can be manipulated by multi-touch gestures. This means that it is possible to either pinch/stretch to zoom or slide to pan the visible area of the trend.

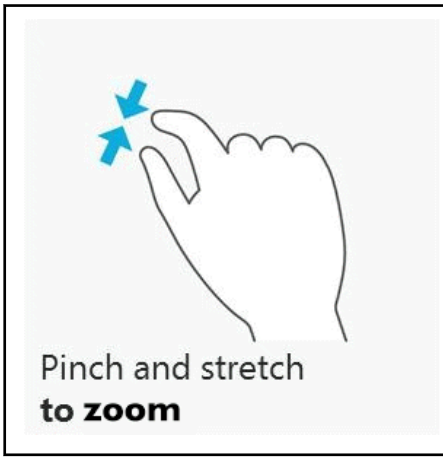


Fig. 8-136: Trend control object gestures (1)

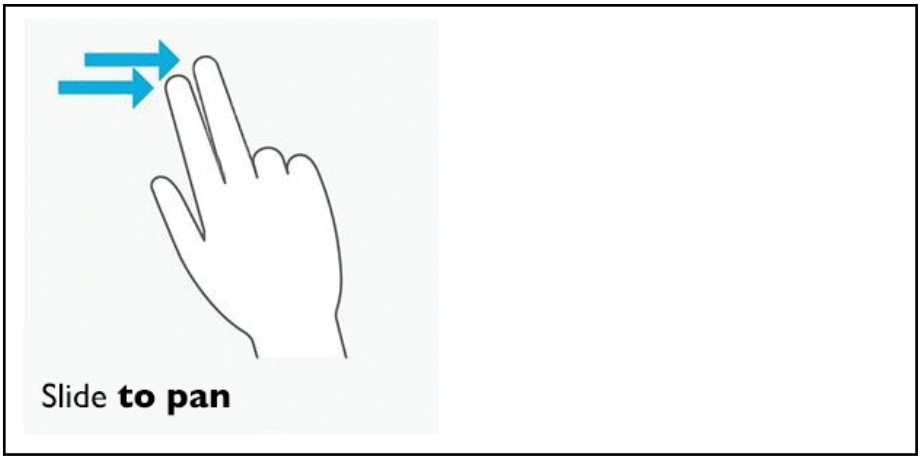


Fig. 8-137: Trend control object gestures (2)

Zooming changes the time period and the scale of the trend (i.e. the X and Y axes) as if one of the **Zoom** tools in the tool bar of the trend control object was used. If **Cancel zoom** is clicked in the tool bar, the zooming process initiated by your gestures is cancelled and the time period and the scale are reset.

Panning only works if the trend control object is configured to display historical data.

These two gestures are supported by Windows Embedded Compact.

Grid object

If multi-touch is enabled for a project screen, every Grid object on page 381 on this screen can be manipulated by multi-touch gestures. This means that it is possible to swipe with one finger to select cells in the grid and to slide with two fingers to scroll/pan the visible area of the grid.

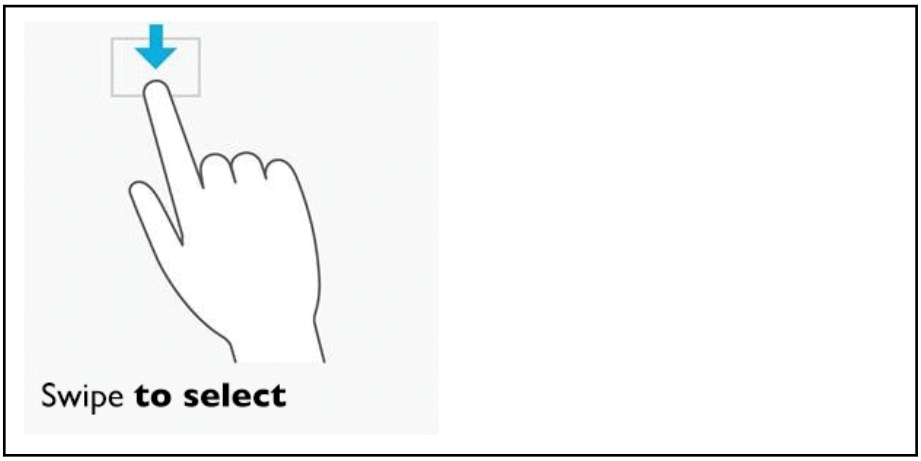


Fig. 8-138: Grid object gestures (1)

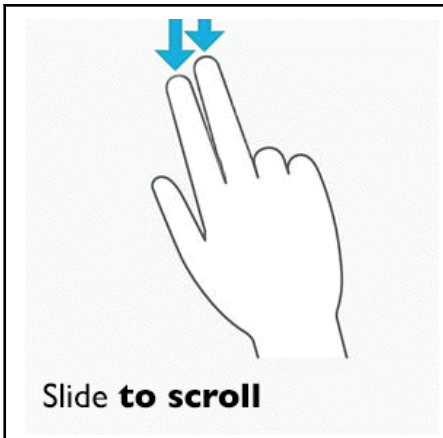


Fig. 8-139: Grid object gestures (2)

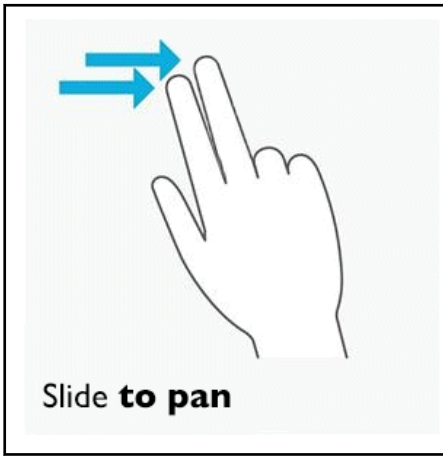


Fig. 8-140: Grid object gestures (3)

The “Slide to scroll” and “Slide to pan” gestures are both supported by Windows Embedded Compact. The “Swipe to select” gesture is not supported.

Using multi-touch gestures in object animations

Use multi-touch gestures to trigger a position, resize or rotation animation during runtime.

Gestures for position and resize animation are supported by Windows Embedded Compact. Gestures for rotation animation are not.



For these gestures to work, all fingers have to touch the same screen object. If only one finger is on the object and the others are outside of it, the touch input is ignored.

Position animation

If multi-touch is enabled for a project screen, every object with a Position animation on page 255 on this screen can be manipulated by multi-touch gestures. This means that it is possible to slide with one or more fingers to move the object.

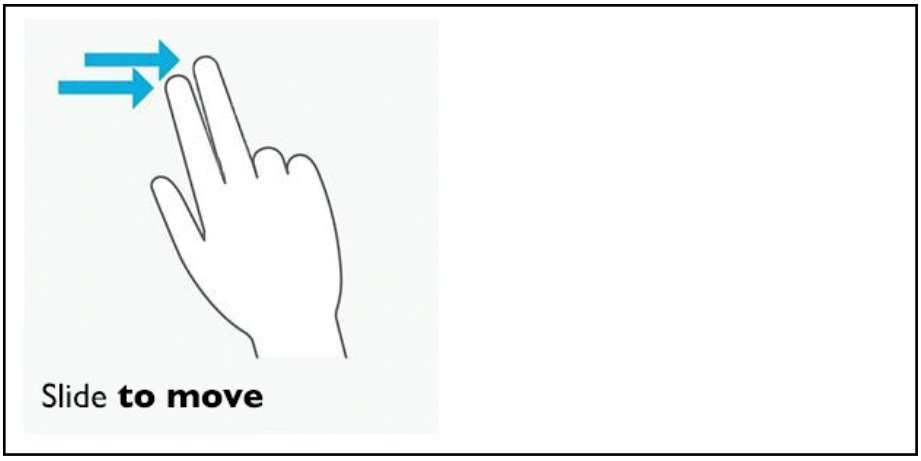


Fig. 8-141: Gestures for position animation

This is essentially the same as if multi-touch is deactivated since the purpose of the position animation is to make the object movable. In this case the main advantage of enabling multi-touch is the inertia which makes the movements of the object look more natural.

This gesture is supported by Windows Embedded Compact.

Resize animation

If multi-touch is enabled for a project screen, every object with a resize animation on this screen can be manipulated by multi-touch gestures. This means that it is possible to pinch and stretch the object with two fingers to change its size.

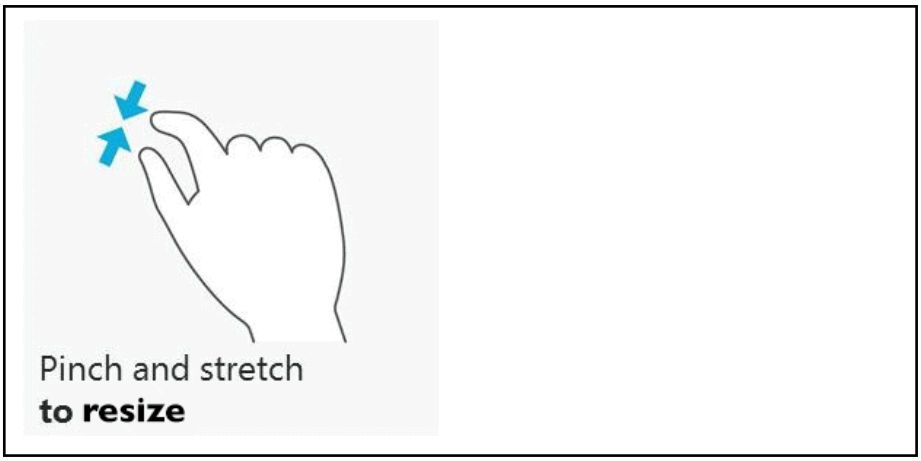


Fig. 8-142: Resize animation gestures

This gesture is supported by Windows Embedded Compact.

Rotation animation

If multi-touch is enabled for a project screen, a gesture with two fingers can be used to "turn" to rotate every object with a rotation animation. Simply touch the object with your fingers and turn it.

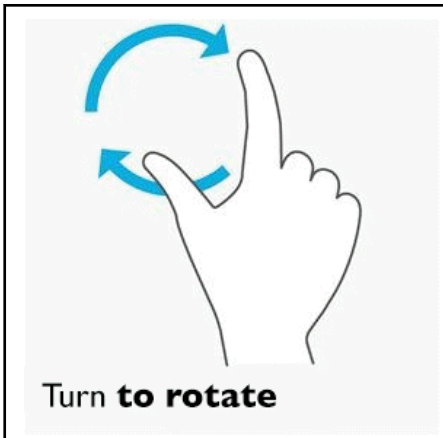


Fig. 8-143: Rotation animation gesture

This gesture is not supported by Windows Embedded Compact.

8.5.4 Touch events

Using touch events

Touch events are pre-defined VBScript sub-routines which can be added to screen objects and project screens to achieve a customized touch behavior. These touch events are based directly on the Windows Touch API - in particular on the _IManipulationEvents interface and the ManipulationStarted, ManipulationDelta and ManipulationCompleted methods. For further information, refer to "Windows Touch" on the Microsoft Developer Network website: msdn.microsoft.com/library/dd562197.aspx.

These methods transfer the raw data of the touch input for up to 10 touch points from the Windows operating system to Rexroth WinStudio, and WinStudio then transfers the data via touch events to your project.

In practice, touch events basically correspond to any other VBScript interface on page 925 in your project. Select a screen object or a project screen, add the corresponding touch event, independent of when the script is to be executed (e.g. when the user starts or stops the touch), and then develop the script which performs the desired actions. The only actual differences between touch events and the other VBScript interfaces are: Touch events can only be executed as sub-routines. This means that they do not return any values and can only receive the selected touch input data as parameters. However, if it is possible to work within these guidelines, touch events can be used to develop customized behavior which exceeds the default gestures by far.

☞ Touch events are not supported by Windows Embedded Compact.

Adding a touch event to a screen object

Add a touch event to a screen object to process a touch input on this object.

Before proceeding with this task, first open the selected screen worksheet for editing. The screen object to which the touch event is to be added needs to exist already as well.

Touch events are added to a screen object via the Command animation on page 246.

Touch events are not supported by Windows Embedded Compact.

Adding a touch event to a screen object:

1. Click on the screen object to which the touch event is to be added. The screen object is selected.
2. In the **WinStudio dynamic functions** toolbar, click on **Command**. The command animation is added to the selected screen object.
3. Double-click the screen object. The Object Properties dialog is displayed.
4. If the properties of the command animation are not displayed yet, click on Command in the properties list (in the top right corner of the dialog).
5. If the VBScript event tabs are not displayed yet, click on **VBScript** in the **Type** list.

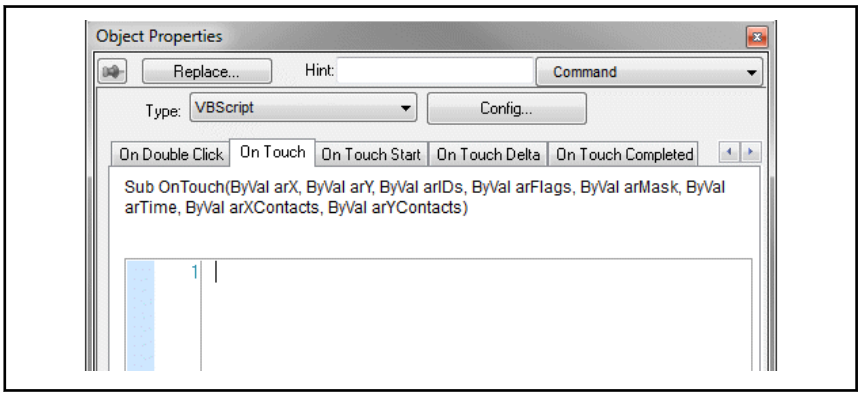


Fig. 8-144: Tabs for VBScript events in the command animation properties

6. On the tabs for VBScript events, click on the tab for the touch event that is to be added. It may be necessary to use the arrow keys to scroll through the tabs.

Tab	Description
On touch on page 301	Process the raw data of the touch input. The sub-routine is continuously executed while the user touches the screen object.
OnTouchStart on page 304	Execute an action once when the user touches the screen object.
OnTouchDelta on page 304	Execute an action every time the user manipulates the screen object.
OnTouchCompleted on page 306	Execute an action once when the user stops touching the screen object.

7. Click on the text field at the bottom and enter your VBScript code.

From now on, the touch event can be developed like any other VBScript sub-routine in your project. For further information, refer to Overview of VBScript on page 925.

Adding a touch event to a project screen

Add a touch event to a project screen to process a touch input on the screen in general instead of on an individual object on the screen.

Before proceeding with this task, first open the selected screen worksheet for editing.

Touch events are added to a project screen via the screen script worksheet.



Touch events are not supported by Windows Embedded Compact.

Adding a touch event to a project screen:

1. Right-click anywhere in the screen worksheet and then click **Screen script** in the context menu. The script worksheet for the screen is opened for editing.
2. Right-click anywhere in the script worksheet and then go to **Add touch event** in the context menu. A sub-menu of the available touch events is displayed.
3. In the sub-menu, click on the touch event that is to be added.

Tab	Description
On touch on page 301	Process the raw data of the touch input. The sub-routine is continuously executed while the user touches the screen object.
OnTouchStart on page 304	Execute an action once when the user touches the screen object.
OnTouchDelta on page 304	Execute an action every time the user manipulates the screen object.
OnTouchCompleted on page 306	Execute an action once when the user stops touching the screen object.

The touch event is inserted into the screen script worksheet as a VBScript sub-routine.

4. Click on the worksheet and enter your VBScript code.

From now on, the touch event can be developed like any other VBScript sub-routine in your project. For further information, refer to Overview of VBScript on page 925.

8.5.5 On touch

Use the On touch sub-routine in VBScript to process the raw data of the touch point which accumulate while the user touches the project screen or screen object.

Syntax

```
Sub OnTouch(arX,arY,arIDs,arFlags,arMask,arTime,arXContacts,arYContacts)
    …
End Sub
```

Inputs

arX

An array of Integer values from `arX(0)` to `arX(n)` which indicate the X coordinates (in pixels from the left of the screen) of the currently active touch points.

arY

An array of Integer values from `arY(0)` to `arY(n)` which indicate the Y coordinates (in pixels from the top of the screen) of the currently active touch points.

arIDs

An array of Integer values from `arIDs(0)` to `arIDs(n)` which indicate the unique IDs of the currently active touch points. Each individual touch point has its own identifier even if it is touched by the same finger then released and then touched again. These identifiers are increased incrementally after the device is switched on and include all touches which were recorded by the operating system, not only those which were recorded during the runtime of your project.

arFlags

An array of Integer values, from `arFlags(0)` to `arFlags(n)`, where every value represents a set of bit switches which determine different aspects of the touching, releasing and moving of touch points.

For further information on bit switches and their possible values, refer to "TOUCHINPUT structure" on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317334.aspx

arMask

An array of Integer values, from `arMask(0)` to `arMask(n)`, where every value represents a set of bit switches which determine which of the optional parameters (i.e., **arTime**, **arXContacts**, **arYContacts**) include valid information. The availability of valid information depends on the device: For **arTime**, some devices, for example, only indicate the time which has passed since the device was switched on instead of the current system time.

For further information on bit switches and their possible values, refer to "TOUCHINPUT structure" on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317334.aspx

arTime

An array of Integer values from `arTime(0)` to `arTime(n)` which indicate the time stamps (in milliseconds) of the currently active touch points.

arXContacts

An array of Integer values from `arXContacts(0)` to `arXContacts(n)` which indicate the widths (in hundredth of a pixel) of the contact areas of the currently active touch points. The contact area of a touch point is the area which is touched by the user's finger tip.

arYContacts

An array of Integer values from `arYContacts(0)` to `arYContacts(n)` which indicate the heights (in hundredth of a pixel) of the contact areas of the currently active touch points. The contact area of a touch point is the area which is touched by the user's finger tip.

Returned value

It is a subroutine (no function) in VBScript. No value is returned.

Notes

This sub-routine is based on the WM_TOUCH system message and the corresponding TOUCHINPUT data structure in the Windows API. For further information, refer to "Windows Touch Input on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317321.aspx.

The sub-routine is continuously executed while the user touches the project screen or screen object. There are no delta or cumulative values which means that nothing needs to be reset when the manipulation stops. This is raw data output by the Windows API.

The received parameters do not have to be used in your code. They only provide the raw data of the touch input which can be used (or not used) as required.

In all parameters described above, the array elements represent individual touch points on the screen, in the order in which the user actually touched the screen. The first array element (position 0) is the first touch point, the second array element (position 1) is the second touch point, and so on until the maximum number of touch points supported by the device is reached.

Please note that the size of the arrays is changed dynamically according to the number of active touch points. In other words: The array elements do not exist until the user's finger actually touches the screen and the corresponding touch points are added. The array elements are deleted again if the touch points are removed. This is why it may be difficult to reference array elements in your project unless the following code (or something similar) is integrated in the sub-routine:

```
n = UBound(arX)

For i = 0 to n
    $TouchX[i] = arX(i)
    $TouchY[i] = arY(i)
    $TouchID[i] = arIDs(i)
    $TouchTime[i] = arTime(i)
Next
```

The UBound function measures the current size of **arX** (all parameters can be used for this). Then the For loop copies the values into correspondingly named tag arrays (e.g. `TouchX`, `TouchY`) in the tag database of your project. When this is finished, the tag arrays instead of the parameters can be referenced. In contrast to parameters, the size of the tag arrays is not changed dynamically. This means that garbage values may remain in the higher array positions when touch points are removed. To remove these garbage values, you can integrate the following code (or something similar) in the sub-routine:

```
s = $TouchX->Size

For i = (n+1) to s
    $TouchX[i] = 0
    $TouchY[i] = 0
    $TouchID[i] = 0
    $TouchTime[i] = 0
Next
```

In the meantime, it has probably become obvious that there is no elegant way to remove array elements from another array position than the highest. If the user touches the screen with two fingers and then removes the second finger, the second element of the array (position 1) is deleted without problems. However, if the user touches the screen with two fingers and then removes the first finger, the first element of the array (position 0) is deleted and the second element (position 1) becomes the first element (position 0).

To handle specific touch points long-term, the unique identifiers provided by **arIDs** can be used instead of array positions that change when the size of the arrays is changed dynamically. How this works exactly, however, depends on how you develop the rest of your project and is therefore beyond the scope of the present documentation.

8.5.6 OnTouchStart

Use the On TouchStart sub-routine in VBScript to perform an action when the user touches the project screen or screen object.

Syntax

```
Sub OnTouchStart(x,y)
    …
End Sub
```

Inputs

x

The initial X coordinate (in pixels from the left of the screen) of the first touch point.

y

The initial Y coordinate (in pixels from the top of the screen) of the first touch point.

Returned value

It is a subroutine (no function) in VBScript. No value is returned.

Notes

This sub-routine is based on the ManipulationStarted method in the Windows API.

For further information, refer to “Windows Touch Input on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317321.aspx.

The sub-routine is executed once when the manipulation starts - i.e., when the first touch point is added to the project screen or screen object. Additional touch points after the first release do not trigger this sub-routine.

The received parameters do not have to be used in your code. They only provide the raw data of the touch input which can be used (or not used) as required.

8.5.7 OnTouchDelta

Use the On TouchStartDelta sub-routine in VBScript to perform an action every time the user manipulates the project screen or screen object.

Syntax

```
Sub OnTouchDelta(x,y,deltaX,deltaY,deltaScale,
                 deltaExpansion,deltaRotation,cumulativeX,cumulativeY,cumulativeScale,
                 cumulativeExpansion,cumulativeRotation,inertiaEnabled)
    …
End Sub
```

Inputs

x

The current X coordinate (in pixels from the left of the screen) of the first touch point.

y

The current Y coordinate (in pixels from the top of the screen) of the first touch point.

deltaX

The change (in pixels) between the previous and the current X coordinate of the first touch point.

deltaY

The change (in pixels) between the previous and the current Y coordinate of the first touch point.

deltaScale

The change (as a percentage) of the distance between the first and the second touch point.

deltaExpansion

The change (in pixels) of the distance between the first and the second touch point.

deltaRotation

The change of the rotation angle (as rotary unit) which is determined by the first and second touch point.

cumulativeX

The total change (in pixels) between the initial and the current X coordinate of the first touch point.

cumulativeY

The total change (in pixels) between the initial and the current Y coordinate of the first touch point.

cumulativeScale

The total change (as a percentage) of the distance between the first and the second touch point from the start of the manipulation.

cumulativeExpansion

The total change (in pixels) of the distance between the first and the second touch point from the start of the manipulation.

cumulativeRotation

The total change of the rotation angle (as rotary unit) which is determined by the first and second touch point from the start of the manipulation.

inertiaEnabled

A Boolean value which indicates whether inertia is enabled for the project screen or screen object.

Returned value

It is a subroutine (no function) in VBScript. No value is returned.

Notes

This sub-routine is based on the ManipulationDelta method in the Windows API.

For further information, refer to “Windows Touch Input on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317321.aspx.

The sub-routine is executed once for each individual movement in the current manipulation. Position changes require one touch point. Changes in size or rotation require two touch points.

The received parameters do not have to be used in your code. They only provide the raw data of the touch input which can be used (or not used) as required.

8.5.8 OnTouchCompleted

Use the On TouchCompleted sub-routine in VBScript to perform an action when the user stops touching the project screen or screen object.

Syntax

```
Sub OnTouchCompleted(x,y,cumulativeX,cumulativeY,cumulativeScale,
                     cumulativeExpansion,cumulativeRotation)
    …
End Sub
```

Inputs

x

The final X coordinate (in pixels from the left of the screen) of the first touch point.

y

The final Y coordinate (in pixels from the top of the screen) of the first touch point.

cumulativeX

The total change (in pixels) between the initial and the final X coordinate of the first touch point.

cumulativeY

The total change (in pixels) between the initial and the final Y coordinate of the first touch point.

cumulativeScale

The total change (as a percentage) of the distance between the first and the second touch point from the start to the end of the manipulation.

cumulativeExpansion

The total change (in pixels) of the distance between the first and the second touch point from the start to the end of the manipulation.

cumulativeRotation

The total change of the rotation angle (as rotary unit) which is determined by the first and second touch point from the start to the end of the manipulation.

Returned value

It is a subroutine (no function) in VBScript. No value is returned.

Notes

This sub-routine is based on the ManipulationCompleted method in the Windows API.

For further information, refer to “Windows Touch Input on the Microsoft Developer Network website: msdn.microsoft.com/library/dd317321.aspx.

The sub-routine is executed once when the manipulation ends - i.e., when the final touch point is removed from the project screen or screen object.

The received parameters do not have to be used in your code. They only provide the raw data of the touch input which can be used (or not used) as required.

8.6 Converting screen resolution

General notes on the project resolution

WinStudio screens are generally created for a specific screen size. These can either be used as full screen, partial screen or as dialog.

The screen size is the display resolution of the visualization device executing the WinStudio application. For the resolution currently used in the WinStudio project, refer to the “Properties” dialog of the "WinStudio" element under "Options". These values are specified when inserting a visualization device into the IndraWorks project by selecting a special device type. The values cannot be changed later on.

When adding new WinStudio screens, their default size is the size set in the project resolution. The size of each WinStudio screen can subsequently be changed to use the screen as full screen, partial screen or dialog window.

Converting WinStudio screens

To use individual WinStudio screens or the complete WinStudio application on a device, convert the WinStudio screens to the resolution. The size of the WinStudio screen is adapted to the new project resolution and the sizes and positions of the respective screen objects are proportionally scaled to the new screen size. IndraWorks provides support when you convert your WinStudio screens by selecting whether the sizes and positions of the screen objects of the WinStudio screens are to be scaled.

To insert an individual WinStudio screen with a different resolution in an existing WinStudio application, proceed as follows:

1. Export the individual WinStudio screen.
2. Import the WinStudio screen into the new WinStudio application.
3. Click on "Convert WinStudio screen" in the context menu of the imported WinStudio screen.
4. In the conversion dialog, select the options suitable for the WinStudio screen and press "Convert".

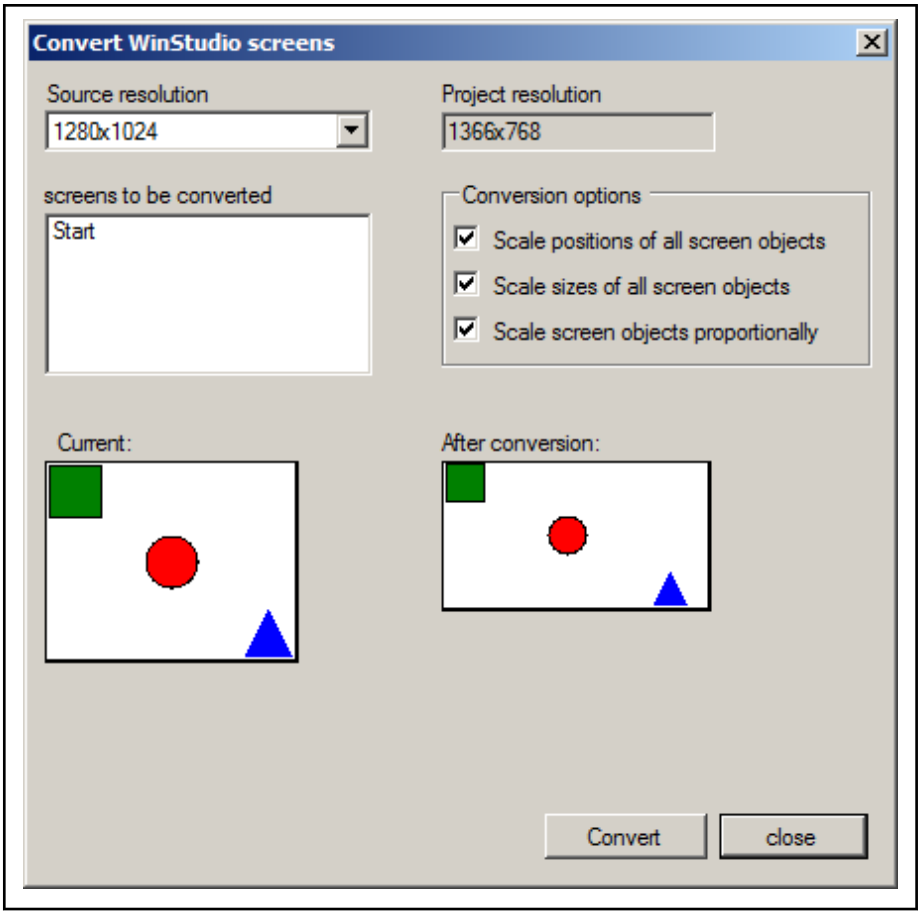


Fig. 8-145: Conversion dialog for WinStudio screens

During conversion, the size of the WinStudio screen is proportionally scaled to the resolution of the WinStudio project. The sizes and positions of the screen objects depend on the selected conversion options:

Scale positions of all screen objects

If this option is selected, the positions of all screen objects in x- and y-direction are proportionally converted with regard to the new project resolution. Deselect this option and all positions remain unchanged.

Scale sizes of all screen objects

When selecting this option, the widths and the heights of all screen objects are proportionally converted into the new project resolution. Deselect this object to keep all object sizes unchanged.

☞ If the aspect ratio of source resolution and project resolution differ, the screen objects are differently scaled in x- and y-direction.

Scale screen objects proportionally

This option is only available if the source resolution and the current project resolution differ in their aspect ratio. If this option is selected, widths and heights of all screen objects are converted with the identical scaling factor. This avoids distortions in the screen objects.

☞ The reference point to scale the size and the position of each screen object is the upper center point.

If WinStudio screens have been imported from multiple projects with different resolution, go to the context menu of the "WinStudio" folder to open the conversion dialog. In this case, the drop-down list "Source resolution" contains all resolutions of the imported WinStudio screens with a resolution deviating from the current project resolution. Select the source resolution of the WinStudio screens to be converted and modify the conversion options if required. The "Convert" button starts the conversion for the selected source resolution. All screens displayed under "WinStudio screens to be converted" are converted with the selected options. The next source resolution can now be selected and the corresponding WinStudio screens can be converted. Thus, the conversion can be repeated until all WinStudio screens are converted.

The conversion of WinStudio screens cannot be undone. If the conversion is not as expected, import it again.

Validating the screen resolution

When executing the "Validate" function, check whether the resolution of all WinStudio screens correspond to the current project resolution. A corresponding note is output in the message window for each WinStudio screen with a resolution deviating from the project resolution.

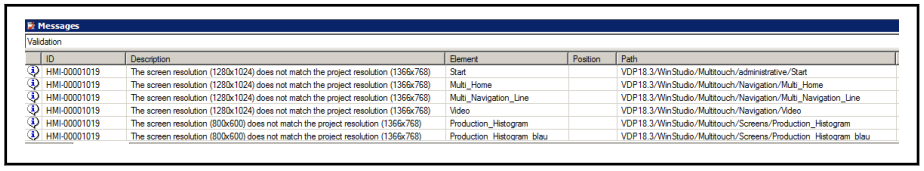


Fig. 8-146: Validating messages if WinStudio screen resolution deviates

It is recommended to convert these resolutions of the displayed WinStudio screens to avoid representation errors at application runtime. Open the conversion dialog as described via the context menu of a WinStudio screen or via the "WinStudio" folder. Alternatively, double-click on the message line. In this case, a dialog opens to select whether only the respective screen or all screens should be converted.

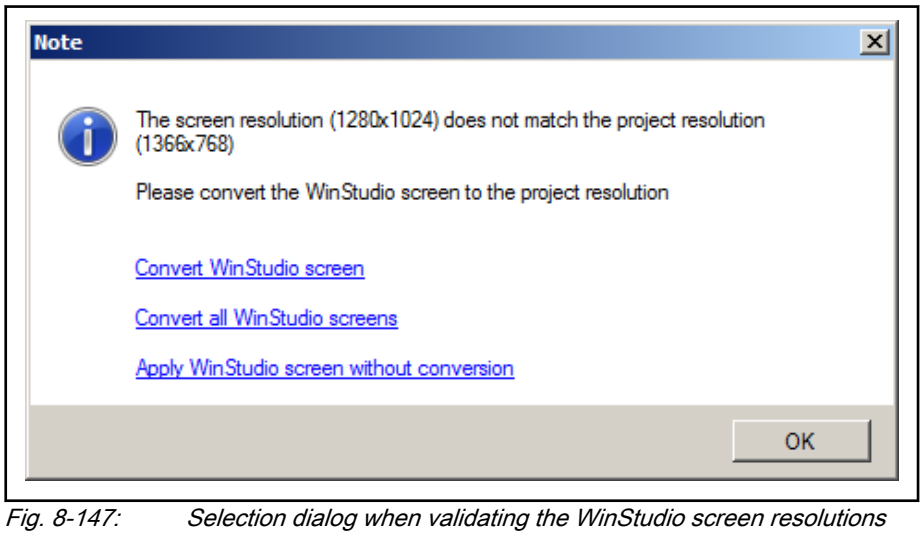


Fig. 8-147: Selection dialog when validating the WinStudio screen resolutions

If the representation of the respective WinStudio screen already corresponds to the current resolution and the correct format, select the option "Apply WinStudio screen without conversion". The screen remains unchanged and no note is displayed anymore for this screen during the subsequent calls of the validating function.

8.7 Importing Studio XML templates

With the help of the import wizard, XML templates can be imported and WinStudio screens can be generated based on them. The design of this external text file corresponds to the XML scheme for WinStudio screens.

Further editing is possible provided that an XML template file is available

A WinStudio XML template file contains the same information as a regular binary screen file (*.SCR). In contrast to the binary file, the content is available in readable form and is therefore more flexible and portable.

If XML templates were created for your screens, use the import wizard to import them in your WinStudio project.



It is also possible to use the ImportXML on page 749 function to import Studio XML screen files during runtime.

Importing one or several XML templates for WinStudio screens

1. In the context menu of the "WinStudio" folder, select **Tools** ▸ **Variable import wizard**.

   The Import wizard dialog window is displayed.

2. In the **Source type** list , click **Studio XML screen** and then **Next**. The next step of the import wizard is displayed.

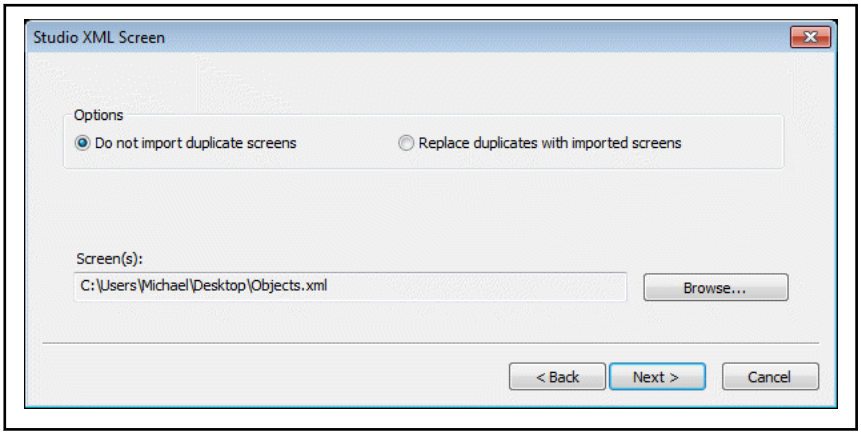


Fig. 8-148: Selection of the screens to be imported

3. Under **Options**, select if the imported screens are to automatically replace the existing screens in your project. Screens are considered as duplicates if they have the same file name. `Objects.xml` and `Objects.scr` would be duplicates, for example.

   - If you do not want the imported screens to automatically replace the existing screens in your project, select "Do not import duplicate screens". A warning is displayed for each duplicate you are trying to import.
   - If the imported screens are to automatically replace the existing screens in your project, select "Replace duplicates with imported screens".

4. Click **Browse** to open the file browser.
5. With the help of the file browser, select XML template files to import and click on "Open". It is possible to select several files by clicking on additional entries while holding down the <CTRL> key. The selected files are displayed in the **Screen(s)** field:

6. Click on **Next**. The selected screens are processed. In the next step of the import wizard, the WinStudio variables which are included in the XML templates are displayed,

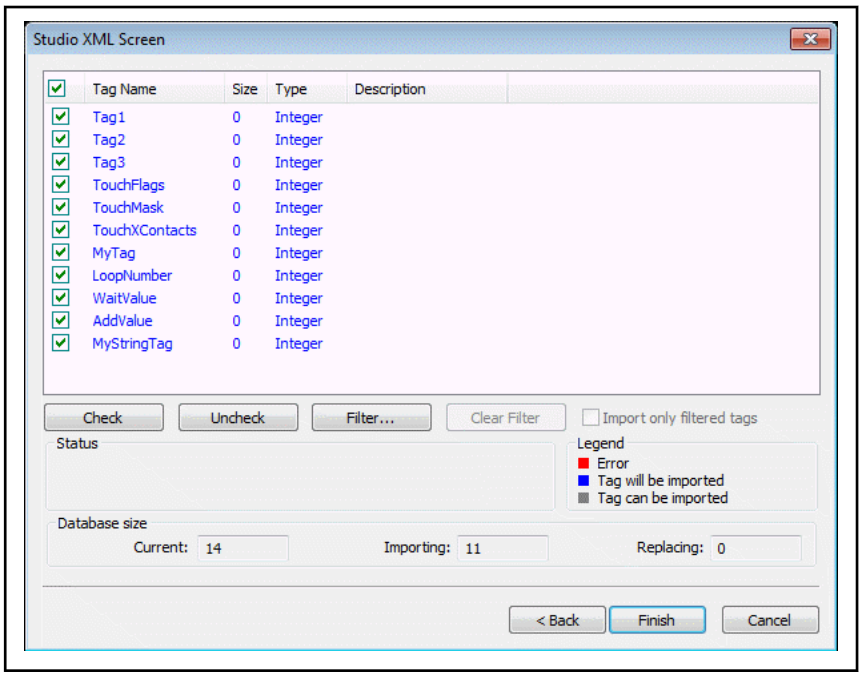


Fig. 8-149: Selection of the WinStudio variables to be imported.

7. Select the variables to import together with the screens from the list:
   - Individual variables can be selected or deselected with the help of the checkbox.
   - If all variables are to be selected or deselected, use the **Check** or **Uncheck** buttons.
   - To filter the tag list, click on **Filter** and configure filter expressions for one or several columns. It is possible to use placeholder characters (* and ?) in the filter expressions.
8. Click **Finish**.

The screens and the corresponding tags are imported into your project. In addition, the screens are automatically published for Thin Clients and Mobile Access — i.e. the corresponding `*.HTML`, `*.SCC` and `*.SSMA` files are automatically generated and saved in the web folder of your project so that the imported screens can be selected immediately.

9 Alarms, events and trends

The Alarm and Trend tasks are used to log historical data, and the Alarm/event and Trend objects are used to display historical data on a project screen.

These two functions are usually used in combination, however, they do not need to be.

Project data may be logged without being displayed at runtime and the data displayed at runtime may be taken from outside the project.

9.1 Alarm worksheet

The Alarms folder enables the configuration of alarm groups and tags related to each group. The Alarm worksheet defines the alarm messages generated by the application. The primary purpose of an alarm is to inform the operator of any problems or abnormal condition during the process so he can take corrective action(s).

The Alarm worksheet is executed by the background task module (refer to Execution tasks, page 91). It manages the state of all alarms and saves the alarm messages to the history if configured. The Alarm worksheet itself, however, does not display alarm messages at runtime. To display alarms, the Screen object “Alarm/event control” on page 331 in the "WinStudio screen tools" tool window has to be inserted into a WinStudio screen and configured.

To create a new worksheet, open the context menu of the "Tasks" folder and select **Add ▸ Alarm**.

To edit an existing alarm worksheet, double-click the element in the Project Explorer.

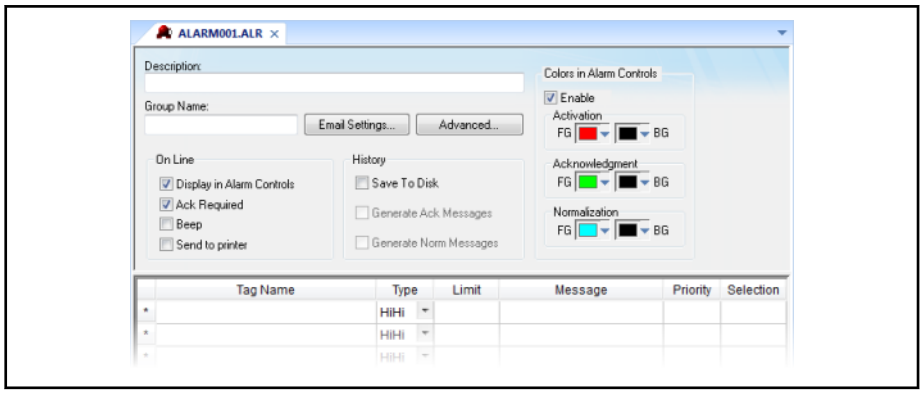


Fig. 9-1: Alarm worksheet

Multiple alarm groups (worksheets) can be created and each group can be configured with independent settings, such as message colors, history log enabled/disabled, etc.

Each alarm worksheet is composed of two areas:

- **Header:** Settings applied to all tags and alarms configured in the same alarm group. These settings enable configuration of the formatting of the messages and actions that have to be triggered based on alarm events (e.g., print alarms, send alarms by email, etc.). For further information, refer to header settings on page 314.
- **Body:** Selecting tags and defining conditions on which alarm messages are created. For further information, refer to body settings on page 319.

- Alarm groups can be configured to send notifications by email automatically, based on alarm events. For further information, refer to email settings on page 315.
- The alarm properties associated with each tag (configured in the body of the alarm group) can also be edited on the Tag properties on page 150 dialog (**Properties** in the "Cross references" tool window). However, before associating a tag with an alarm group, it is necessary to create the alarm group and to configure the settings on its header. These will be applied to all tags associated with the group.
- As of "WinStudio" v6.1+SP2, the Alarm task has been modified to avoid automatically acknowledging alarms by another alarm. For example, the Hi (Lo) alarm is not to be automatically acknowledged when the HiHi (LoLo) alarm becomes active. To enable the this behavior, set the following key in the project file (`.APP`):

```
[Alarm]
UseLegacyPriorityAck=1
```

The settings configured in the body of each Alarm worksheet are stored in the Tags Database archive(s). Therefore, changes to the tags database may affect the content of the alarm worksheets (body).

Notice that each tag/type cannot be available in more than one alarm group simultaneously because the alarm group is a property associated to each tag/alarm type (e.g., `tag: Level; alarm type: Hi; alarm group: 2`).

9.1.1 Alarm worksheet header

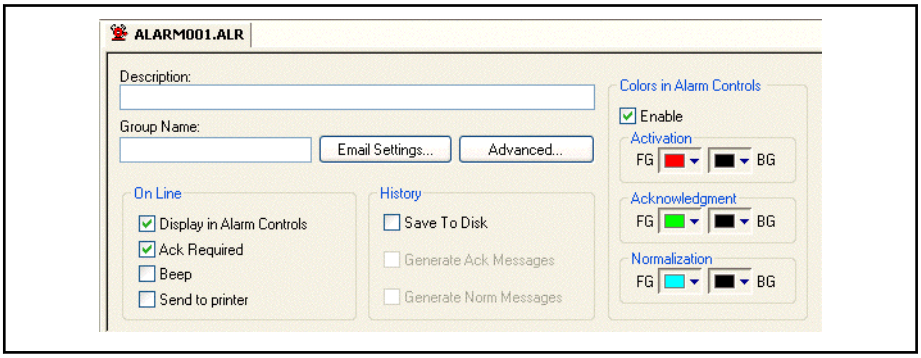


Fig. 9-2: Header settings on an alarm worksheet

The following table describes the header settings on an alarm worksheet (refer to Alarm worksheet on page 313):

Field	Description	Syntax
Description	Description of the alarm group. It is displayed on the workspace. This field is used for documentation only.	Text (up to 80 characters)
Group Name	Name of the alarm group. At runtime, the operator can filter alarms based on the group name by the built-in filter dialog of the “Alarm/event control” object.	Text (up to 32 characters)

Field	Description	Syntax
Email settings	Opens the Email settings on page 315 dialog using which the automatic sending of email notifications in the case of alarms can be configured.	Button
Advanced	Opens the Advanced settings on page 318, in which the settings for the automatic sending of email notifications in the case of alarms can be configured.	Button
Online > Display in alarm object	If this option is enabled, the alarms are available to be displayed on the Alarm/event control object on page 331.	Checkbox
Online > Ack required	If this option is enabled, the alarms require acknowledgment. In this case, the alarms are displayed on the Alarm/event control object on page 331 (Online mode) until they are acknowledged AND normalized.	Checkbox
Online > Beep	If this option is enabled, the computer keeps beeping, if there are active alarms to be acknowledged.	Checkbox

Online > Send to printer	If this option is enabled, the alarm messages are sent to the printer as soon as the alarm event occurs. This option requires a matrix printer (instead of an inkjet or laser printer) in order to print the message(s) and it hast to be fed by just one line, otherwise, each alarm is printed on a different sheet of paper. The alarms are printed on the default printer. If alarms are to be sent to a printer different from the default printer, the printer path/name can be specified by editing the following parameter in the `project_name.APP` file:
	`[AlarmLog]`
	`Device=PrinterPath/PrinterName`
		Checkbox

History > Save to disk	If this option is enabled, the alarm messages are stored in the History log on page 324 when they become active.	Checkbox
History > Generate ack message	If this option is enabled, the alarm messages are stored in the History log on page 324 when they are acknowledged.	Checkbox
History > Generate norm. Messages	If this option is enabled, the alarm messages are stored in the History log on page 324 when they are normalized.	Checkbox
Colors in alarm controls > Enable	If this option is enabled, the alarms configured in this group are be displayed with the colors assigned to each alarm state (**Activation**, **Acknowledgement** or **Normalization**), according to the colors configured in the alarm group.	Colors on page 277
Colors in alarm controls > FG and BG	Text foreground colors (FG) and background colors (BG) for the alarms displayed can be configured on the Alarms/Events control object on page 331. Each alarm state can be displayed with a different color scheme: • **Activation**: Alarm active and not acknowledged • **Acknowledge**: Alarm active and acknowledged • **Normalization**: Alarm no longer active and not acknowledged.	Colors on page 277

Email settings for alarm worksheet

"WinStudio" offers the function to send emails automatically when alarm events occur. Emails are sent using the standard SMTP (Simple Message Transfer Protocol). Therefore, only a valid email account with an SMTP and

POP3 server is required. It is not necessary to install any additional software, such as Microsoft Outlook.



Before emails can be sent, it is necessary to execute the CnfE-mail on page 865 function at least once successfully (from the internal WinStudio language). This function sets the email account parameters used when sending emails from the project (e.g., SMTP server, user name, password, etc.).

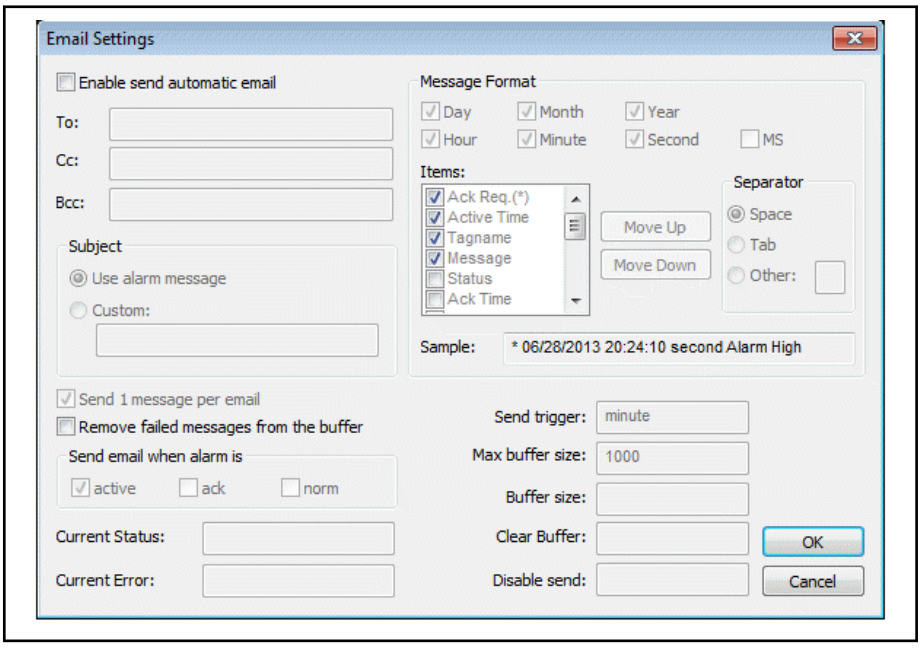


Fig. 9-3: Alarms worksheet – Email settings

Field	Information	Syntax
Enable send automatic email	Name of the tag associated with the alarm.	Checkbox
To, Cc, Bcc	Target addresses for the emails. Multiple email addresses can be configured in each box (To, Cc and/or Bcc) by separating the addresses with a semi-colon (";").	Text and/or {Tag} (up to 1024 chars)
Subject	When selecting "Use alarm message", the alarm message itself is used as the subject of the email to be sent. When selecting "Custom", a custom text can be configured as subject when sending the alarm.	Radio button/text (up to 1024 chars)
Send 1 message per email	If this option is enabled, each alarm is sent in an individual email and all emails are sent when the **Send trigger** is triggered. Otherwise, all alarm messages are buffered and sent in only one email when the send trigger is triggered. This option cannot be disabled (unchecked) when "Use alarm message" is configured under subject.	Checkbox
Remove failed messages from the buffer	If this option is enabled, emails are removed from the buffer after attempting to send them even if there was an error (failure) and the email was not sent. Otherwise, the messages are kept in the buffer until they are sent successfully or when the buffer reaches its maximum size.	Checkbox

Field	Information	Syntax
Send email when alarm is	Enables configuration of which alarm events are to generate emails: Please note that each event can be enabled/disabled individually. • **Active**: When the alarm becomes active. • **Ack**: When the alarm is acknowledged. • **Norm**: When the alarm is normalized.	Checkbox
Current status	The tag configured in this field, if any, is updated with the current status of the current or last email that the project attempted to send: • -2: Incorrect version of the INDMail.DLL library. • -1: The INDMail.DLL library is corrupted. • 0: The function SendEmailExt on page 869 is not executed. • 1: Sending email(s) • 2: Last email was sent successfully. • 3: There was an error sending the last email.	Tag
Current error	The tag configured in this field, if any, is updated with the error message describing the result of the last email that the project attempted to send. Therefore, when configuring a tag in this field, this tag has to be a String type.	Tag
Message format	This interface enables configuration of the actual format of the message sent by email, based on the alarm event(s): When configuring these settings, the **Sample** field displays an example of the format of the message according to the settings being configured. • **Day**, **Month**, **Year**, **Hour**, **Minute**, **Second**, **MS**: The enabled options define the timestamp for the alarm messages. MS stands for milliseconds. • **Items**: The enabled options define the email message for each alarm. The order of the items can be configured by using the Move Up and Move Down buttons. • **Delimiters**: Enables choosing the separator used between the items checked in this interface.	Checkbox and radio button
Send trigger	When alarm events are generated, they are kept in an internal buffer (memory). If the tag configured in this field changes its value, the email(s) on the internal buffer are sent to the addresses configured in the **To**, **Cc** and **Bcc** fields. After being successfully sent, the emails are removed from the internal buffer.	Tag
Max buffer size	Maximum number of alarm messages (events) that can be stored in the internal buffer simultaneously. If this limit is reached, the buffer follows a FIFO (First In, First Out) principle, deleting older messages as soon as newer messages are generated. This way, it is guaranteed that the buffer does not exceed the limit configured in this field.	Tag or number

Field	Information	Syntax
Buffer size	The tag configured in this field, if any, is updated with the number of messages (events) currently stored in the internal buffer.	Tag
Clear buffer	If the tag configured in this field changes its value, all messages (events) currently stored in the buffer are deleted. These messages are never sent.	Tag
Disable send	If the value of the tag configured in this field is TRUE, the Email function is temporarily disabled. Alarm events generated while the Email function is disabled are not stored in the internal buffer. Also, emails will NOT be sent in this condition, even if the tag configured in the field **Send trigger** changes its value.	Tag

Advanced settings for alarm worksheet

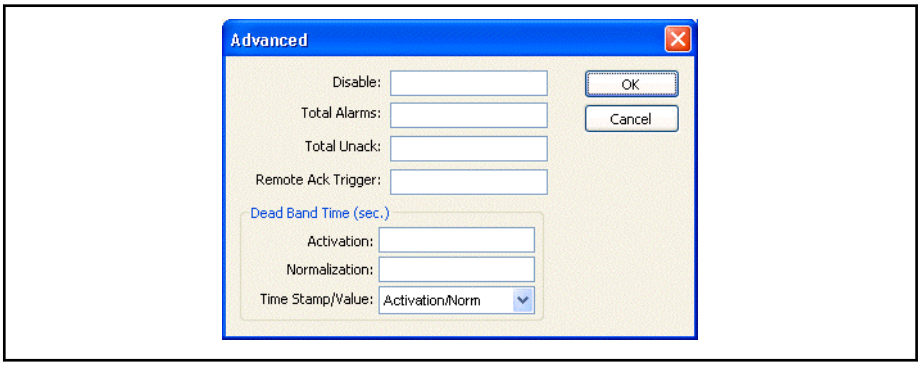


Fig. 9-4: Alarm worksheet, "Advanced" dialog

The following table describes the advanced settings on an Alarm worksheet on page 313:

Field	Description	Syntax
Disable	When the value of the tag configured here is TRUE, all alarms configured in this group are temporarily disabled. This option is useful to disable alarms under special conditions (e.g., during maintenance).	Tag
Total alarms	The tag configured in this field, if any, is updated with the number of alarms from this group, which are currently active.	Tag
Total unack	The tag configured in this field, if any, is updated with the number of alarms from this group, which are currently active AND have not been acknowledged yet.	Tag
Remote ack trigger	If the tag configured in this field changes its value, all active alarms from this group are acknowledged. This option can be used to acknowledge alarms regardless of any action from the operator.	Tag
Dead band time > activation	Each alarm has to remain continuously in its alarm condition for the period of time specified in this field before becoming active. This option is useful to generate alarms on intermittent conditions (e.g., noise). If this field is left in blank, the alarm becomes active as soon as its condition is true.	Tag or number

Field	Description	Syntax
Dead band time > normalization	Each alarm has to remain continuously out of its alarm condition for the period of time specified in this field before becoming normalized. This option is useful to normalize alarms on intermittent conditions (e.g., noise). If this field is left in blank, the alarm become normalized as soon as its condition is no longer true.	Tag or number
Dead band time > time stamp/value	Each alarm maintains a time stamp of the last significant activity, along with the value of the tag at that time. The type of activity that updates the time stamp: • **Activation/Standard** (default): The time when the dead band ended, i.e., when the alarm becomes activated or normalized. • **Last tag change**: The time when the value of the tag last changed during the dead band. • **Start condition**: The time when the dead band started.	Combo box

9.1.2 Alarm worksheet body

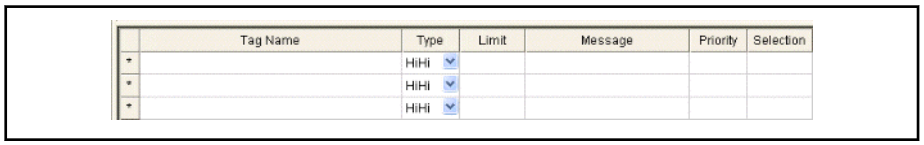


Fig. 9-5: Alarm worksheet body

The following table describes the body settings on an Alarm worksheet on page 313:

☞ No more than one alarm of the same type can be configured on the same tag.

Field	Description	Syntax
Tag name	Name of the tag associated with the alarm.	Tag
Type	Type of the alarm: When using the types Rate, DevP and DevM, it is necessary to configure additional settings in the Tag Properties dialog (**Tag Properties** in the "Cross references" tool window). • **HiHi**: Activates the alarm if the tag value is equal or higher than the limit. • **Hi**: Activates the alarm if the tag value is equal or higher than the limit. (For Boolean tags, if the value is 1.) • **Lo**: Activates the alarm if the tag value is equal or lower than the limit. (For Boolean tags, if the value is 0) • **LoLo**: Activates the alarm if the tag value is equal or lower than the limit. • **Rate**: Activates the alarm if the tag value varies faster than the rate specified to the alarm. (For Boolean tags, if the value changes.) • **Dev+**: Activates the alarm if the tag value is equal or higher than the command value plus the limit. • **Dev-**: Activates the alarm if the tag value is equal or lower than the command value minus the limit.	Combo box
Limit	Limit associated with each alarm. The limits can be modified dynamically at runtime by the tag fields HiHiLimit, HiLimit, LoLimit, LoLoLimit, Rate, DevP and DevM (e.g., TagLevel->HiLimit).	Count
Message	Message associated with the alarm. The message can be displayed on the Alarm/event control object on page 331 and/or stored in the alarm history on page 324 and/or sent by email on page 315, depending on the settings configured in the header on page 314 of the alarm group.	Text and/or {Tag} (up to 256 characters)
Priority	Priority number associated with the alarm. When displaying alarms on the Alarm/event control object on page 331, the operator can filter and/or sort the alarms by priority.	Number (from 0 to 255)
Selection	Alias associated with the alarm (e.g., AreaA, AreaB, etc). When displaying alarms on the Alarm/event control object on page 331, the operator can filter and/or sort the alarms by their selection value.	Text (up to 7 characters)

9.1.3 Sorting or filtering the lines of a worksheet

Sort or filter the lines of a worksheet to facilitate searching the lines or finding a particular element.

Before starting this task, insert a worksheet and open it for editing. In addition it should be clear how sorting and filtering works in general spreadsheet applications.

Please note that the lines can only by sorted and filtered in the following types of worksheets:

In Project tags, Shared tags and System tags data sheets;

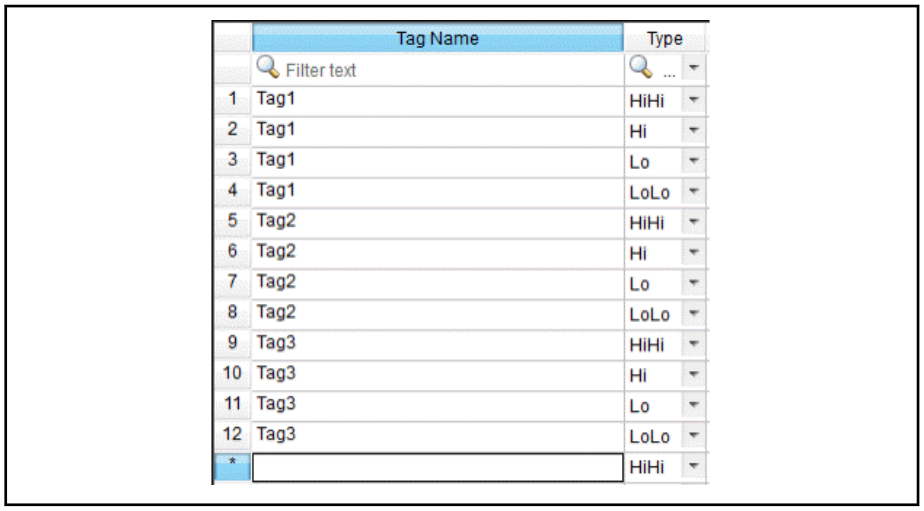
In the Translation table worksheet;

In all Task worksheets except for report and script which do not have lines;

In all Communication worksheets.

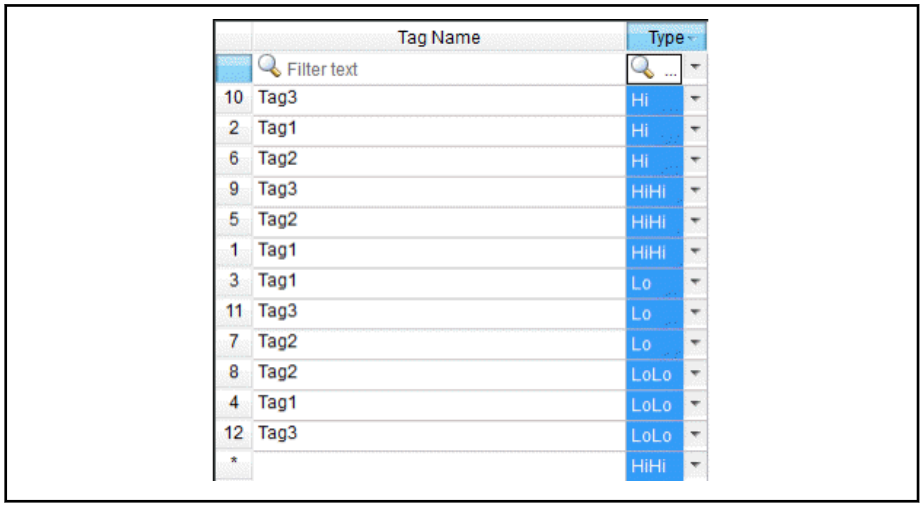
The other worksheets do not have lines to sort or filter.

The worksheets are sorted alphanumerically by selected column either in ascending ((0-9, A-Z) or descending order (Z-A, 9-0).



	Tag Name	Type
	Filter text	...
1	Tag1	HiHi
2	Tag1	Hi
3	Tag1	Lo
4	Tag1	LoLo
5	Tag2	HiHi
6	Tag2	Hi
7	Tag2	Lo
8	Tag2	LoLo
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

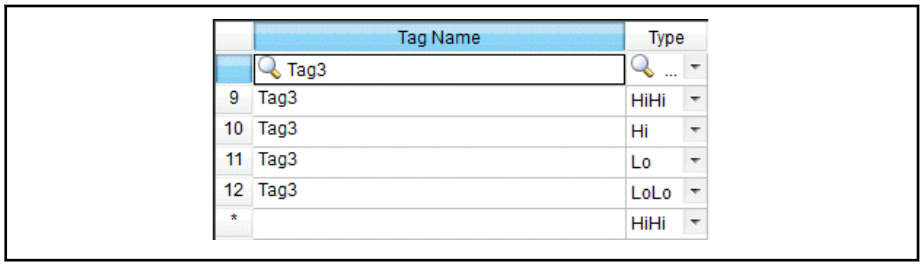
Fig. 9-6: Alarm worksheet lines in their original order



	Tag Name	Type
	Filter text	...
10	Tag3	Hi
2	Tag1	Hi
6	Tag2	Hi
9	Tag3	HiHi
5	Tag2	HiHi
1	Tag1	HiHi
3	Tag1	Lo
11	Tag3	Lo
7	Tag2	Lo
8	Tag2	LoLo
4	Tag1	LoLo
12	Tag3	LoLo
*		HiHi

Fig. 9-7: Alarm worksheet lines sorted by type

The worksheets are filtered according to the string entered in the selected column. Only lines that match the string will be displayed.



	Tag Name	Type
	Tag3	...
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

Fig. 9-8: Alarm worksheet lines filtered by tag name is "Tag3"

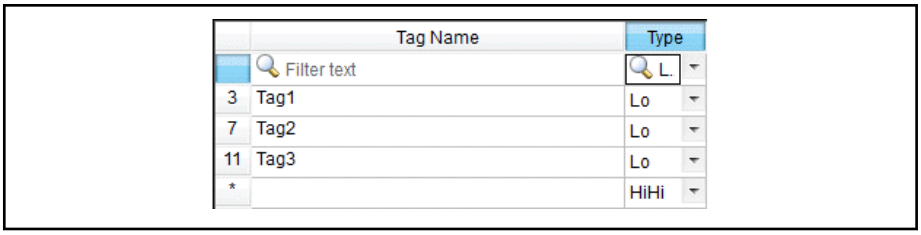


Fig. 9-9: Alarm worksheet lines filtered by type is "Lo"

It is still possible to delete lines while they are sorted or filtered.

Sorting or filtering lines:

1. To sort the lines, click on the column header by which you would like to sort. Click once to sort in ascending order and again to sort in descending order. The current order (i.e. the sorting direction) is indicated by the arrow to the right of the column name.

It is not possible to sort by several columns.

2. To cancel the sorting and restore the original order of the lines, click on the header of the first (numbered) column.
3. To filter the lines, enter the sting by which you would like to filter in the top line (zero) of the worksheet and press either **Tab** or **Enter**.

   *It is possible to use * and ? as wildcard characters in your string:*

   - * stands for any number of characters or no character. Tag* for example corresponds to `Tag`, `Tag3`, `Tag34567`, `TagA` and `Tag_TEMP`.
   - ? stands for exactly one character. `Tag?`, for example, corresponds to `Tag3` and `TagA`, while `Tag?????` corresponds to `Tag34567` and `Tag_TEMP`.

   It is possible to filter several columns. Only lines that match the filter string in all columns will be displayed.
4. To cancel filtering and restore the original order of the lines, delete the string you entered and either press **Tab** or **Enter**.

Please note that sorting or filtering the lines of a worksheet only facilitates the editing of this worksheet. How the worksheet is executed during runtime does not change. The lines are executed in their original order according to the numbering (column on the left), unless a line is moved or deleted.

9.1.4 Saving the alarm history to an external database

By default, the alarm history and event log are saved to proprietary-format text files in the “Alarms” folder of the project. However, the project settings can be modified to save them to an external SQL database instead.

If your project was created with Rexroth WinStudio v7.3+SP2 or earlier and then upgraded to the current version, you should consider starting over with new database tables.

Beginning with Rexroth WinStudio v7.3+SP3, new database tables are automatically indexed by event time in order to improve runtime performance. Existing database tables cannot be indexed in this way, so if you can afford to discard that data, you should change your database configuration to create new tables.

If you do this, you must also manually edit your project file (projectname.APP) to add the following setting:

```
[Alarm]
AddEventTimeColumn=1
```

The setting exists for backward compatibility, and it defaults to 0 for projects that were upgraded from earlier versions.

1. Open the **Options** tab in the "Project settings" dialog.

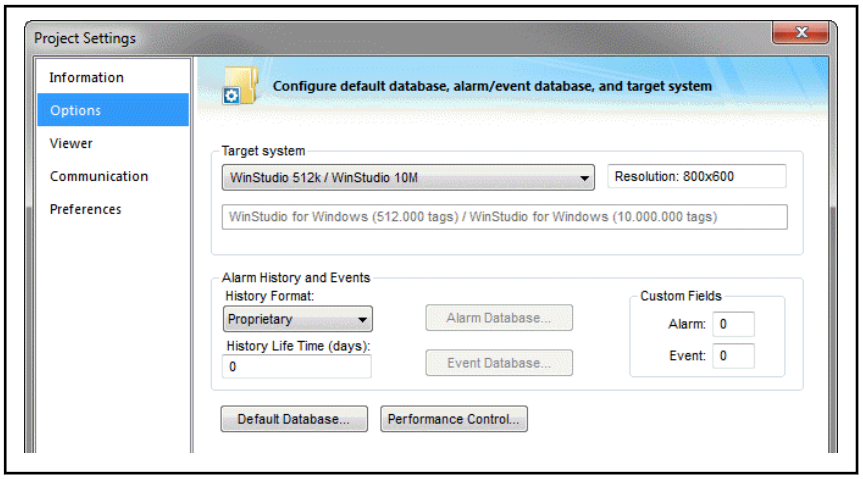


Fig. 9-10: Project settings: Options

2. In the **Alarm history and events** pane, in the **History life time (days)** input field, enter the number of days of history that are to be saved. As the history exceeds the specified number of days, it is automatically deleted in a first-in, first-out manner. If no number is specified - that is, if it is left blank or set to 0, - the history is never deleted. There is no limit on how much history can be saved, however, the more is saved, the more disk space it takes.
3. From the **History format** list, select **Database**.
4. To configure a single, default database to be used for both the alarm history and the event log (as well as all other runtime tasks), click **Configure** in the **Default database** pane. The "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For further information, refer to Configuring a default database for all task histories on page 567.
5. To configure a separate database, either for the event log or the alarm history, click **Event database** or **Alarm database**. In either case, a "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For more information, refer to Default database configuration on page 563.
6. Click on **OK**.

9.1.5 Format of the alarm history

The location and format of the alarm history depends on whether the History Format is set to Proprietary or Database. This section describes both.

If the history format is proprietary (default), the alarm history is saved as a series of text files in the project “Alarm” folder, with one file per calendar day. The name of each file is `Alyymmdd.ALH`, where:

- `yy` is the last two digits of the year in which the alarm history file was generated;
- `mm` is the month in which the alarm history file was generated; and
- `dd` is the day of the month on which the alarm history file was generated.

Therefore, the alarm history file for 07 May 2003 can be found at

`[…]\Documents\WinStudio v7.4 Projects\projectname\Alarm \AL030507.ALH`.



To change the location where these files are saved, use the function SetAppAlarmPath on page 800.

Within a `*.ALH` file for a specific day, each alarm is saved as a new line using the pipe character (|) to delimit the fields as illustrated below:

**.ALH file*

```
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13|P14|P15|P16|P17|P18|P19|P20|P21|P22|P23
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13|P14|P15|P16|P17|P18|P19|P20|P21|P22|P23
.
.
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13|P14|P15|P16|P17|P18|P19|P20|P21|P22|P23
```

If the history format is set to database, the alarm history is saved as a table in the database that is configured as alarm database. For further information, refer to Saving the alarm history to an external database on page 322.

The fields/columns of the alarm history are described below:

Proprietary	Database		Description	File vers.
Field number	Column name	Data type		
P1	—	—	File version (current = 003)	001
P2	AI_Start_Time	Time stamp	Start date (MM/DD/YYYY)	001
P3			Start time (HH:MM:SS)	001
P4	AI_Tag	String	Tag name	001
P5	AI_Message	String	Alarm message	001
P6	AI_Ack	Boolean	Ack, where: • **0**: Alarm was acknowledged or does not require acknowledgment • **1**: Alarm was not acknowledged	001
P7	AI_Active	Boolean	Active, where: • **0**: Alarm is not active • **1**: Alarm is active	001
P8	AI_Tag_Value	Real	Tag value when the alarm occurred	001

Proprietary	Database		Description	File vers.
Field number	Column name	Data type		
P9	Al_Group	Integer	Alarm group number	001
P10	Al_Priority	Integer	Priority number	001
P11	Al_Selection	String	Selection	001
P12	Al_Type	Integer	Type, where: • **1** is HiHi • **2** is Hi(On) • **4** is Lo(Off) • **8** is LoLo • **16** is Rate(Change) • **32** is Deviation+ • **64** is Deviation-	001
P13	Al_Ack_Req	Boolean	Ack required, where: • **0**: Alarm requires acknowledgement • **1**: Alarm does not require acknowledgement	001
P14	Al_Norm_Time	Time stamp	Normalization date (MM/DD/YYYY)	001
P15			Normalization time (HH:MM:SS)	001
P16	Al_Ack_Time	Time stamp	Ack date (MM/DD/YYYY)	001
P17			Ack time (HH:MM:SS)	001
P18	Al_User	String	User name	002
P19	Al_User_Comment	String	Comment	002
P20	Al_User_Full	String	User full name	003
P21	Al_Station	String	Station	003
P22	Al_Prev_Tag_Value	Real	Previous value	003
P23	Bias	Integer	Time zone bias	003
—	Al_Start_Time_ms	Integer	Number of milliseconds for the starting time of the time stamp. This field is used if the database does not support ms in a time stamp field.	003
—	Al_Norm_Time_ms	Integer	Number of milliseconds for the normalization time of the time stamp. This field is used if the database does not support ms in a time stamp field.	003
—	Al_Ack_Time_ms	Integer	Number of milliseconds for the acknowledgement time of the time stamp. This field is used if the database does not support ms in a time stamp field.	003

Proprietary	Database		Description	File vers.
Field number	Column name	Data type		
—	Al_Deleted	Boolean	Deleted, where: • 0: Alarm message was not deleted by the user (not visible). • 1: Alarm message was deleted by the user (visible).	003
—	Last_Update	Time stamp	Time stamp of the last update for this alarm.	003
—	Last_Update_ms	Integer	Number of milliseconds for the "Last_Update" time stamp. This field is used if the database does not support ms in a time stamp field.	003



It is possible to customize the names of the columns in the database table by manually editing the project file (e.g., `[…]\Documents\WinStudio v7.4 Projects\projectname\projectname.APP`) as follows:

```
[Alarm]
DefaultName=NewName
```

Example:

```
[Alarm]
Message=Alarm_Message
Ack=Acknowledgement
```

9.2 Events and event logs

9.2.1 Events and event logs, general information

This section describes the logging and event-retrieval features of "WinStudio". An event can be any tag change, generating reports or recipes, opening and closing screens, logging onto and logging off the security system, and so forth. "WinStudio" saves all of these events in a log file, which can then be retrieved by the Alarm/event control object.

9.2.2 Enabling the event logger

Event logging is disabled by default to conserve runtime resources. To enable saving of events to the history file, use the event logger in the Project Explorer.

1. Open the context menu of the "WinStudio" folder and select the entry **Tools ▸ Event logger**. The "Event logger" dialog is displayed.

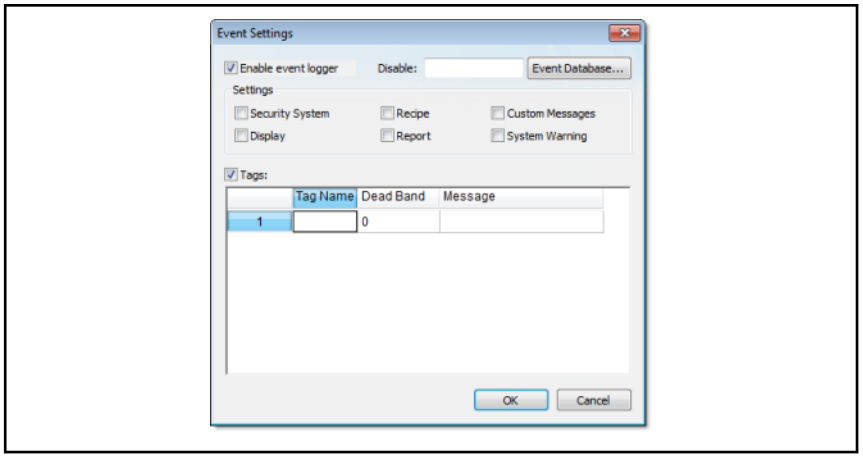


Fig. 9-11: "Event settings" dialog

2. Select **Enable event logger**.
3. In the **Disable** field, type the name of an application tag. If the value of the tag is TRUE (i.e., not zero) at runtime, event logging is suspended.
4. In the **Settings** pane, select the types of events that are to be logged to the history file.

Option	Description
Security system	Events generated by the project's security system, including: User logged on/off User created/removed by calling the functions CreateUser on page 678 or RemoveUser on page 687: User blocked/unblocked by calling the function BlockUser on page 675 or UnblockUser on page 689: User blocked by the security system after several attempts to enter an invalid password Password expired Password changed Invalid log on attempt
Display	Open screen and close screen events.
Recipe	Recipes loaded, saved, initialized or deleted.
Report	Reports saved to disk or sent to printer.
Custom Messages	Events created by calling the function SendEvent on page 894:
System warning	Various runtime warnings and errors, including: Errors that occur when sending alarms by email Tag was blocked/unblocked Division by zero Connection/disconnection of the remote security system

5. To log changes in certain application variables, activate the **Tags** pane and specify the variables in the table.

   The **Tags** option is useful for logging events that are not sufficiently important to be alarm conditions (for example, "Motor on", "Motor off", etc.).

Column	Description
Tag name	The name of the application variable to be logged in the history file.
Dead band	A value to filter changes against, so that only changes greater than this value are logged (dead band). For example, if a dead band of 5 is defined for a tag value of 50 that changes to 52, the system does not register this change in the event log as the change is less than 5. However, if the tag value change is at least 5, the system logs the new value in the history file.
Message	A string (message) related to this tag change. Tags can be specified in messages using the **{tagname}** syntax.

The **Tags** option is useful for logging events that are not sufficiently important to be alarm conditions (for example, Motor On, Motor Off, etc.).

6. Click on **OK**.

By default, the event log is saved as a series of text files in the project alarms folder. For further information, refer to Format of the event log on page 329.

Alternatively, the event log can be saved to an external SQL database. For further information, refer to Saving the event log to an external database on page 322.

9.2.3 Saving the event log to an external database

By default, the alarm history and event log are saved to proprietary-format text files in the “Alarms” folder of the project. However, the project settings can be modified to save them to an external SQL database instead.



If your project was created with Rexroth WinStudio v7.3+SP2 or earlier and then upgraded to the current version, you should consider starting over with new database tables.

Beginning with Rexroth WinStudio v7.3+SP3, new database tables are automatically indexed by event time in order to improve runtime performance. Existing database tables cannot be indexed in this way, so if you can afford to discard that data, you should change your database configuration to create new tables.

If you do this, you must also manually edit your project file (projectname.APP) to add the following setting:

```
[Alarm]
AddEventTimeColumn=1
```

The setting exists for backward compatibility, and it defaults to 0 for projects that were upgraded from earlier versions.

1. Open the **Options** tab in the "Project settings" dialog.

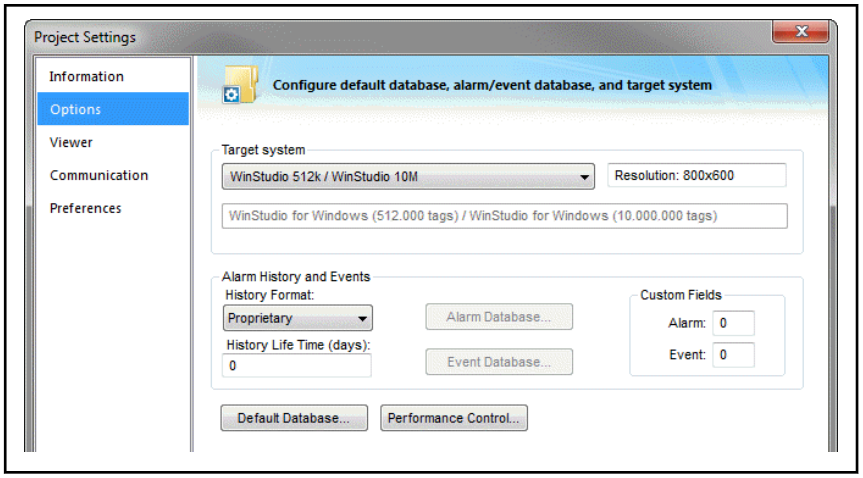


Fig. 9-12: Project settings: Options

2. In the **Alarm history and events** pane, in the **History life time (days)** input field, enter the number of days of history that are to be saved. As the history exceeds the specified number of days, it is automatically deleted in a first-in, first-out manner. If no number is specified - that is, if it is left blank or set to 0, - the history is never deleted. There is no limit on how much history can be saved, however, the more is saved, the more disk space it takes.
3. From the **History format** list, select **Database**.
4. To configure a single, default database to be used for both the alarm history and the event log (as well as all other runtime tasks), click **Configure** in the **Default database** pane. The "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For further information, refer to Configuring a default database for all task histories on page 567.
5. To configure a separate database, either for the event log or the alarm history, click **Event database** or **Alarm database**. In either case, a "Default database configuration" dialog is displayed. Configure the database connection using this dialog. For more information, refer to Default database configuration on page 563.
6. Click on **OK**.

9.2.4 Format of the event log

The location and format of the event log depend on whether the history format is set to proprietary or database. This section describes both.

If the history format is proprietary (default), the event log is saved as a series of text files in the project “Alarm” folder, with one file per calendar day. The name of each file is `EVyymmdd.EVT`, where:

- `yy` is the last two digits of the year in which the event log file was generated;
- `mm` is the month in which the event log file was generated; and
- `dd` is the day of the month on which the event log file was generated.

Therefore, the alarm history file for 7 May 2003 can be found at `[…]\Documents\WinStudio v7.4 Projects\projectname\Alarm\EV030507.EVT`.

To change the location where these files are saved, use the function SetAppAlarmPath on page 800.

Within a `*.EVT` file for a specific day, each event is saved as a new line using the pipe character (|) to delimit the fields, as illustrated below:

**.EVT*

```
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13
.
.
P1|P2|P3|P4|P5|P6|P7|P8|P9|P10|P11|P12|P13
```

If the history format is set to database, the event log is saved as a table in the database that is configured as event database. For further information, refer to Saving the event log to an external database on page322.

The fields/columns of the event log are described below:

Proprietary	Database		Description	
Field number	Column name	Data type		
P1	Version	Integer	This field is created only when the history format is proprietary. Current version: 002	
P2	Event_Type	Integer	1	Security system
			2	Display
			3	Recipe
			4	Report
			5	Custom message
			6	System warning
			7	Log tags
P3	Event_Time	Time stamp	Time stamp indicating when the event occurred. When the history format is set to proprietary, the project saves the event time in the following format: `MM/DD/YYYY HH:MM:SS.MSS`. If the history format is set to database, the project saves the event time in the default time stamp format of the database.	
P4	Event_Info	String	Tag name.	
P5	Value	Real	Tag value when the event occurred.	
P6	Source	String	Name of the task that generated the event.	
P7	User	String	User logged on when the event occurred.	
P8	User_Full	String	Full name of the user logged on when the event occurred.	
P9	Message	String	Event message.	
P10	Station	String	Name of the station (computer) where the event occurred.	
P11	Comment	String	Comment (optional) typed by the operator when the event occurred. This field is only available for Version >= 2.	
P12	Previous_Value	Real	Tag value that occurred before the event. This field is only available for Version >= 2.	
—	Deleted	Boolean	This field is created only if the history format is set to database. • 0 (FALSE): Event message was not deleted. • 1 (TRUE): Event message was deleted.	

Proprietary	Database		Description
Field number	Column name	Data type	
P13	Bias	Integer	Difference (in minutes) between Event_Time and the GMT time. This field is only available for Version >= 2.
—	Last_Update	Time stamp	Time stamp indicating when the register was created/modified. This field is used to synchronize the databases when using the secondary database in addition to the primary database. This field is created only if the history format is set to database.

The names of the columns in the database table can be individually adapted by manually editing the project file

(e.g. `[…]\My Documents\WinStudio v8.0 Projects\project_name\Web\Translation.trn`)

as follows:

```
[EventLogger]
DefaultName=NewName
```

Example:

```
[EventLogger]
Event_Info=Information
Message=Event_Message
```

9.3 Alarm/event control object

Use the **Alarm/event control** tool to add an alarm or event control object to an project screen.

To create and configure an alarm/event control object, proceed as follows:

1. In the "WinStudio screen tools" tool window, click on the **Alarm object** element.
2. Click in an open WinStudio screen and drag the mouse to create and adjust the shape of the object.
3. Double-click on the object to open the following "Object properties" dialog.

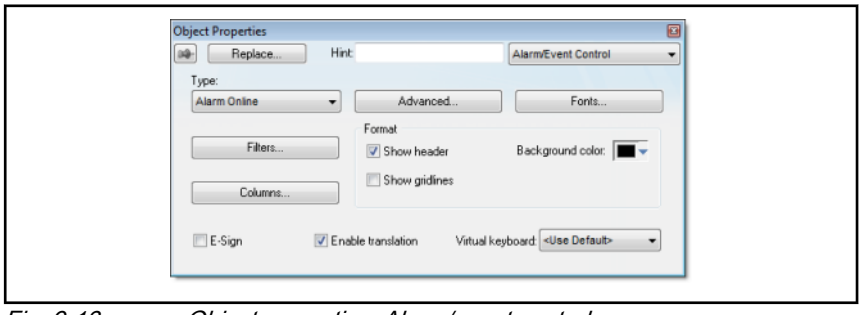


Fig. 9-13: Object properties: Alarm/event control

In this dialog, the following parameters can be specified:

- Select an alarm object mode in the "Type" pane:
  - **Alarm online**: Displays only current alarm messages.
  - **Alarm history**: Displays only alarm messages from the alarm history database.

  - **Alarm history + event**: Displays both alarm messages from the alarm history database and logged events from the event history database.
  - **Event**: Displays only logged events from the event history database.
- Click on (enable) the **Enable translation** checkbox to enable external translation of messages using the translation tool. (Refer to Translation tool on page 487 for further information.)
- **E-Sign** checkbox: If this option is enabled, a prompt is displayed to enter the electronic signature before executing the animation
- **Virtual keyboard**: Select the Virtual keyboard type used for this object. Before configuring the virtual keyboard for this interface, the "Virtual keyboard" option has to be selected in the "Viewer" settings (**Viewer** tab in the WinStudio Properties dialog).

Filter To filter alarm messages at runtime, click the **Filters** button. The "Filters" dialog is displayed to specify filtering parameters for the alarm control object.

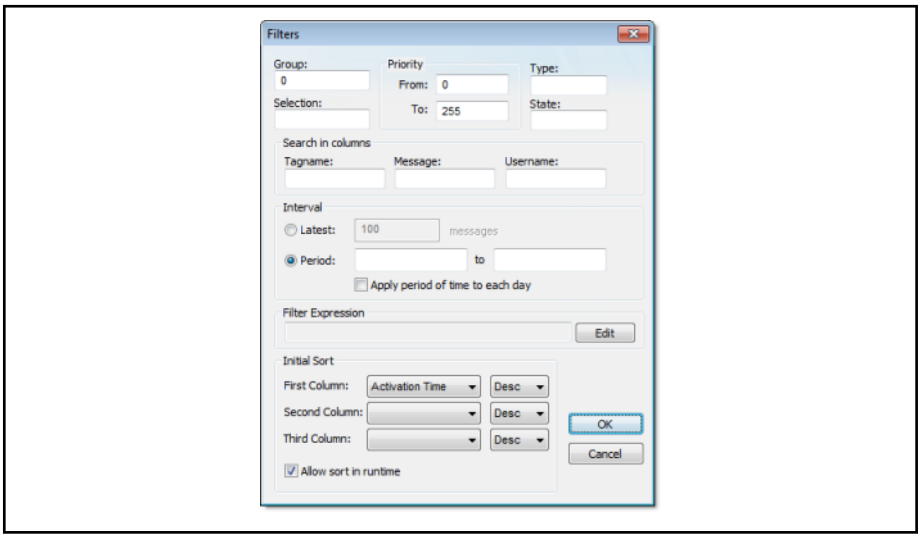


Fig. 9-14: "Filters" dialog

- Use the **Group** field to filter messages by the number of the alarm worksheet on page 313. The worksheets are organized in the folder "Alarms" under the WinStudio\Tasks folder, starting with 1. If 0 is specified for the group, all worksheets are displayed. Commas or dashes can be used to specify an array of groups, for example,**1,3,5-6**.
- Use the **Selection** field to filter messages by the selection text configured on the alarm worksheet.
- In the "Priority" pane, use the **From** and **To** fields to filter messages according to the priority configured on the alarm worksheet. Enter numerical values into these fields to limit the priority range.
- Use the **Type** field to filter messages by the alarm type (e.g., HiHi, Hi, Lo, LoLo, Rate, Dev+, Dev-). Commas can be used to specify more than one type; for example,"HiHi, LoLo".
- Use the **State** field to filter messages by the alarm status:

Value	Description
0	All alarms (default)
1	All active and unacknowledged alarms

Value	Description
2	All active and acknowledged alarms
3	All inactive and acknowledged alarms
4	All inactive and unacknowledged alarms

Leaving this field blank is effectively the same as entering a value of 0.

- In the "Search in columns" pane, use the **Tagname**, **Message**, and/or **Username** text fields to specify criteria for filtering messages. Enter a tag name, message, and/or user name into the text field to search for in "WinStudio".
- Use the parameters in the "Interval" pane to filter messages by the last **x** messages (**Latest**) or based on a period of time (**Period**). If no interval is specified, only the alarms for the current day are displayed.
- Use the "Filter expression" pane to configure an expression to filter unwanted messages out of the display. Only messages that fulfill the expression are shown. To enter an expression, click on the **Edit** button. The dialog "Alarm filter expression" is displayed.

☞
- String tags can be specified in curly brackets (e.g. {tagname}) in the **Group**, **Selection**, **Tagname**, **Message**, and **Username** fields to change these values at runtime.
- String tags have to be specified without curly brackets in the **Type** field and the **Period** fields of the Interval pane. In these fields, values cannot be entered directly.
- Integer tags can be specified in the **From** and **To** fields in the "Priority" pane and the **State** field, and the **Latest** field in the "Interval" pane.
- Placeholder characters (* and ?) can be used when specifying values for the **Selection**, **Tagname**, **Message**, and **Username** fields.

- Use the **Filter expression** pane to configure an expression to filter unwanted messages out of the display. Only messages that fulfill the expression are shown.

  To enter an expression, click on the **Edit** button. The dialog "Alarm filter expression" is displayed.

  The filter expression has to comply with the following basic syntax:

  ```
  [Column Name]ComparisonOperator'Value'
  ```

  where the **column name** is the name of a column in the “Alarm/event” control object.

  Example:

  ```
  [Activation Time]>'08/17/2007 15:00'
  ```

  This filter only displays alarm messages with activation times greater (later) than 15:00 on 08/17/2007.

☞
- The maximum number of characters is 1024 for engineering mode and 2048 for runtime mode.
- The **Display value** and **State** columns are not supported by the filter expression.



It is not necessary to use square brackets if the column name consists of only one word (e.g., Value). The brackets, however, make the filter expression more readable.

Several conditions can be combined simultaneously by using the logic operators AND, OR and NOT.

Example:

```
[Custom1]='MyCustomArea' AND [State]=0
```

```
[Type]='HiHi' OR [Type]='LoLo' AND [Activation Time]>'08/17/2007 15:00'
```

In the filter expression, placeholder characters (* and ?) can be used.

It is not necessary to use square brackets if the **column name** consists of only one word (e.g., **Value**).

The filter expression can be changed at runtime by specifying String tags in curly brackets.

Example:

```
[Value]='{AlarmFilterValue}'
```

```
[ActivationTime]<='{DateTime2UTC(Date + " " + Time)}'
```

Please note that filtering based on time only works if the alarm/event control object is configured to display the alarm history. All times are to be indicated in Coordinated Universal Time since the alarm history is saved in this way. To switch between local time and UTC, use the Date and time functions, page 647.

The Filter expression alarm dialog box has a limit of 1024 characters. It is, however, possible to configure a filter expression containing up to 2048 characters by using the syntax with curly brackets described above. The content of the brackets is evaluated during runtime. This means that it is possible to specify two or more project tags (String type) which contain parts of the overall expression and then use operators AND to combine the parts:

```
{MyFilterExpression1} AND {MyFilterExpression2}
```

- Use the settings in the Initial sort group to set the default sorting order. Select a sort type from the **Column** combo box and select **Asc** or **Desc** to sort in ascending or descending order. Up to three levels of sorting can be configured.



If all three levels are configured with sort types other than activation time, the project automatically sorts on a fourth level according to activation time in descending order.

The type of this fourth-level sort cannot be changed, however, its default order can be toggled - from descending to ascending - by manually editing the project file (`project_name.app`) by changing the following setting:

```
[Objects]
```

```
DescendingAlarmListTime=TRUE | FALSE
```

`TRUE` sorts in descending order, `FALSE` sorts in ascending order. Please note that this setting only works for projects created with or updated to "Rexroth WinStudio v7.2" or later.

Click on the **Allow sort at runtime** checkbox to enable changing the sort order at runtime.

Columns Click on the **Columns** button to open the "Columns" dialog to specify display properties for columns in the object.

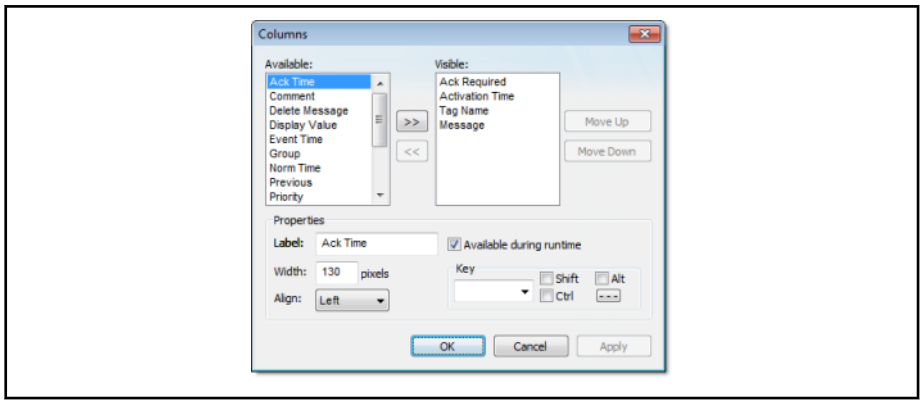


Fig. 9-15: "Columns" dialog

- The "Available" list contains all of the column types available for this object. The "Visible" list contains all of the column types currently in use for this object. Click the **»** and **«** buttons to move selections between the two lists.

Alarm objects can be configured to display recently replaced values together with their new values. To do so, move both **Value** and **Previous** to the "Visible" list. Click on the **Move up** or **Move down** buttons to rearrange the order of columns in the "Visible" list.

- Use the **Label** and **Width** fields in the "Properties" pane to change the default column labels and widths at runtime.
- Use the **Align** combo box to specify alignment (**Left**, **Center**, or **Right**) for the alarm message text within a specified column.
- Click on (enable) the **Available at runtime** checkbox to allow the user to add selected columns to the Visible list at runtime on page 331.
- Use the **Key** box to assign a shortcut to each column. This way, the information on the alarm control object can be sorted by any column, using keyboard keys instead of the mouse cursor.

To finish the settings, click **OK** to close the "Columns" dialog.

When acknowledging an alarm, the alarm control object sends a message to the alarm task on page 313 with the following information: **Tag Name**, **Type**, **User** and **Station**. This is a solution to control acknowledged alarms from a thin client.

It is possible to associate labels with priority values so that more important information is displayed in the priority column of the alarm control. Change your project file (projectname.APP) manually and add the following entries:

Syntax

```
[Alarm]
PriorityLabelCount=N (total number of labels)
PriorityValue1=value
PriorityLabel1=label
...
PriorityValueN=value
PriorityLabelN=label
```

Example:

```
[Alarm]
PriorityLabelCount=3
PriorityValue1=490
PriorityLabel1=ALMTest
PriorityValue2=480
PriorityLabel2=Test2
PriorityValue3=470
PriorityLabel3=Test100
```

Extended settings

Click the **Advanced** button to open the "Advanced" dialog to set advanced properties for the alarm object.

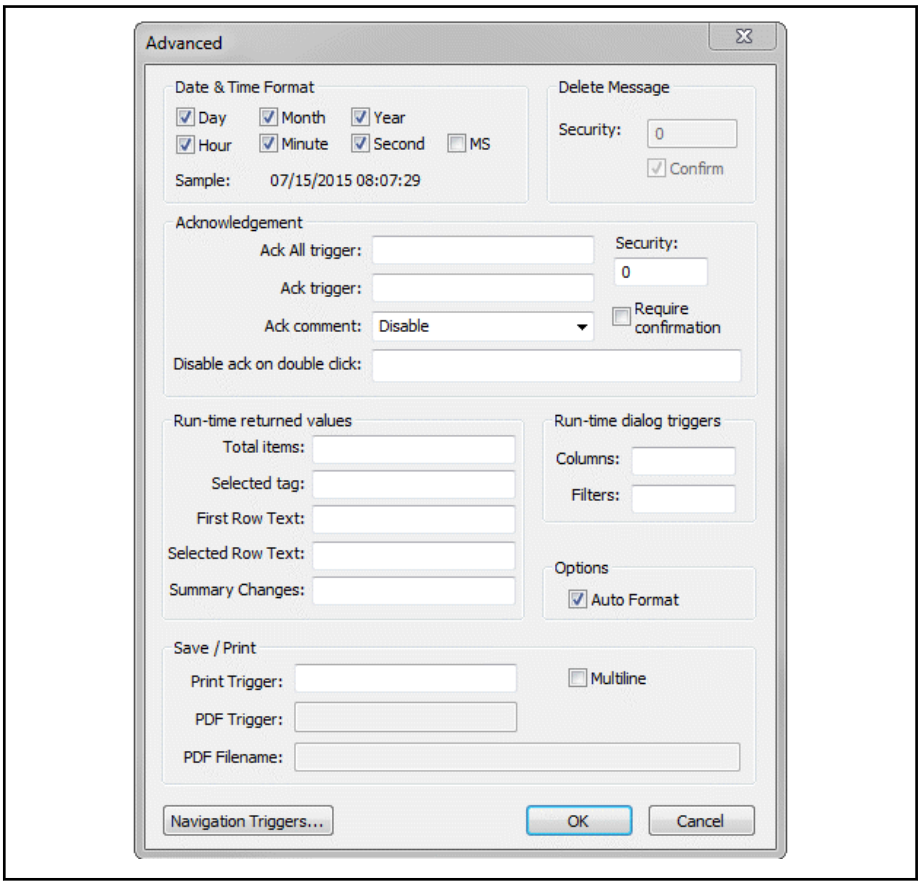


Fig. 9-16: "Advanced" dialog

- Use the parameters in the "Date & time format" pane to control which date and time information displays in the alarm message. Click on (enable) a checkbox to include that element in the display. Note: **MS** stands for milliseconds.



Please note the **Sample** text as preview how the information will look in the alarm message.

- Use the settings in the **Delete message** pane to control who is authorized to delete alarm messages from the alarm history:
  - **Security:** Use this field to specify which security level can delete alarm messages. Only users with the specified security level are allowed to delete an alarm message.
  - **Require confirmation**: Enable this option to require the user to confirm deleting a message before WinStudio actually deletes the selected alarm message.
- Use the parameters in the **Acknowledgement** pane to control how alarms are acknowledged.
  - **Ack all trigger** field: Enter a tag to receive a value. If the tag changes its value, it indicates that all messages in the alarm object have been acknowledged.
  - **Ack trigger** field: Enter a tag to receive a value. If the tag changes its value, it indicates that the message at the top of the alarm object has been acknowledged.
  - **Ack comment:** Select Disable, Optional or Required to determine if the user needs to or can enter comments after acknowledging alarms.
  - **Disable ack on double click**: Either enter a numerical value or a tag/expression. If it is evaluated as TRUE (not zero), the user cannot acknowledge alarms by a double-clicking (or double-tapping). This option can, for example, either be used to force the user to click on a different button to acknowledge the alarm or to prevent alarms from being acknowledged on Thin Clients.

    Security: field Enter a numeric value to specify which security level can acknowledge an alarm message. Only users with the specified level can respond.
  - **Require confirmation** checkbox: Click on (enable) this box to display a confirmation dialog when the user tries to acknowledge a single alarm.
- Use the settings of the **Run-time returned values** group to receive information on the alarms during runtime:
  - **Total items** field: Enter an Integer tag to see how many alarms remain after WinStudio filters the alarm objects using parameters specified in the Filters dialog box.
  - **Selected tag**: Enter a String tag to enable the end user to click on an alarm message to see the name of the tag associated with that message.
  - **First row text**: Enter the name of a project tag or array (String type). This tag or array receives the content of the columns of the first row of the alarm/event control. If a tag is specified, the columns are separated by tabs. If an array is specified, each array element receives a column. If the array is not big enough to receive all columns, the remaining columns are discarded. Whenever the first row changes – either due to a new alarms/events or simply because the rows are reordered – the specified array is updated.
  - **Selected row text**: Enter the name of a project tag or array (String type). This tag or array receives the content of the columns of the selected row of the alarm/event control (i.e. of the row the user has clicked/tapped). If a tag is specified, the columns are separated by

tabs. If an array is specified, each array element receives a column. If the array is not big enough to receive all columns, the remaining columns are discarded. If the selected row changes - i.e. if the user clicks/taps on a different row - the specified array is updated.

– **Summary changes**: Enter the name of a project tag (Integer type). This tag receives the running count of the number of changes in the Alarm/event control. For example, if a new alarm occurs or an alarm is acknowledged, the value of the configured tag is increased incrementally. Reordering the rows is not counted as a change.

- Use the settings in the **Run-time dialog triggers** group to enable the user to customize objects during runtime:
  – **Columns** field: Enter a tag to receive a value. If the tag changes its value, a dialog is opened for customization of the columns that are visible in the object.
  – **Filters** field: Enter a tag to receive a value. If the tag changes its value, a dialog is opened to filter the columns that are visible in the object.
- **Auto format**: When checked, decimal values in the **Display Value**, **Previous** and **Value** columns of the object will be formatted according to the virtual table created by the SetDecimalPoints, page 767 function.
- Use the **Save / Print** group to control the printing of alarms during runtime:
  – **Print Trigger**: If the tag configured in this field is toggled, the current state of the Alarm/event control object is sent to the default printer.
  – **PDF trigger** field: If the tag configured in this field is toggled, the current state of the Alarm/event control object is saved as a PDF file at the location specified by **PDF Filename**.
  – **PDF file name** field: Enter a complete file path and name where the PDF file is to be saved. Tag names using the **{tag}** syntax can also be entered.



PDF trigger and **PDF file name** are not supported in projects running on Windows Embedded or thin client.

  – **Multiline** checkbox: If this option is enabled, the print output or PDF is formatted according to the available column space, and the text within each cell is wrapped so that it is completely displayed.

- Click on the **Navigation triggers** button to open the following dialog:

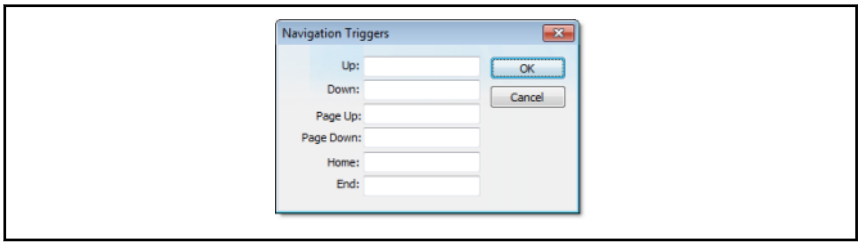


Fig. 9-17: “Navigation triggers” dialog

By configuration of tags in the respective fields, the functions scroll up, scroll down, page up, page down, home or end can be executed for displayed alarm objects. If the values of the configured tags change, the alarm control object navigates that way. This is useful for adding navigation control elements to a screen. For example, if the same tag is set to the **Up** field in this dialog as well as to a pushbutton object on page 197, the alarm control object scrolls up in case the pushbutton object is pressed.

Finish the settings by clicking on **OK** to close the "Advanced" dialog.

Fonts

Click on the **Fonts** button to open a standard fonts interface on page 280 to specify display properties for the message text.

Format

Use the Format pane in the object properties to configure the appearance of the alarm/event control object during runtime:

- Click on (enable) the **Show header** checkbox to display a header on the object. In the header, the column headers are displayed.

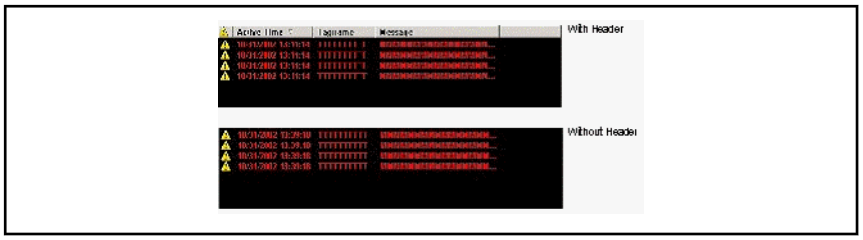


Fig. 9-18: Displaying a header

- Click on (enable) the **Show gridlines** checkbox to display gridlines in the object. Gridlines can simplify the differentiation between the individual rows and columns in the object.



Fig. 9-19: Displaying gridlines

- Use the **Background color** field to select a background color for the object. Click on the color box to open the color palette pop-up and simply click a color to select it.
- Click on **Colors** to open the Colors dialog box where colors for background and foreground of the individual alarms can be selected. These colors have the default colors which were selected in your alarm worksheet(s). In this way, special alarms can be highlighted.

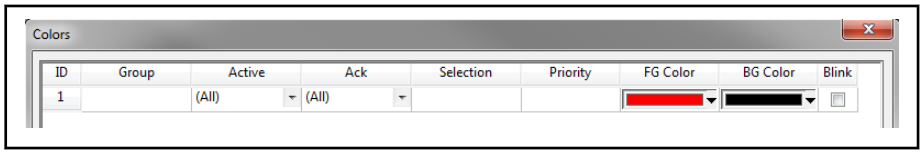


Fig. 9-20: Colors dialog box

This function is not supported in projects which run on Windows Embedded target systems.

In the first row of this table, a subset of alarms with similar criteria as in the Filters dialog box can be configured and individual background and foreground colors can be selected for this subset:

1. Enter the number of the alarm group/worksheet in the **Group** field. You can use commas and/or dashes to specify a range of groups.
2. In the **Active** field, either select **All** (active and normalized alarms), **Active** (active alarms only) or **Norm** (only normalized alarms).
3. In the **Ack** field, either select **All** (acknowledged and not acknowledged alarms), **Ack** (only acknowledged alarms) or **Not ack** (only not acknowledged alarms).
4. In the **Selection** field, enter the selection text you configured in the alarm worksheet. This field can remain empty.
5. In the **Priority** field, enter the priority number you configured in the alarm worksheet. This field can remain empty.
6. Click on the **BG color** field to open a color selection and select a color to be set as the background color of the alarm message.
7. Click on the **FG color** field to open a color selection and select a color to be set as the foreground color of the alarm message.
8. Select **Blink**, if the alarm message is supposed to flash. This makes it more visible.

Please note that the subsets which are configured here have to pass all filters which were configured earlier in the Filters dialog box. If the filters, for example, were configured to display the groups `1-6` in the alarm/event control object, configuring a subset of group 7 does not have any effect here.

In addition, the selected colors are used to display any possible alarm states (i.e., activation, acknowledgement and normalization) if you configure the subset to include all of these states. If different colors are to indicate different alarm states, additional subsets have to be configured.

9.4 Trend worksheet

Data sets to record data series can be configured in the "Trend" folder. They can be used to represent trend curves. If required, declare in a driver worksheet from which variables the values are to be saved in a file and about the configuration of historical data for trend curves. WinStudio saves the recorded values in a binary history file (*.hst). The values can be displayed together with the current variable values in a trend curve.

The Trend worksheet is executed by the background task module (refer to Execution tasks, page 91). It manages saving of trend data in the history, however, does not display this data to the operator. To display trend data, the Trend control object on page 353 in the "WinStudio screen tools" tool window has to be inserted into a WinStudio screen and configured.

To create a new trend worksheet, open the context menu of the "Tasks" folder and select "Trend" in the "Add" menu.

To edit an existing trend worksheet, double-click it in the Project Explorer.

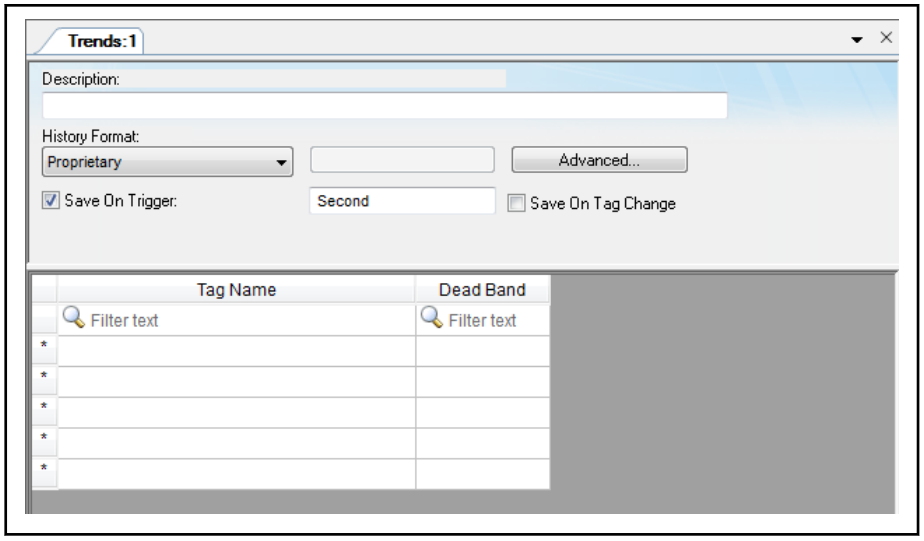


Fig. 9-21: Trend worksheet

The trend worksheet is divided into two areas:

- Header area (top section) containing information for the whole group
- Body area (bottom section) to define each tag in the group. This section contains several columns (only two are shown in the preceding figure).

Use the header parameters on this worksheet as follows:

- **Description** field: Enter a description of the worksheet for documentation purposes.
- **History format**: Click to select a trend history format. The following options are available:
  - **Proprietary**
    - **File format:** Binary
    - **Default path:** …\project_name\Hst\GGJJTTMM.HST, where:
      - YY = two last digits of the year
      - MM = month
      - DD = day

☞ "WinStudio" provides the HST2TXT.EXE and TXT2HST.EXE programs to convert trend history files from binary (.hst) to text (.txt) and vice versa. For further information on these programs, refer to Converting trend history files from binary to text on page 345 and Converting trend history files from text to binary on page 346.

  - **Database**
    - **Database type:** Selected by the user
    - **Default table name:**TRENDGGG (GGG = trend worksheet number; e.g., TREND001 for the trend worksheet 001)

For further information on the structure of the database table that "WinStudio" uses to save history files, refer to Database interface on page 553.



String tags can be specified in many fields of the trend worksheet to change these values at runtime. However, this may influence how these values are stored in the trend history:

- If the history format is **Proprietary**, the value of the String tag is converted to a numerical value (if possible) and then saved to the history file. If numeric conversion is not possible, a value of 0 is saved.
- If the history format is **Database**, the actual value of the String tag is saved in the database.

- **Database Configuration**: Opens the Database configuration on page 563 dialog, to enter the required settings to link the project to an external SQL relational database for the purpose of saving the trend history.
- **Save on trigger** checkbox and field: Click on (enable) the checkbox and type a tag name to save trend samples if someone changes the specified tag. (The tag change can be an event from the Scheduler.)
- **Save on tag change** checkbox: Click (enable) to always save the trend sample in case of a value change in any of the tags from that group.
- **Advanced**: Click to display the "Trend advanced settings" dialog. For information on completing the fields in this window, refer to Batch history configuration on page 346.

Use the body parameters on this worksheet as follows:

- **Tag name** field: Enter the variable name to be saved in the history file.
- **Dead band** checkbox: Enter a value to filter acceptable changes when **Save on tag change** is used.

  Example:

  The dead band has a value of 5. If the tag value is set to 50 and changes to 52, the system does not register this change in the database as it is smaller than 5. If the change is at least 5, the new value is saved in the history file.
- **Field**: Name of the field in the database where the tag is stored. If this field is left blank, the name of the tag is used as the tag name. For array tags and classes, the characters "[", "] " and " . " replaced by " _ ".

  Examples:

Tag name	Default field
MyArray[1]	MyArray_1
MyClass.Member1	MyClass_Member1
MyClass[3].Member2	MyClass_3_Member2



The trend task can accept only up to 240 tags in a single worksheet. If more than 240 tags are manually configured in the same worksheet, the Trend task generates an error on running the finished project.

Also refer to:

- Converting trend history files from binary to text on page 345
- Converting trend history files from text to binary on page 346
- Creating batch history on page 346
- Configuring a default database for all task histories on page 567

9.4.1 Sorting or filtering the lines of a worksheet

Sort or filter the lines of a worksheet to facilitate searching the lines or finding a particular element.

Before starting this task, insert a worksheet and open it for editing. In addition it should be clear how sorting and filtering works in general spreadsheet applications.

Please note that the lines can only by sorted and filtered in the following types of worksheets:

In Project tags, Shared tags and System tags data sheets;

In the Translation table worksheet;

In all Task worksheets except for report and script which do not have lines;

In all Communication worksheets.

The other worksheets do not have lines to sort or filter.

The worksheets are sorted alphanumerically by selected column either in ascending ((0-9, A-Z) or descending order (Z-A, 9-0).

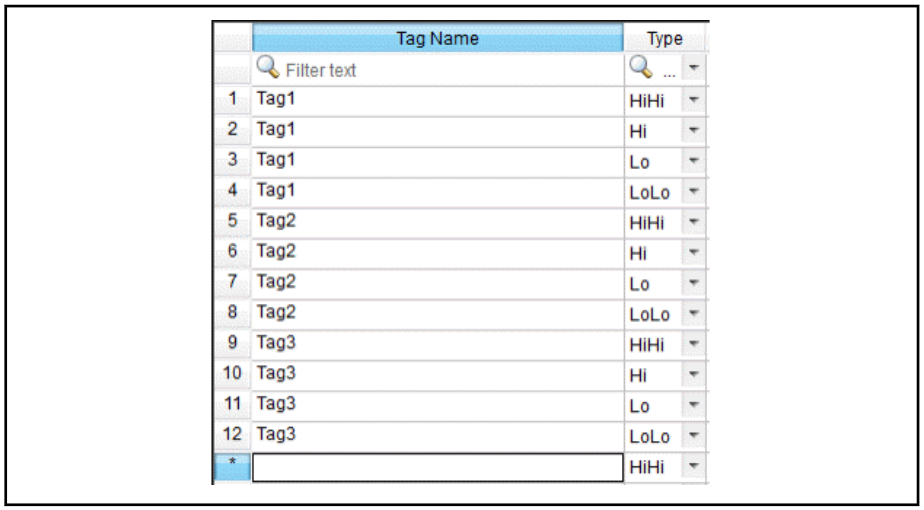
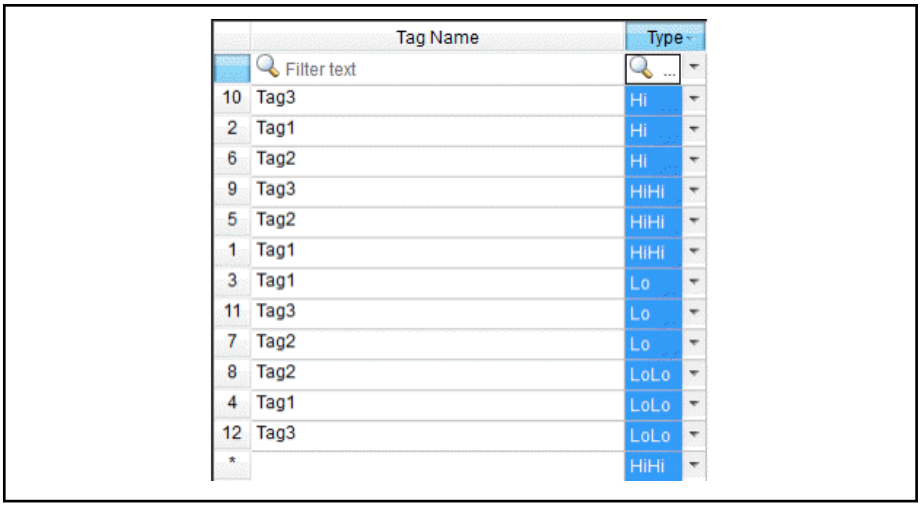


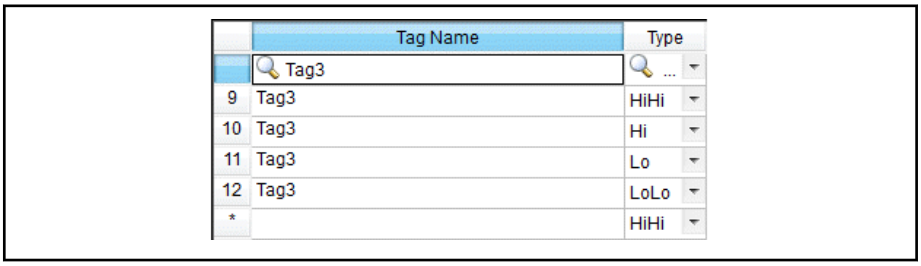
Fig. 9-22: Alarm worksheet lines in their original order



	Tag Name	Type
	Filter text	
10	Tag3	Hi
2	Tag1	Hi
6	Tag2	Hi
9	Tag3	HiHi
5	Tag2	HiHi
1	Tag1	HiHi
3	Tag1	Lo
11	Tag3	Lo
7	Tag2	Lo
8	Tag2	LoLo
4	Tag1	LoLo
12	Tag3	LoLo
*		HiHi

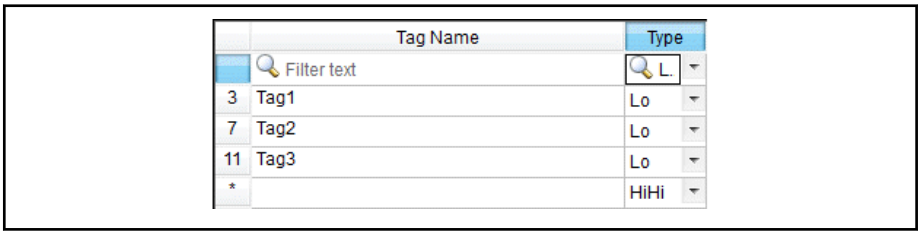
Fig. 9-23: Alarm worksheet lines sorted by type

The worksheets are filtered according to the string entered in the selected column. Only lines that match the string will be displayed.



	Tag Name	Type
	Tag3	
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

Fig. 9-24: Alarm worksheet lines filtered by tag name is "Tag3"



	Tag Name	Type
	Filter text	L.
3	Tag1	Lo
7	Tag2	Lo
11	Tag3	Lo
*		HiHi

Fig. 9-25: Alarm worksheet lines filtered by type is "Lo"



It is still possible to delete lines while they are sorted or filtered.

Sorting or filtering lines:

1. To sort the lines, click on the column header by which you would like to sort. Click once to sort in ascending order and again to sort in descending order. The current order (i.e. the sorting direction) is indicated by the arrow to the right of the column name.



It is not possible to sort by several columns.

2. To cancel the sorting and restore the original order of the lines, click on the header of the first (numbered) column.
3. To filter the lines, enter the sting by which you would like to filter in the top line (zero) of the worksheet and press either **Tab** or **Enter**.

*It is possible to use * and ? as wildcard characters in your string:*

- * stands for any number of characters or no character. Tag* for example corresponds to `Tag`, `Tag3`, `Tag34567`, `TagA` and `Tag_TEMP`.
- ? stands for exactly one character. `Tag?`, for example, corresponds to `Tag3` and `TagA`, while `Tag?????` corresponds to `Tag34567` and `Tag_TEMP`.

It is possible to filter several columns. Only lines that match the filter string in all columns will be displayed.

4. To cancel filtering and restore the original order of the lines, delete the string you entered and either press **Tab** or **Enter**.

Please note that sorting or filtering the lines of a worksheet only facilitates the editing of this worksheet. How the worksheet is executed during runtime does not change. The lines are executed in their original order according to the numbering (column on the left), unless a line is moved or deleted.

9.4.2 Converting trend history files from binary to text

By default, "WinStudio" saves trend history files in a binary format (`.hst`). If these files are required in `.txt` format, "WinStudio" provides the **HST2TXT.EXE** program to convert trend history files from binary into text format.

To convert a file, proceed as follows:

1. From a DOS window, change directory (`cd`) to the "WinStudio" `\Bin` directory:

   ```
   cd C:\Program Files\Rexroth\WinStudio\Bin
   ```

2. At the command prompt, copy the `Hst2txt.exe` into the same directory where the `.hst` file is located.
3. Enter `Hst2txt.exe` into the command prompt and specify the following parameters:
   - `filename`: Name of the trend history file to convert
   - `[separator]`: Data separator character (default is <TAB>)
   - `[/e]`: Advanced functionality (convert data with more than 10 characters)
   - `[/f:HH:MM:SS]`: Start time in hours (`HH`), minutes (`MM`) and seconds (`SS`)
   - `[/f:HH:MM:SS]`: End time in hours (`HH`), minutes (`MM`) and seconds (`SS`)
   - `[/m]`: Include milliseconds in the **Time** column (enter `1` to print the milliseconds value in the text file created from the `.hst` file)

     Example:

     ```
     Hst2txt.exe 01952010.hst
     ```

     The program creates a `.hdr` file (header area) and the `.txt` file. Both are plain text files that can be displayed in any text editor (e.g. Notepad).
   - The `.hdr` file contains the name of the tags configured in the trend worksheet.
   - The `.txt` file contains the tag values saved in the history file.
4. After the program has converted the file, enter "Exit" to close the DOS window.



Alternatively, the calculation script HST2TXT on page 746 can be used in a math worksheet on page 400 to automatically convert binary files into text format without having to use a DOS window.

Also refer to:

- Converting trend history files from text to binary on page 346
- Creating batch history on page 346
- Configuring a default database for all task histories on page 567

9.4.3 Converting trend history files from text to binary

"WinStudio" provides the `TXT2HST.EXE` program to convert text files back into binary format.

To convert a file, proceed as follows:

1. From a DOS window, change directory (`cd`) to the "WinStudio" `\Bin` directory:

   ```
   cd C:\Program Files\Rexroth\WinStudio\Bin
   ```

2. In the command prompt, copy the `Txt2hst.exe` into the same directory where the `.txt` file is located.
3. Enter `Txt2hst.exe` into the command prompt and specify the following parameters:
   - `filename`: Name of the ASCII file with history data to convert
   - `[separator]`: Data separator character (default is <TAB>)
   - `[/e]`: Extended functionality (data value with more than 10 characters)
   - `[/f:HH:MM:SS]`: Start time of data value in hours (`HH`), minutes (`MM`) and seconds (`SS`)
   - `[/f:HH:MM:SS]`: End time of data value in hours (`HH`), minutes (`MM`) and seconds (`SS`)

     Example:

     ```
     Txt2hst.exe 02950201.txt
     ```

     The program creates a `.hdr` ("header") file and converts the `.txt` file into a binary `.hst` file.
4. After the program has converted the file, enter "Exit" to close the DOS window.



For the `TXT2HST.EXE` program, no calculation scripts can be created for use in a math worksheet on page 400 to convert text files into binary format as for `HST2TXT.EXE`. The math script shortcut is available for binary files only.

Also refer to:

- Converting trend history files from binary to text on page 345
- Creating batch history on page 346
- Configuring a default database for all task histories on page 567

9.4.4 Batch creation of history files

"WinStudio" provides powerful tools to create and manage historical batch information. Batches can be created using the following formats:

- **Proprietary**: When using the proprietary format, each batch is stored in a different historical file. Historical data can be saved in the normal historical file and in the batch files simultaneously (for more information on these files, refer to Trend folder on page 340).
- **Database**: The historical data used for the batch is saved in the same table as the normal historical data. An additional table called Batch history contains tabs with information on the batches. The following list describes the fields on the batch history table:

Field name	Data type	Description
Group_Number	Integer	Trend group number. This is the number of the worksheet created to specify the tags to be stored in the batch history.
Batch_Name	String	Batch name
Start_Time	Time stamp	Date and time the batch was started.
End_Time	Time stamp	Date and time the batch was ended.
Pri_Table	String	Reserved
Sec_Table	String	Reserved
Description	String	Batch description
Deleted	Boolean	0: Batch was not deleted 1: Batch was deleted



The name of the table and the name of the columns created in the database can be customized by editing the `project_name.APP` file as follows:

```
[Trend]
DefaultName=NewName
[TrendGroupPRI|SEC]
BatchHistory=Tablename
```

Example:

```
[TREND001PRI]
BatchHistory=MyTableForPrimaryDB

[TREND001SEC]
BatchHistory=MyTableForSecondaryDB

[Trend]
Group_Number=Trend_Worksheet
Batch_Name=Load_Number
```

Configuration of the batch history

Add a trend worksheet (refer to Trend folder on page 340) and click on the Advanced button to display the following window:

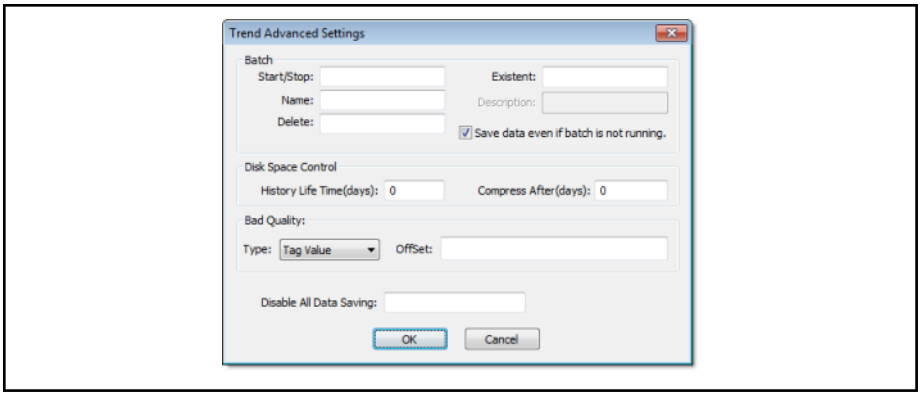


Fig. 9-26: Trend advanced settings

In the **Batch** area, saving the batch history can be configured:

- **Start/Stop (input):** Enter the tag that is to start/stop the batches. If the tag in this field is set to TRUE (different from 0), "WinStudio" either starts saving data to the batch file (if the Proprietary format is used) or adds a new tab to the batch history table on the database indicating that a new batch has been started. Note that historical data is saved according to the configuration of the **Save on trigger** and **Save on tag change** fields on the trend worksheet.
- **Name (input):** This field represents the batch name. The meaning of this field depends on the format selected on the trend worksheet:
  - If **Proprietary** is selected in the **Type** field, the **Name** is to comply with the format **[Path]<FileName>**, where:
    - **Path**: An optional field. If the path is not specified, the batch history file is stored under the same path as the **project_name.app** file.
    - **FileName**: Name of the batch history file.
  - If **Database** is selected in the **Type** field, the value in this field is stored in the **Batch_Name** field of the "BatchHistory" table.



In this field, tag names can be entered in curly brackets (e.g. `C:\MyBatches\{MyTagName}{MyTagNumber.hst`).

- **Delete (input):** If the tag specified in this field changes its value, the batch is deleted. In **Proprietary** format, the batch history file is removed. In the **Database** format, the **Delete** field in the "Batch history" table will be set to TRUE. However, the saved historical data remains. The Trend object only detects batches if the delete field is set to "0" (zero).
- **Existent (output)**: The tag value entered into this field is "1" if the batch specified in the **Name** field exists. Otherwise, the tag value is "0".
- **Description (output)**: This field is only available for the **Database** format. If the tag in the **Start/Stop** field changes to TRUE, the tab added to the "BatchHistory" table displays the string in this field.



In this field, tag names can be entered between curly brackets (e.g., **{MyTag}**).

- **Save data even if batch is not running**: If this checkbox is disabled, historical data will be only saved if the tag in the **Start/Stop** field is TRUE.



Historical batch data can be displayed to the user either in graphical or table format.

Refer to Trend folder on page 340 or Grid object on page 381 to display information in these formats.

In the **Disk space control** pane, the use of the disk space can be controlled:

- **History Life Time (days)** field: Specify how many days the history file is kept on the disk. After the specified period, "WinStudio" automatically deletes the file. Use this option only for files based on a date.
- **Compress after (days)** field: Specify how many days the trend history file (***.hst**) is kept on the disk before compressing the file. After the specified period, "WinStudio" automatically compresses the file. Use this option only for files based on a date. This option is not available for Windows Embedded target systems.

In the **Bad quality** area, the value to be actually saved in the batch history is specified if the tag quality is BAD.

Type	Description
Tag value	The actual value of the project tag if the tag quality was BAD, plus the specified **Offset** (if any).
Min Value	The minimum historical value of the project tag, minus the specified **Offset** (if any).
Max value	The maximum historical value of the project tag, plus the specified **Offset** (if any).
Value	The specified **Value** only.
NaN	Not a number. Please note: If the **history format** is set to **database** and **bad quality** is set to **NaN**, all database fields are saved as Float type. Additionally, if a trend object is configured to use the history created by this trend worksheet, NaN entries are counted as 0 for the calculation of statistic average values and trend deviations.



The **Bad quality** option cannot be used in projects running on Windows Embedded target systems.

Finally, enter the name of a tag into the **Disable all data saving** field. When the value of the tag is TRUE (non-zero) at runtime, data saving is disabled for this worksheet. Other trend worksheets are not affected.

9.5 Trend control object

The trend control object shows data points (values) from different data sources in an image format.

The main functions of the trend control object are:

- Display of multiple trend curves simultaneously
- Support of different data sources, such as tag, batch, database and text file
- Capability to generate X/Y graphs from the configured data sources (please refer to Appendix A on page 374 for an example of an X/Y chart).

- Simultaneous display of an unlimited number of data points. This function might be limited by the hardware used since the available memory and performance may vary.
- Built-in toolbar, which provides interfaces to interact with the trend control object at runtime.
- Built-in legend, which displays the main information associated with each pen linked to the object.
- Zooming and auto-scaling tools
- Horizontal and vertical orientation



This object cannot be moved behind active objects or shapes on a project screen. In other words: it must always be at the front.

For additional information, refer to In the foreground/background, page 269.

9.5.1 About the Trend control runtime interface

The trend display object is provided with individual operating elements used to change the display at runtime. In the following, the main operating elements and their function are described.

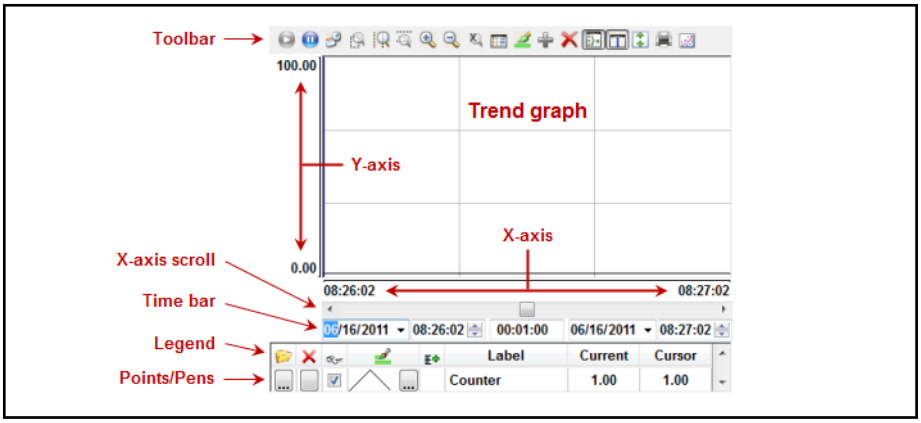
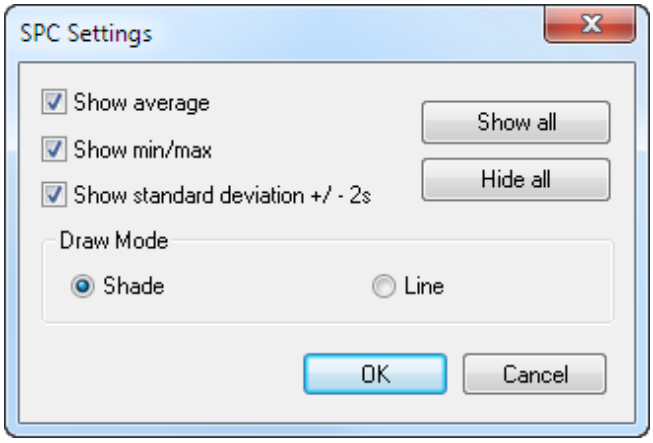


Fig. 9-27: Trend control runtime interface

Toolbar

Command	Symbol	Description	Activation tag
Run		Sets the trend object to the runtime mode (also called online mode). In this mode, the x-axis continues to scroll with the time behavior and the trend curves show the current variable values.	0 = Run mode on 1 = Run mode off Changing the value of this tag also changes the value of the activation tag for stop.
Stop		Sets the trend display to Stop mode (also historical mode). In this mode, the x-axis is stopped and the trends display only historical data. If the "Decimation" option is enabled for one or more trends, calculation and redrawing only takes place in this mode.	0 = Stop mode on 1 = Stop mode off Changing the value of this tag also changes the value of the activation tag for start.
Period		Opens a dialog to modify the main settings for the x-axis scale.	1 = Open dialog Is reset to 0 after opening

Command	Symbol	Description	Activation tag
Window zoom		Allows the user to click on the trend graph and drag the cursor to select the area that has to be visible when the cursor is released. This option is disabled if the **Multiple section** option (for the Y scale) is active.	0 = Disable zoom 1 = Enable zoom Is reset to 0 after user input
Horizontal zoom		Allows the user to click on two points in the trend graph, defining the horizontal scale that has to be available.	
Vertical zoom		Allows the user to click on two points in the trend graph, defining the vertical scale that has to be available. This option is disabled if the **Multiple section** option (for the Y scale) is active.	
Zoom in		Enables zooming in when clicking on the trend graph.	1 = Execute command Is reset to 0 after execution
Zoom out		Enables zooming out when clicking on the trend graph.	
Cancel zoom		Cancels the current **Window**, **Horizontal** or **Vertical zoom** and returns the trend graph to its original scale.	
Legend properties		Opens a dialog to modify the legend main settings.	
Pen style		Opens a dialog to modify the pen style of the selected trend.	
Add pen		Opens a dialog to add a new trends to the trend display.	
Remove pen		Removes the selected trend from the trend display.	
Multiple sections		Switches the Y scale between multiple sections (a section for each trend) and single section (all trends share the same Y scale section).	0 = Individual section 1 = Multiple sections
Cursor		Switches the cursor (ruler) between visible and hidden.	0 = Cursor hidden 1 = Cursor visible

Command	Symbol	Description	Activation tag
Auto scale		Changes the Y axis scale to fit all values from the trends that are currently being monitored.	1 = Execute command Is reset to 0 after execution.
Print		Prints the current state of the trend control. (Historical data is not printed.)	
SPC		Opens a dialog to display the statistical process control (SPC) information for the selected trend:  *Draw mode…* • **Shade**: Draws the average value as a dashed line and the min/max values and standard deviation as shaded areas. **Line**: Draws the average value and standard deviation as dashed lines and the min/max values as solid lines. • **Show average**: Show the calculated average of all historical trend values. **Note**: If a value is invalid (NaN) – for example, if a tag is marked with the attribute "Bad quality" – it is counted as 0 when calculating the average. • **Show standard deviation**: Shows the standard deviation of the trend. A low standard deviation indicates that the actual value tends to stay close to the average. A high standard deviation indicates that the actual value tends to vary greatly from the average. • **Show min/max**: Shows the minimum and maximum historical values of the trend.	If the activation tag changes its value (e.g., toggles), this command is executed. Additionally, the bit properties of the tag (B0–B4) can be used to pre-select options in the dialog: • **tagname->B0** 0 = Close dialog 1 = Open dialog • **tagname->B1** 0 = **Draw mode: Line** selected 1 = **Draw mode: Shade** selected • **tagname->B2** 0 = **Show average** deleted 1 = **Show average** selected • **tagname->B3** 0 = **Show standard deviation** deleted 1 = **Show standard deviation** selected • **tagname->B4** 0 = **Show min/max** deleted 1 = **Show min/max** selected

Activation tags are configured in the object properties of the trend object. For further information, refer to Toolbar on page 365.

Time bar

The time bar shows the start date/time and the end date/time of the graph or, in other words, the time period of the X axis. For further information, refer to Axes dialog, page 360

If the trend control is configured to show historical data from a batch file, the data and times can be changed during project runtime to increase or reduce the time period of the graph. The longer the time period, the more data is displayed by the graph but with lower resolution. The shorter the time period, the

less data is displayed by the graph but with higher resolution. This corresponds to the "Horizontal zoom" tool.



If a (start or end) date prior to 01/01/2000 is entered, it is corrected automatically to 01/01/2000. If a (start or end) date after the current date is entered, it is corrected automatically to the current date.

Legend

Command	Symbol	Description
Selection		Opens a dialog to replace the data point associated with the selected trend on the legend.
Remove		Removes the selected trend from the trend display.
Hide		If this option is enabled, the selected trend is invisible. Otherwise, it is displayed.
Pen style		Opens an embedded dialog to modify the pen style of the selected trend.
Scale		If this option is enabled, the Y axis scale is visible. Otherwise, it is hidden. The scale can only be hidden if the “Multiple sections” option is disabled.

9.5.2 Object properties: Trend control dialog

The “Object properties” dialog: Trend control is used to configure the basic properties of a trend control screen object.

Accessing the dialog

To access the "Object properties" dialog for a specific screen object, proceed as follows:

- Open the WinStudio screen with the trend object to be configured in the editor
- Right-click on the screen object and select **Properties** in the context menu
- Double-click on the screen object.

The dialog in detail

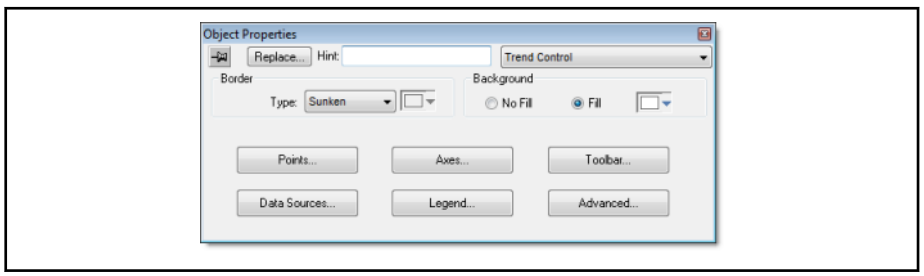


Fig. 9-28: Object properties: Trend Control dialog

In addition to the elements that are similar in all Object properties dialogs, page 178, the "Object properties dialog: Trend object" provides the following elements:

Pane/element name		Description
Border	**Type**	Sets the type of border around the graph area of the trend display. (There are no borders around the legend or toolbar of the trend display)
	Color	Sets the color of the border, if the border type is set to **Solid**. For further information, refer to Selecting colors and fill effects.

Pane/element name		Description
Background	**NoFill/Fill**	Enables the background fill for the graph area of the trend display. (There are no backgrounds for the legend or toolbar of the trend display.) If the fill is disabled, the graph is transparent to for any other screen object behind the trend display.
	Color	Sets the color and fill effect of the background fill if enabled. For further information, refer to Selecting colors and fill effects.
Points		Opens the Points dialog to configure the data points (or pens) of the trend display. For further information, refer to Trend object: Points dialog on page 354
Axes		Enables configuration of the X and Y axes of the trend display as well as the horizontal or vertical position of the axes. For further information, refer to "Trend display: Axes"
Toolbar		Enables configuration of the user toolbar that is displayed above the trend display. For further information, refer to "Trend display: Toolbar"
Data sources		Enables configuration of multiple data sources for the trend display. For further information, refer to "Trend display: Data sources"
Label		Enables configuration of the legend that is displayed below the trend display. For further information, refer to "Trend display: Legend"
Advanced		Enables configuration of the advanced properties of the trend display, e.g. runtime options and tag triggers. For further information, refer to "Trend display: Advanced settings"

Although trend objects support flexible configurations to fulfill specific project requirements, most of the settings are set by defaults based on the most common functions. Therefore, in many cases, only data points are configured (displayed at runtime) by simply clicking on the **Points** button in the "Object properties" window.

Points dialog

The "Points" dialog is used to configure the data points for a trend display screen object. The value of each data point is represented as a pen in the trend display. It can be changed dynamically whether a data point is visible at runtime, regardless of how many data points are associated with the screen object.

Accessing the dialog

To access the "Points" dialog for a specific trend control screen object, first select the Object properties dialog for the screen object on page 353 and click on **Points**.

The dialog in detail



Fig. 9-29: "Object properties" dialog

The following table summarizes the properties of each data point:

Column name	Description
Point	A unique ID number for the point, which is assigned automatically when the point is created in this interface.
Label	The label associated with the point can be displayed on the legend at runtime, providing a short reference to for each point.
Color	The color of the pen used to draw the values of the point on the trend display.

Column name	Description
Data source	The data source for this point. **Tag** is available by default. All other sources have to be configured in the "Data sources" dialog.
Variable/field	The meaning of this parameter depends on the data source type associated with the data point: If the **Data source** is **Tag**, enter the name of the tag including the values to be displayed. If the tag is configured in a trend task worksheet, its history is called up and displayed automatically. Otherwise, only the online values of the tag — i.e. the actual values of the tag during runtime — are displayed. (Please note that this means that the trend line of the tag cannot be redrawn after zooming; only new values can be drawn when received. For further information on zooming, refer to Trend control runtime interface, page 350.) If the **Data source** is **Batch**, enter the name of the tag, for which values are to be called up from the batch history file created by the trend task worksheet. If the **Data source** is **Database**, enter the name of the field (column) in the SQL relational database that holds the data point values. For further information, refer to Using a database as data source, page 377. If the **Data source** is **Text file**, enter the number of the column in the text file that contains the values of the point. The number 0 refers to the first column, 1 refers to the second column, etc. For further information, refer to Using a text file as data source, page 374.
Min scale / Max scale	The scaling of the Y-axis for this point. It overrides the default scaling that is set in the Axes dialog, page 360. **Note:** The **Min scale** and **Max scale** properties can hold real numeric values with up to six decimal places. If higher precision is required, the **Min scale** and **Max scale** properties have to be configured with Real tags and the values have to be stored in those tags.
Style	The Line and marker style for this point. Click on the … button to open the Pen style dialog, page 355.
Options	Additional options for this point. Click on the … button to open the Options dialog, page 357.
SPC	Calculated statistics to be used in statistical process control (SPC). Click on the … button to open the SPC dialog, page 360.
Hide	Tag trigger: If the value is TRUE, the data point is hidden in the trend display.

Pen style

In this dialog, the style of the pen used to draw the data point values on the object at runtime can be configured.

Accessing the dialog

To access the "Pen Style" dialog for a specific data point, open the "Points" dialog and click on the **Style** column for the data point.

It can also be opened at runtime to customize these settings on-the-fly.

The dialog in detail

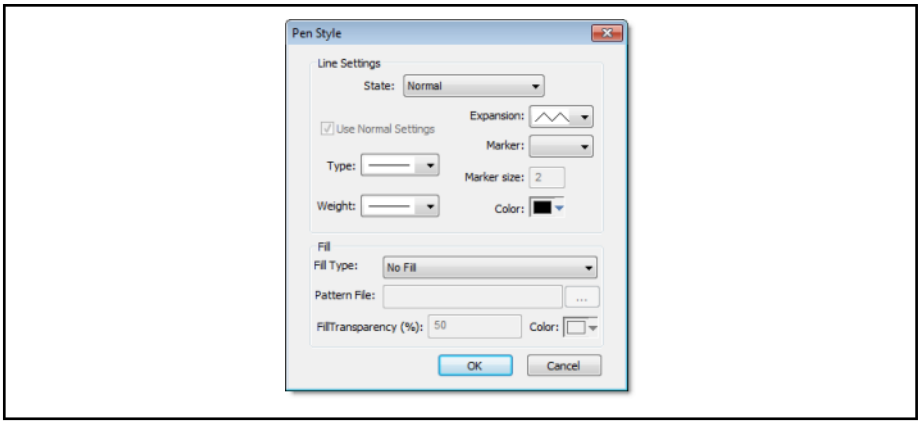


Fig. 9-30: Pen Style dialog

The "Pen Style" dialog includes the following elements:

Elements in the "Pen Style" dialog

Pane / Element name		Description
Line Settings	**State**	With the Options dialog a Hi Limit and a Lo Limit for each data point can be defined. The Pen Style dialog enables configuring different settings for the pen (e.g., color), both when its values are within the limits (Normal state) and not within the limits (Out of Limits state).
	Use Normal Settings	Available only for the **Out of Limits** state. If this option is enabled, the pen will always be displayed with the settings for the Normal state, even if the data point values are not within the limits configured for it.
	Type	The type of line (e.g., solid, dashed, dotted) that connects the data points.
	Weight	The weight of the line that connects the data points.
	Expansion	The algorithm used to connect the points, as follows: •  : Consecutive points are directly connected to each other by an analog line. This option is suitable for numerical values. •  : Consecutive points are only connected by horizontal or vertical steps (depending on the orientation of the trend display). This option is suitable for Boolean values.
	Marker	The shape used to mark each data point. If no shape is selected, only the connecting line between points is displayed.
	Marker size	The size of the data point marker.
	Color	The color of the trend line and data point markers.
Fill	**Fill type**	The type of fill between the trend line and the number line.
	Pattern file	The graphics file used to fill the trend area. Available only if the Fill Type is set to **Custom pattern**. Click the browse button to open a Windows file browser and select the desired graphics file. The file should be located in the project folder. The following example shows trends with custom fill patterns.
	Color	The color used to fill the trend area. Available only if the Fill Type is set to **Solid color**.
	Fill Transparency (%)	The transparency level of the fill. (If the fill is transparent, other trends behind it can be seen through it, making the entire graph easier to read.) Available for both **Custom pattern** and **Solid color**.

When viewing the project on either a Windows Embedded device or a Thin Client (any OS), the Pen Style dialog available during runtime only enables changing the pen color.

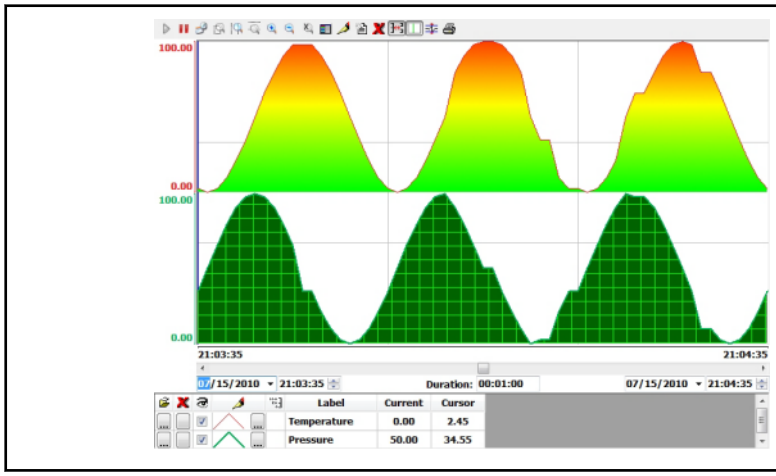


Fig. 9-31: Trends with custom fill patterns

Options

Changing the pen style of a point

Use the Options dialog box to configure additional options for a specific point in a Trend control object, page 349.

Accessing the dialog

To access the "Pen style" dialog for a specific data point, first access the "Points" dialog and then click the **Options** column for that data point.

The dialog in detail

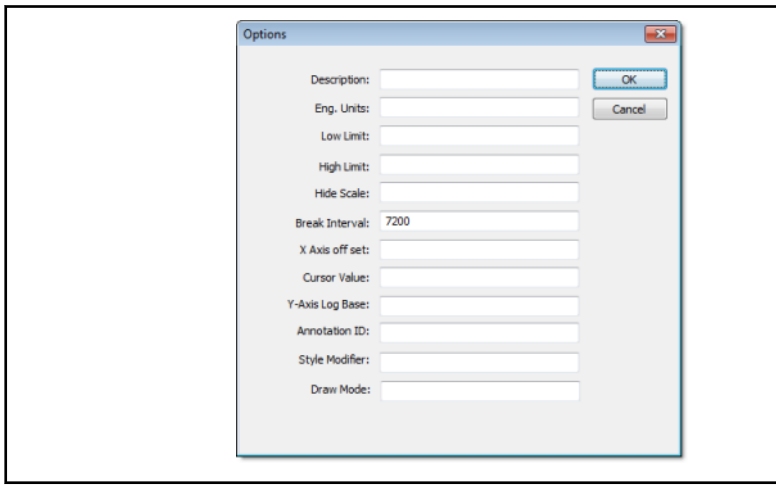


Fig. 9-32: Options dialog

The "Options" dialog includes the following elements:

Element name	Description
Description	This text can be displayed in the legend, providing a short description about the data point at runtime. When tags are used the default description is the one configured for the tag.
Eng. Units	This text can be displayed in the legend, providing the engineering unit (i.e., the unit of measurement) associated with the data point at runtime. If tags are used the default units are the ones configured for the tag.

Element name	Description
Low limit	If the data point value is below this limit its pen can be displayed with a different style (e.g., color) at runtime. For further information, refer to Pen style dialog on page 354. If tags are used, the lower default limit value is the low alarm value configured for this tag.
High limit	If the data point value is above this limit its pen can be displayed with a different style (e.g., color) at runtime. For further information, refer to Pen style dialog on page 354. If tags are used, the default upper limit value is the high alarm value configured for this tag.
Hide scale	In this field, a tag can be configured to control the visibility of the scale (Y axis) associated with this pen at runtime by changing the value of this tag (0=Show; 1=Hide).
Break interval	Maximum interval between two consecutive points. If the time between two consecutive samples is higher than this number (in seconds) the trend control assumes that there was no data collection in this period and does not draw a line linking both samples. If the x-axis is configured as numeric the value on this field represents a numeric scalar value. If the x-axis is configured as date/time the value for this field is given in seconds. This field has some special values: • -1: Do not connect points. • -2: Connect only points in ascending order.
x-axis offset	Offset for this data point from the x-axis scale configured for the object. This option is useful to display data from two or more data points using a different X scale (period of time/value) for each one to compare them. This enables comparing them. If the x-axis is configured as numeric the value on this field represents a numeric scalar value. If the x-axis is configured as date/time the value for this field is given in seconds.
Cursor value	Enter the name of a project tag. At runtime, the trend object updates the value of this tag with the value of the intersection between the data point pen and the vertical cursor (if any).
y-axis log base	Enter a tag name or numerical value. If the value is 1 the y-axis of the trend is displayed on a logarithmic (e.g., 1, 10, 100, 1000) rather than linear scale.
Annotation ID	N/A
Style modifier	Changing the expansion, line color, line thickness or line type of the pen style.
Draw mode	Enter a tag name or numerical value. If the value is 1 the historical data for this trend is decimated before the trend is drawn. This means, the x-axis of the trend is divided into a number of intervals (determined by **Max points** in the Advanced on page 371 settings) and all of the data points within each interval are averaged together to be drawn as a single point. This is similar to the Decimation feature in the Advanced settings, except that the decimation is done only for this trend rather than for all trends in the trend display.



If Decimation is enabled and the X axis is configured as a numerical value instead of date/time, the data used in the X axis have to be properly sorted. For further information, refer to Data source settings, page 383.

Changing the pen style of a point during runtime

By default, the data points in a trend control are drawn in a thick black line. The style of this line can be changed — or the style of the pen drawing the line — by changing the settings in the Pen style dialog box. It is also possible to open the Pen style dialog box and change the settings during runtime. For further information, refer to Pen style dialog box, page 355.

Alternatively, the Style modifier (in the Options dialog box) can be used to change some of the pen style settings during runtime in the program. In other words: It is possible to enter project tags which determine the pen style settings and to change the tag value during runtime using scripts or user input.

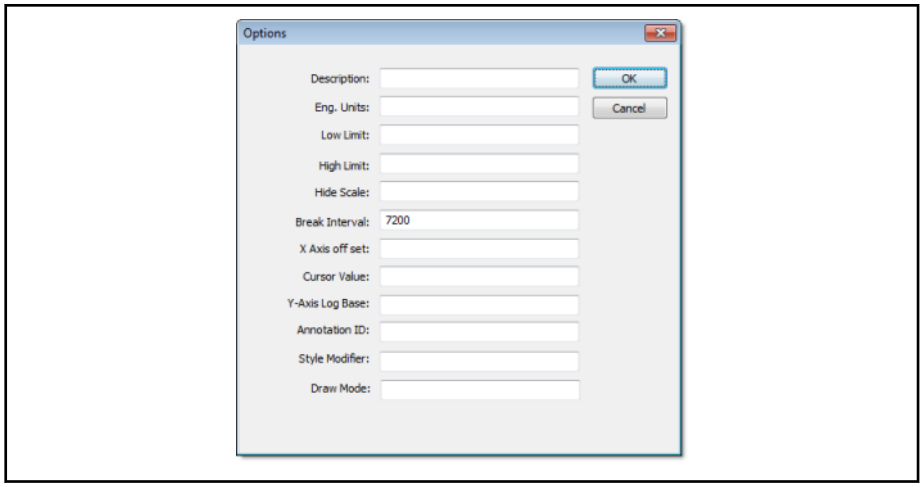


Fig. 9-33: Options dialog

A text string with one or more parameters can be entered in the Style modifier field and each parameter modifies an element of the pen style. The text string must have the following basic format:

Syntax

```
<Parameter1>=<Value1>;<Parameter2>=<Value2>;…;<ParameterN>=<ValueN>
```

The following table lists the supported parameters and their accepted values:

Parameter	Description	Accepted values
Expansion	The method or algorithm used to link the data points.	0 (straight/analog) 1 (angular/digital)
Type	The type of line (e.g., solid, dashed, dotted) that connects the data points.	0 (solid) 1 (dashed) 2 (dotted) 3 (dash-dot) 4 (dash-dot-dot)
Weight	The weight (i.e. the thickness) of the line in pixels.	from 0 to 10
Color	The color of the line. For further information on colors in WinStudio, refer to Color interface, page 277.	a 24 bit color value between 0 and 16777215

☞ These parameters correspond to the settings in the pen style dialog box.

It is possible to enter direct values for each individual or for all parameters but this is essentially the same as using the Pen style dialog box to configure these settings. The key to modifying the pen style in the program is to enter tag names or expressions in curly brackets ({}) for the parameter values. The pen style is then adjusted if the value of a specific tag/expression changes.

Enter, for example, the following string in the **Style modifier** field:

```
Expansion={ExpansionTag};Color={RGBColor(0,0,ColorTag)}
```

For the `Expansion` parameter, the `ExpansionTag` project tag was entered in curly brackets. There are only two accepted values for the `Expansion` (0 and 1), i.e. the `ExpansionTag` could be Boolean type. If the **ExpansionTag** value changes during runtime, the expansion method is adjusted accordingly.

For the `Color` parameter, an expression was entered which calls the RGBColor, page 763 function to convert the RGB color values to a 24 bit color value. The color values red and green remain constant at 0 but the color value blue is determined by the ColorTag project tag. If the ColorTag value changes during runtime, the value returned by the functions also changes and the line color is adjusted accordingly.

Following this example, it is possible to enter any project tag or expression for the parameters as long as your return values are acceptable values for the parameters.

SPC

In the "SPC" dialog, project tags can be specified that will receive certain statistical values that are calculated from the entire data point history. These statistics are used in statistical process control (SPC), which is a method for monitoring processes and ensuring that they operate efficiently.

Accessing the dialog

To access the "SPC" dialog for a specific data point, first open the Trend control object, page 354 and click on the **SPC** column for that data point.

The dialog in detail

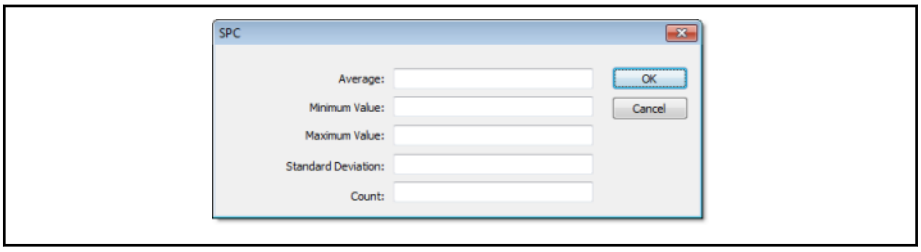


Fig. 9-34: Dialog: “Trend object” properties

The "SPC" dialog includes the following elements:

Element name	Description
Average	Project tag (Real type) that receives the calculated average of all of the historical values of the data point. If a value is not a number (NaN) – for example, if a tag is marked as BAD quality – it is counted as 0 for the purpose of calculating the average.
Minimum value	Project tag (Real type) that receives the minimum historical value of the data point.
Maximum value	Project tag (Real type) that receives the maximum historical value of the data point.
Standard deviation	Project tag (Real type) that receives the standard deviation of the data point. A low standard deviation indicates that the value of the data point is close to the average. A high standard deviation indicates that the value tends to vary greatly from the average.
Count	Project tag (Integer or Real type) that receives the total number of historical values or samples for the data point. The count increases as the project runs and the historical database grows.

Axes

Accessing the dialog

To access the "Axes" dialog for a specific trend object, first select the Object properties dialog for the screen object on page 353 and click on **Axes**.

The dialog in detail

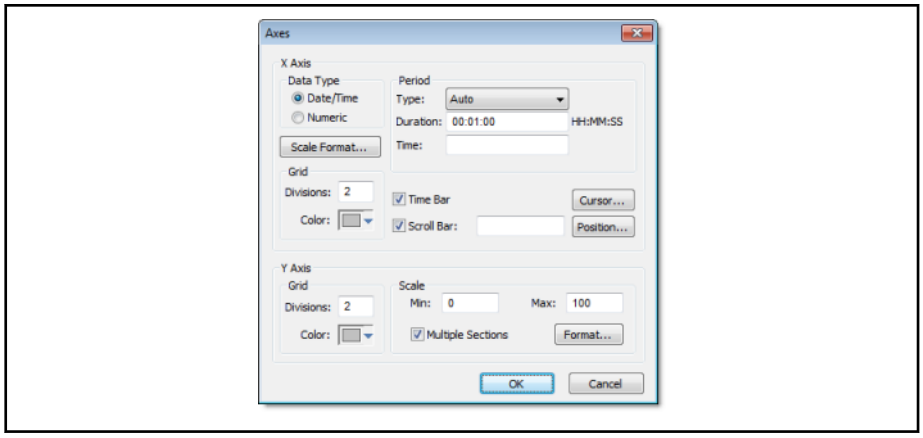


Fig. 9-35: Axes dialog

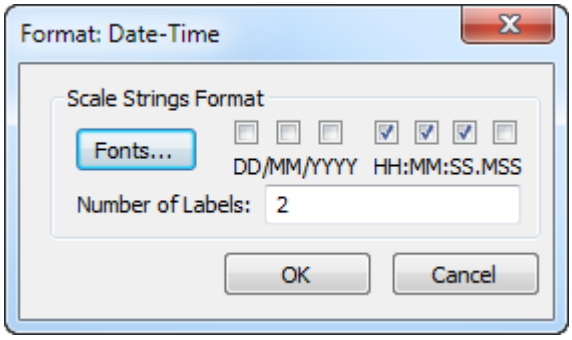
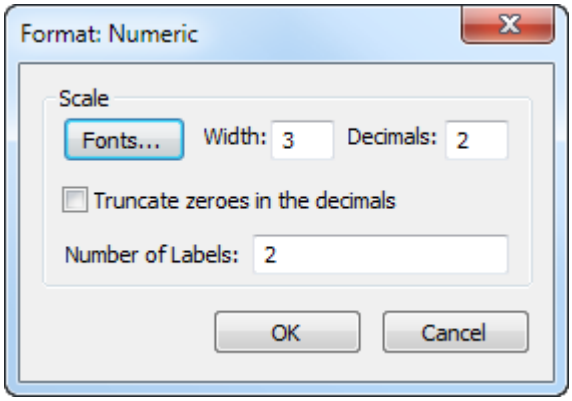
The "Axes" dialog contains the following elements:

Pane/element name			Description
X Axis (1)	Data type	Date/Time	
		Numeric	
		Scale format	
	Period (if data type is set to Date/time)	Type	• **Auto**: If this option is selected the trend control object works with **Start date/time** when triggered to Pause mode and works with **Time before now** when triggered to Play mode. • **Start date/time**: If this option is enabled, the value of the tag configured in the **Time** field defines the starting date/time for the data displayed on the object. • **Time before now**: If this option is enabled, the value of the tag configured in the **Time** field defines the time before the current date/time, which is used as the starting date/time for the data displayed on the object.
		Duration	Defines the period of data displayed on the object. A String tag can be configured in this field to change the duration dynamically at runtime by changing the value of this tag. The format of the value supported by this property is HH:MM:SS. For example, `36:00:00` (thirty six hours).
		Time	This field is optional. The value of the tag configured in this field represents a period of time, rather than a specific date or time. The meaning of this value depends on the option set for under **Type**. If the **Time** field is left blank (or if the tag configured in this field has the value `0`), the object displays data up to the current date/time. • If **Type** is set to start date/time, the value of the tag configured in this field has to comply with the date/time format. For example, `02/10/2005 18:30:00`. • If **Type** is set to the time before now, the value of the tag configured in this field has to comply with one of the following formats: **Time (String value)** e.g. `48:00:00` (forty eight hours). **Number of hours (Real value)** e.g. `2.5` (two hours and thirty minutes). If the **Time** field is left blank (or if the tag configured in this field has the value 0), the object displays data up to the current date/time.
	Period (if data type is set to Numeric)	Min/Max	Minimum and maximum values displayed on the x-axis. The **Min** and **Max** properties can hold real numeric values up to six decimal places. If higher precision is required, the **Min** and **Max** properties have to be configured with Real tags and the values have to be stored in these tags.
		Eng. units	Engineering unit (e.g., kg, psi) that is associated with the x-axis at runtime.

Pane/element name			Description
X axis (2)	Grid	Divisions	The number of divisions (vertical or horizontal lines) drawn on the object can be configured for the X- and/or y-axis respectively as well as the color of these lines.
		Color	
		Time bar	If enabled, the time bar is displayed below the x-axis at runtime; otherwise, it is hidden. The time bar is a standard interface that can be used by the operator to change the x-axis scale at runtime.
		Scroll bar	If checked, the scroll bar is displayed below the x-axis at runtime; otherwise, it is hidden. The scroll bar is a standard interface that can be used by the operator to navigate through the x-axis scale at runtime. Optionally, a tag can be configured in the Scroll Bar field to define the period for the scroll bar. If this field is left empty, the period is equal to the current value for duration of the x-axis.
		Cursor	The cursor is an optional ruler orthogonal to the x-axis that can be used at runtime to obtain the value of any pen at a specific point (intersection of the pen with the cursor). Press this button to display the cursor dialog to configure the settings for the optional vertical cursor as follows:
		Position	Defines the position of the x-axis as well as its direction and orientation, as follows:
y-axis	Grid	Divisions	The number of divisions (vertical or horizontal lines) drawn on the object can be configured for the x- and/or y-axis respectively as well as the color of these lines.
		Color	
	Scale	Min/Max	Default minimum and maximum values displayed in the y-axis. Used if more than one pen share the same scale (Multiple sections disabled) and/or for points whose “Min” and “Max” fields are not configured (left blank).
		Multiple sections	If enabled, the y-scale is automatically divided into one section for each pen; otherwise, all pens share the same Y-scale.
		Format	Opens a dialog for configuring the format of the labels displayed by the y-axis.

☞ The tags configured in the **Period/Range** fields are automatically updated if the X-scale is changed dynamically at runtime, using the time bar embedded in the object.

- **Data type**: According to this setting, the X-axis can display either date/time values or numeric values.

Data type	Scale format
Date/Time	
Numeric	

The number of decimal points for the X-or Y-scale (decimals) can be configured with a tag. Therefore, this setting can be modified dynamically at runtime.

- **Cursor**: The cursor is an optional ruler orthogonal to the x-axis that can be used at runtime to obtain the value of any pen at a specific point (intersection of the pen with the cursor). Press this button to display the cursor dialog to configure the settings for the optional vertical cursor as follows:

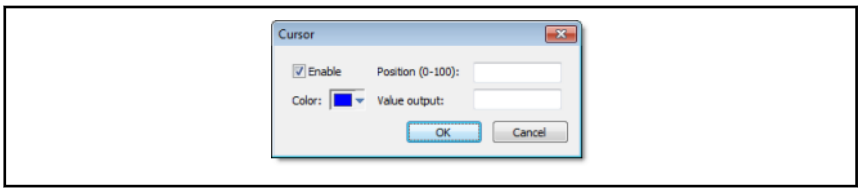


Fig. 9-36: Cursor dialog

Property	Description
Enable	If enabled, the vertical cursor is visible at runtime.
Color	Color of the line drawn for the cursor.
Position (0-100)	In this field, a numeric tag can be configured, which is proportional to the position of the cursor on the x-axis, from 0 to 100%. If this value is changed the position of the cursor is automatically modified.
Value output	In this field, a String tag can be configured that returns the value of the x-axis in which the cursor is currently positioned.

- **Position**: Defines the position of the x-axis as well as its direction and orientation, as follows:

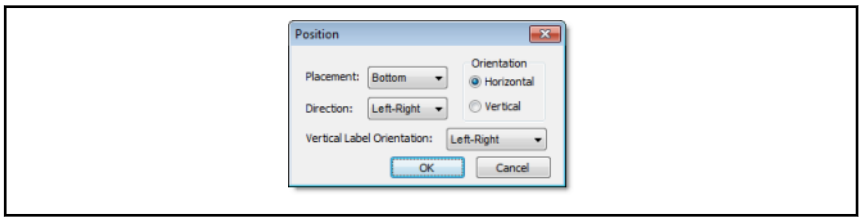


Fig. 9-37: Position dialog

Property	Description
Placement	The side of the trend control on which the x-axis is placed.
Direction	Direction of the x-axis.
Orientation	Orientation of the x-axis.
Vertical orientation	The orientation of the text labels on the vertical axis, regardless of whether the vertical axis is X or Y.

Toolbar

The "Toolbar" dialog is used to adjust the toolbar in a trend object.

Accessing the dialog

To access the "Toolbar" dialog for a specific trend object, first select the Object properties dialog for the screen object, page 353 and click on **Toolbar**.

The dialog in detail

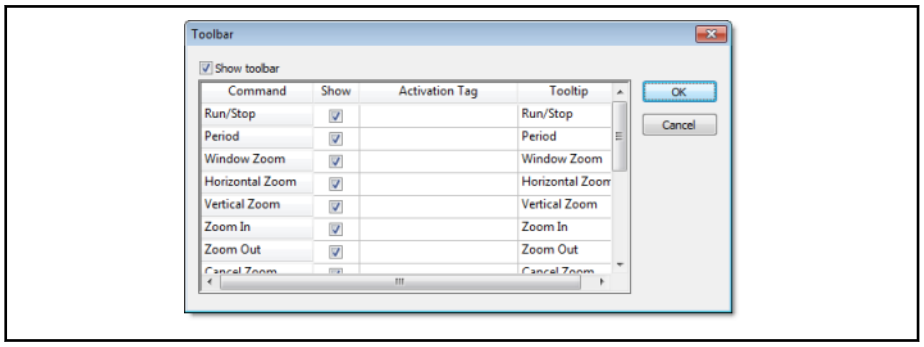


Fig. 9-38: Toolbar dialog

The **Show toolbar** option controls whether the entire toolbar is shown at runtime. The toolbar can be hidden to save space or to prevent users from changing the trend display.

The commands/tools on the toolbar have the following properties:

Column name	Description
Command	The name of the command/tool. For further information on each tool, refer to
Show	The option to display the tool on the toolbar.
Activation tag	An optional tag trigger – if the value of the tag changes from FALSE (0) to TRUE (any non-zero value), the command is activated as if the operator clicked on the tool. This function can be used to script changes in the trend display at runtime.
Tooltip	The tooltip that is displayed when the mouse cursor is moved over the tool.

Data source

Accessing the dialog

To access the "Data sources" dialog for a specific trend object, first select the Object properties dialog for the screen object on page 353 and click on **Data sources**.

The dialog in detail

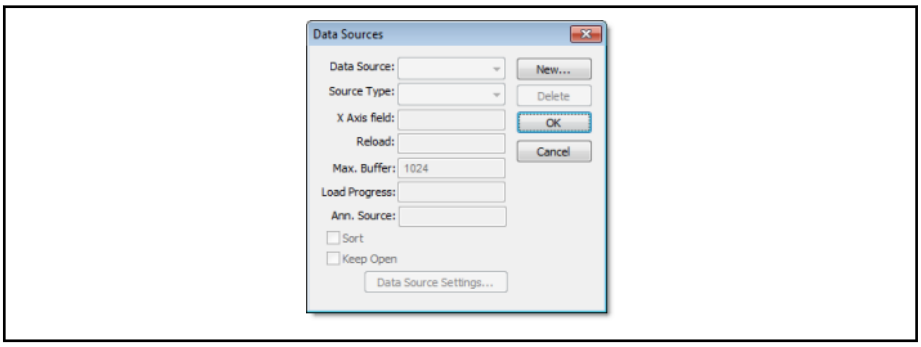


Fig. 9-39: "Object properties" dialog: Trend object

The **Data source** defines the location of the values of the data points associated with it. Multiple points can share the same data source – it is not necessary to create a data source for each data point.

The data source tag is available by default to the trend object. Additional data sources can be added with the **New** button. The name entered is used as an alias to link the data points to this new data source.

The other fields in this dialog enable editing the data source settings:

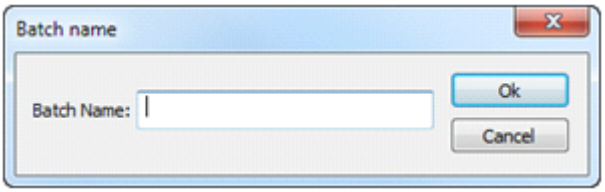
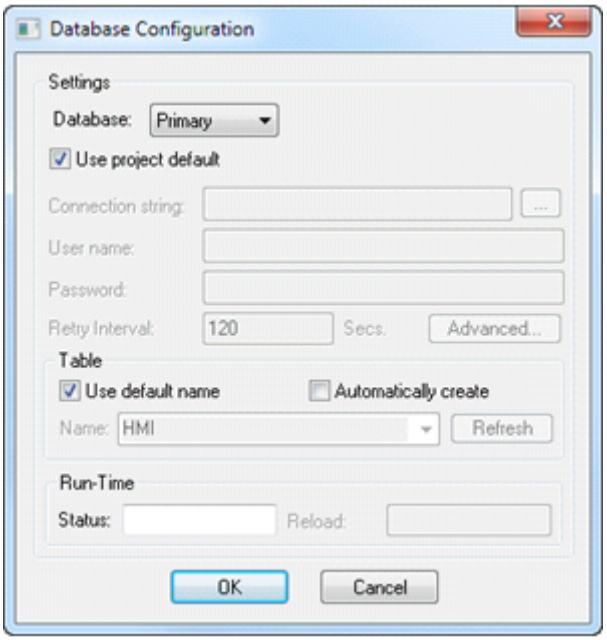
- **Source type**: Select the source type of the location of the data point values.
- **X-axis field**: Enter the name of the field (column) of the data source that holds the x-axis data.

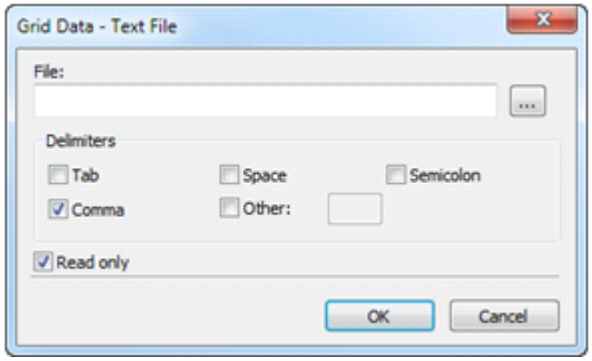
☞ If Decimation has been enabled (either in the Advanced settings, page 371, for all trends or in the Options, page357, for an individual trend), the field specified here has to be sorted in ascending order. The procedure for this varies depending on the source type (e.g. text editor, spreadsheet program, external database, etc.). For further information, refer to the documentation for your specific type.

- **Max. buffer**: The maximum amount of data (in bytes) that is held in the runtime memory.
- **Load progress**: The tag in this field receives a real value (0-100) that represents the percentage of the data source load progress.
- **Sort**: This option is useful for plotting data from a text file. If this option is enabled (checked), the data is sorted and the Cursor column value is displayed until the **Max. buffer** is full. If this function is disabled (unchecked) the data is not sorted and the Cursor column value is not displayed.
- **Keep open**: This option keeps the data source open as long as the screen that contains the trend display is open. This increases the performance of the runtime project. However, keeping the data source open may cause other problems like database connection errors (if **Source type** is set to **Database**) and file write conflicts (if **Source type** is set to **Batch** or **Text file**). To close the data source after the data has been loaded, disable (uncheck) this option.

- **Data source settings**: Click to define the settings for the selected **Source type**.

The following table summarizes the settings for each **Source type**:

Da-tasource type	Description	X-axis field	Data source settings
Batch	Batch generated by the **Trend** task of "WinStudio"	Disabled. The x-axis data is automatically retrieved at the correct position of the proprietary batch file generated by "WinStudio".	 Enter the data point values in the **Batch name** field for their retrieval. In this field, tags can be configured in curly brackets to change this setting dynamically at runtime.
Database	SQL relational database	Field name that contains the x-axis data	 Configure the settings to link this data source to the SQL relational database that holds the data point values. For further information on this dialog interface, refer to Database configuration dialog window on page 563. Please refer to Appendix B on page 377 for an example of configuring databases.

Datasource type	Description	X-axis field	Data source settings
Text file	Text file (e.g., CSV file) with data point values separated by a specific delimiter	Number of the column that holds the X axis data. The number 0 refers to the first column, 1 refers to the second column, etc.	 Enter the name of the text file that holds the data points. The default path is the current project folder. In this field, a tag in curly brackets can be configured to dynamically change this setting at runtime. Additionally, one or more delimiters can be selected for the data saved in the text file. The value of each line is written in the text file between two delimiters. When using a comma as a delimiter, the grid object can read data from CSV files. Also custom delimiters can be selected by selecting the **Other** option. Please refer to Appendix A on page 374 for an example of configuring text files.

☞ There is a default query timeout of 120 seconds to prevent the project client from wasting time on an unusually long data source query. To adjust the timeout, change your project file (`projectname.APP`) manually and add the following setting:

```
[Trend]
QueryTimeout=120
```

If the value is changed to 0, there is no timeout at all; data source queries always continue until they are completed.

Legend dialog

Accessing the dialog

To access the "Legend" dialog for a specific trend object, first select the Object properties dialog for the screen object on page 353 and click on **Legend**.

The dialog in detail

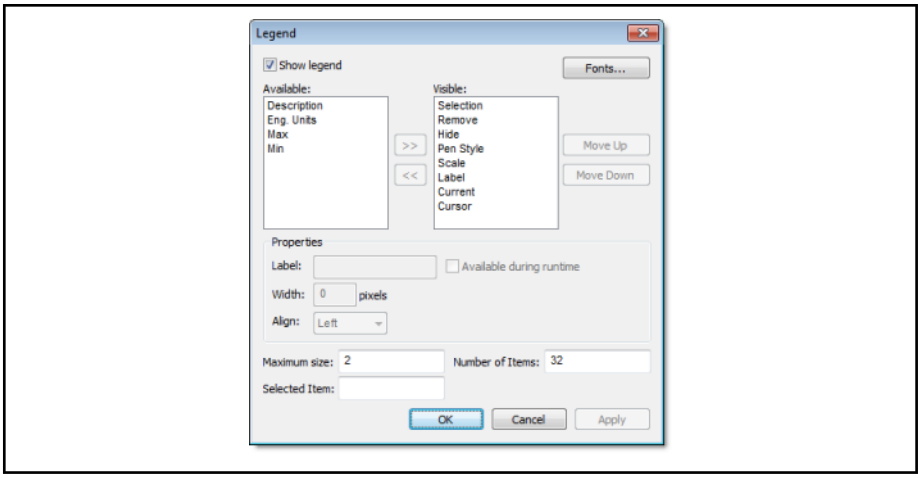


Fig. 9-40: Legend dialog

The "Legend" dialog contains the following elements:

- **Show legend**: If enabled, the embedded legend is displayed at runtime. This interface provides useful information regarding the currently used pens.
- **Available/Visible**: The items in the **Visible** box are displayed in the legend at runtime. Items can be added to and removed from the **Visible** field using the **»** and **«** buttons respectively. Additionally, the **Move up** and **Move down** buttons can be used to change the order in which the items are displayed in the legend at runtime. The following table lists the available legend items:

Element	Legend icon	Description
Eng. Eng. units		The Engineering Units of tag/pen
Min Value		The minimum possible value of tag/pen
Max value		The maximum possible value of tag/pen.
Selection		Press this button to select a different tag for this pen
Remove		Press this button to completely remove this pen from the legend and the trend chart
Hide		Select (check) this option to hide this pen in the trend chart
Pen style		Press this button to change the line style, weight, color, markers, etc. of the pen
Scale		Select (check) this option to display the pen scale on the trend chart
Description		Description of the tag/pen
Current		The current value of the tag configured for the pen
Cursor		The value of the pen where it intersects the cursor line

- **Properties**: Enables configuration of the properties for the field highlighted in the **Available** or **Visible** list:

Property	Description
Label	Label for the field displayed at runtime
Width	Width for the field (in pixels) at runtime

Property	Description
Align	Alignment of the data displayed in the field
Available at runtime	If this option is enabled, the user can show or hide the field at runtime

- **Maximum size**: Defines the size of the legend in terms of number of lines. For instance, the user might have 8 points being displayed in the trend object, if the maximum size is set to two, the legend will have a scroll bar to allow the user to scroll to the other points.
- **Number of Items**: Number of points (default) displayed on the legend. Users can be enabled to add/remove points at runtime regardless of the value in this field.
- **Selected item**: In this field, a numeric tag can be configured. The object writes the number of the selected line into this tag. In addition, different rows can be selected by writing their values in this tag.
- **Fonts**: Sets the font for the text displayed in the legend.

Advanced

Accessing the dialog

To access the "Advanced" dialog for a specific trend object, first select the Object properties dialog for the screen object on page 353 and click on **Advanced**.

The dialog in detail

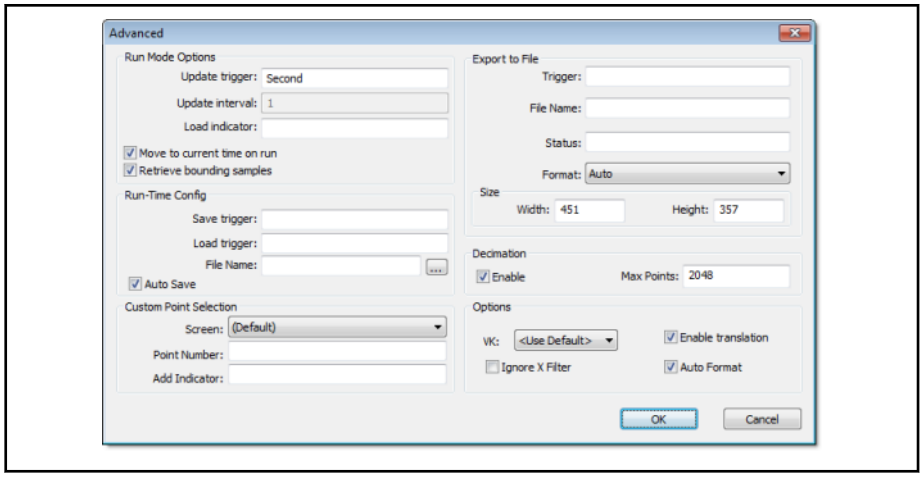


Fig. 9-41: Axes dialog

The "Advanced" dialog contains the following elements:

Pane/element name		Description
Run mode options	Update trigger	If the tag configured in this field changes value, the trend object is updated (refreshed).
	Update interval	If the update trigger is issued and the x-axis is type numeric, the value on this field is added to the minimum and maximum values of the x-axis.
	Load indicator	Enter the name of a project tag. While the trend object is loading external data, the tag receives a value of 1 and if the trend object has finished loading the data, the tag receives a value of 0.
	Move to current time on run	If this checkbox is enabled, the x-axis automatically shifts to the current time if the object is triggered to Play mode at runtime.
	Retrieve bounding samples	If this box is checked, the object retrieves the output data of the object (first points only). Disabling this option can improve the performance, since the output points of the object are not retrieved from the history. However, the object does not draw lines linking the first and last samples to the maximum limit of the object.
Runtime config	Save trigger	The settings of the Trend object modified at runtime can be saved in temporary files. This option can be used as follows: Keep the settings consistent, so when the user closes the screen and opens it again, or restarts your project, the settings configured at runtime are not lost. Create standard settings for different scenarios and load the appropriate configuration at runtime, based on a pre-defined condition or based on the user-selection. If the tag configured in this field changes its value (e.g., toggles), the current settings of the trend object are saved in the temporary file. This command is not available for the thin client. The following settings are saved: **Points:** Color, tag/field (if indirect tags are used), min. scale / max. scale, hide **Axes:** Duration (start time, end time, etc.), grid division, number of markers, enable cursor and current zoom **Toolbar**: Show/hide status of all commands for which no activation tags are configured **Legend**: Visible columns with widths, maximum size, number of items, selected items
	Load trigger	If the tag configured in this field changes its value (e.g., toggles), the settings from the temporary file are loaded and applied to the trend object at runtime. **Note**: After the screen with the configured trend object is saved, the settings are not automatically loaded from the temporary file when the screen is reopened, unless the **Load trigger** command is executed before the screen is closed.
	File name	If this field is left blank, the temporary file is saved in the project `Web` sub-folder with the syntax `ScreenNameObjectIDTrendObject.stmp` (e.g., `MyScreen10TrendObject.stmp`). The thin client station saves/loads the temporary file in the standard "Temp" directory of the operating system (e.g., `\Documents and Settings\CurrentUser\Local Settings\Temp`). A customized file name can be configured for the temporary file in this field or even a string tag between curly brackets, to enable users changing the name of the configuration file dynamically at runtime by changing the value of this tag. If no path is specified, the file is saved in the project `Web` sub-folder by default.
	Auto save	If this checkbox is enabled, the current settings of the trend are automatically saved in the temporary file in case the screen with the configured trend is closed at runtime. If the checkbox is disabled, the settings are saved only if the **Save trigger** command is executed.

Pane/element name		Description
Custom point selection	**Screen**	This interface enables creating custom dialogs to modify or insert pens to the object. Name of the screen that has to be launched if a command is triggered at runtime to modify or insert a new pen to the object.
	Point number	Point number (from the Points dialog, page 354), indicating the point associated with the pen that is inserted or modified during runtime.
	Add indicator	Flag that indicates that an action was triggered to insert a new pen (value 1) instead of modifying a pen that is already visualized (value 0).
Export to file	**Trigger**	If the tag configured in this field changes its value (e.g., toggles), the current state of the trend display is exported to an image file. Toolbar and scroll bar are not included. The legend and time display are included only if the trend display is configured to display them.
	File name	The file path and name of the exported file. If no path is specified, the file is saved in the project directory. If no extension is specified, it is defined by **Format**. **Tip:** A project tag can be specified in curly brackets (e.g., **{tagname}**) to change the file name at runtime.
	Status	The tag configured in this field receives status codes that indicate the success or failure of the export.
	Format	The graphic format of the exported file. If **Auto** is selected, the format is determined by the extension specified under **File name**. If **Auto** is selected but no extension is specified, the default format is BMP.
	Size	The image file is exported at full size by default. However, width and height (in pixels) can be specified.
Decimation	**Enable**	If this option is selected, the trends in the trend object that are configured to display historical data have their data decimated before the trends are drawn. This means for each trend that the x-axis is divided into a number of intervals (determined by **Max points**) and all of the data points within each interval are averaged together to be drawn as a single point. This can improve runtime performance in case there is a large amount of historical data to display. Additionally, trends can be made easier to read. Decimation only works if the trend object is in Stop mode (or Historical mode). When selecting this option, please observe that decimation is carried out for all trends that have been configured for display of historical data. To use this function only for a single trend, use the **Draw mode** in the Options, page 357. **Note**: If Decimation is enabled and the X axis is configured set to numeric instead of date/time, the data used in the X axis have to be properly sorted. For further information, refer to Data source settings, page 383.
	Max points	The maximum number of data points used to draw each trend. Default is 2048.

Pane/element name		Description
Options	VK (Virtual keyboard)	For this object, the virtual keyboard on page 66 type is used.
	Ignore X filter	If this checkbox is enabled, the X filter is ignored to avoid adding WHERE or querying clause to the data source.
	Enable translation	Enable the external translation for the text displayed by this object.
	Auto format	If this checkbox is enabled, the decimal values in the columns on page 369 "Current", "Cursor", "Max", "Min" and "Scale" are formatted according to the virtual table created by the function SetDecimalPoints on page 767: **Note**: For the **Auto format** function to work, decimals formatting on the X-axis has to be disabled, i.e., the **Decimals** in the Axes dialog on page 360 have to be left blank.

9.5.3 Using a Text File as Data Source

The trend display can generate trend charts from any text file where values are organized in columns and lines. The columns should be separated from each other by special characters (usually the comma). Each sample (pair of values representing a point in the graph) is represented by a line. Suppose that the user wants to display a chart with the information in the following table:

X value	Y1 value	Y2 value
0	0	10
1	1	20
2	2	30
3	3	40

One variable represents the X-axis and two variables (Y1 and Y2) represent different lines in the chart. The first step is to convert the data into a text file. If commas are used as separators, the file is displayed as follows:

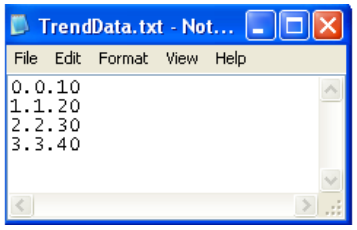


Fig. 9-42: Trend data text

It is highly recommend to save the file in the same folder as the project. By doing so, it is not necessary to specify the entire path and the project will still work, even if it is copied to a different computer.

Once the trend object is added to the screen, double-click on the object to open the "Object Properties" and click on **Axis...**.

Change the "Data Type" of the "X Axis" to **Numeric** and set the ranges as shown in the picture below:

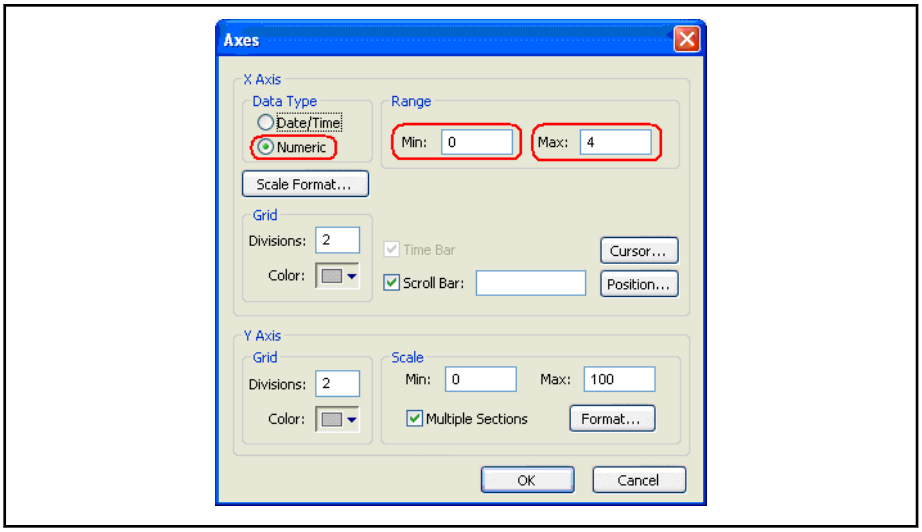


Fig. 9-43: Axes dialog

Click on **OK** and, in the "Object Properties" window, on the **Data Sources...** button. The following window is displayed:

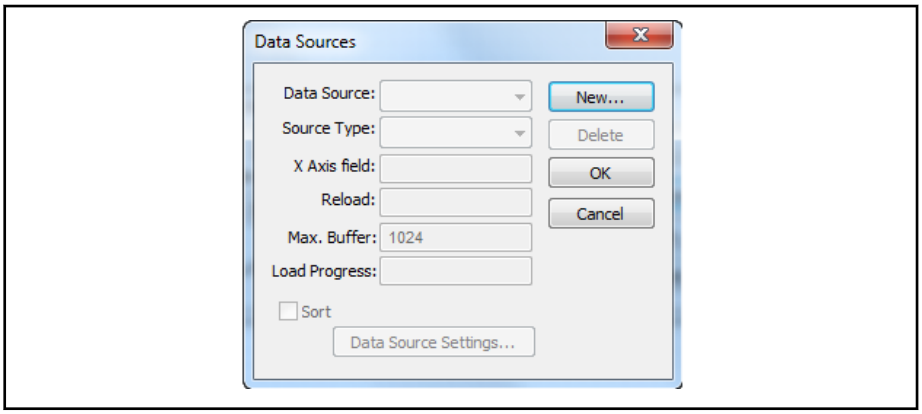


Fig. 9-44: Trend object – Data Sources dialog

A data source has to be created to access the text file. Click on the **New...** button, specify the **Data Source** name **MyTextFile** and click **Create**. Now, the following information is displayed:

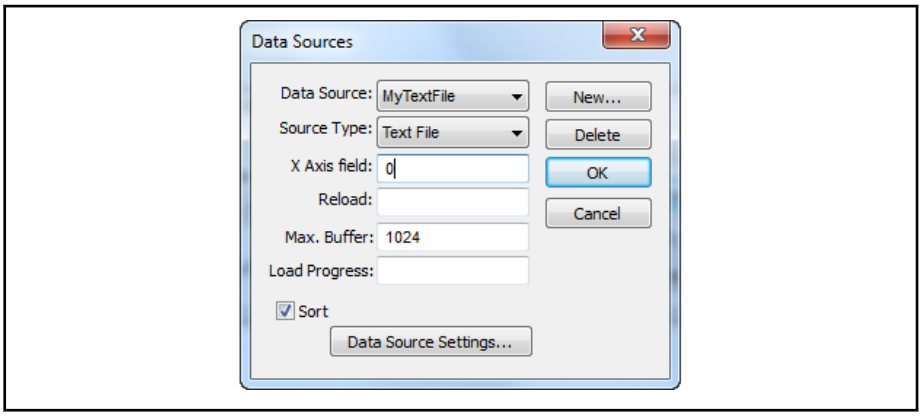


Fig. 9-45: Setting X Axis field to 0

In the **X Axis field**, the column in which the text file represents the X-axis has to be specified. In this example, column 0 is used. Enter "0" for this field and click on **Data Source Settings...**. The following window is displayed:

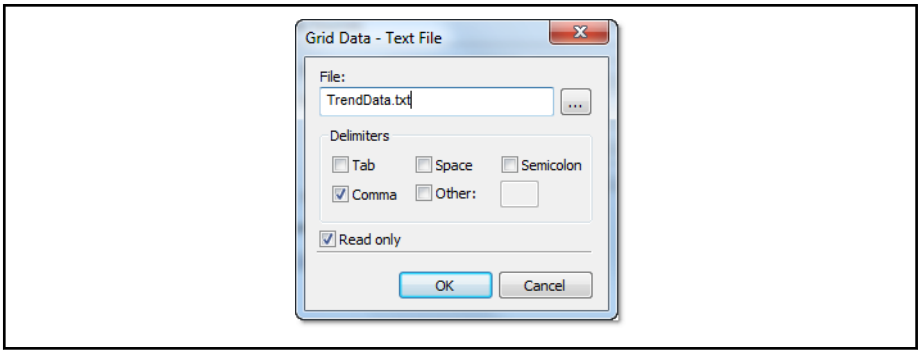


Fig. 9-46: Selecting the text file

If the text file has been copied to the project folder, only the file name has to be specified. Otherwise, enter the complete path of where the file is located (if required, use the browse button). Click on **OK** in this window and on **OK** again to finish the data source configuration and close the "Data Source" configuration window.

Now, Y1 and Y2 have to be defined. These are represented by points on the trend display. Double-click on the trend object to open the "Object Properties" window and click on **Points...**. The next step is to define the points according to the following figure:

After completing these steps, execute the project. The result is to correspond approximately to the following figure:

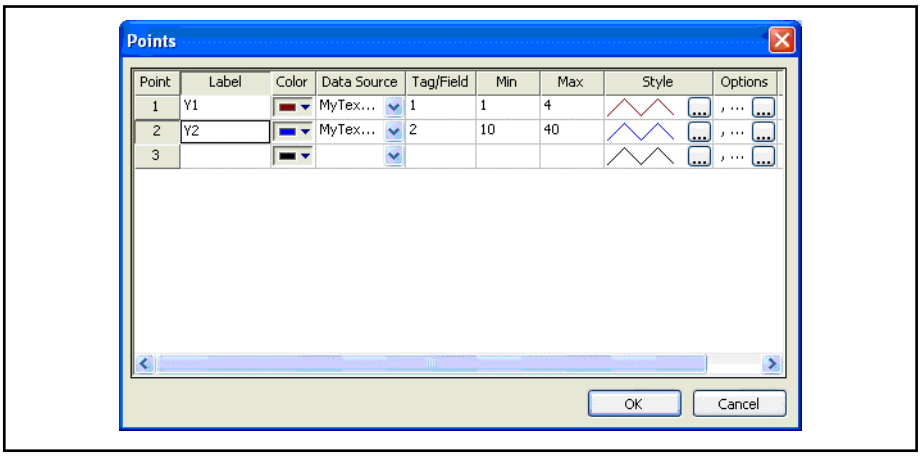


Fig. 9-47: Points table

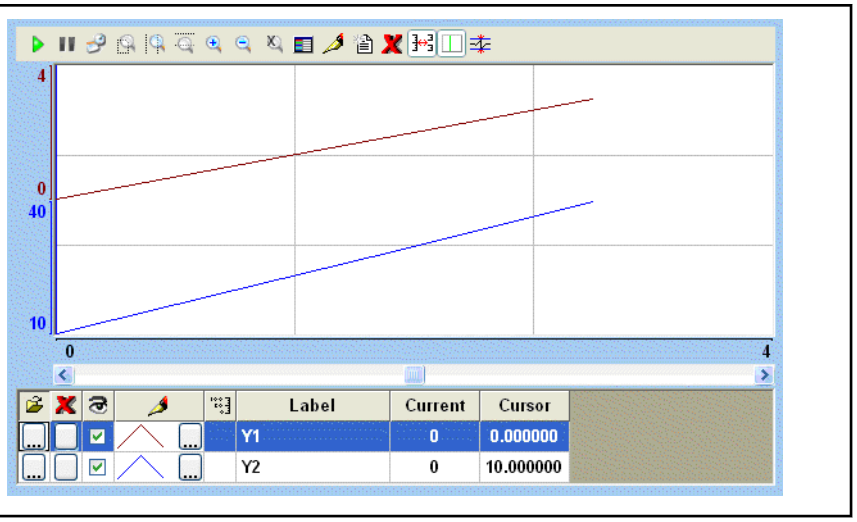


Fig. 9-48: Diagram

9.5.4 Using a Database as Data Source

The trend display can generate trend charts from any relational database that can be accessed through the ADO.Net technology.

This appendix describes access to a Microsoft Access database.

If a different type of database is used almost all definitions apply, however, the connection has to be configured differently. For information on how to configure other databases, please refer to the appendixes in the database interface section of this manual.

If an Access database named "mydata.mdb" is stored under C and a graphic is to be created based on the information in the following table:

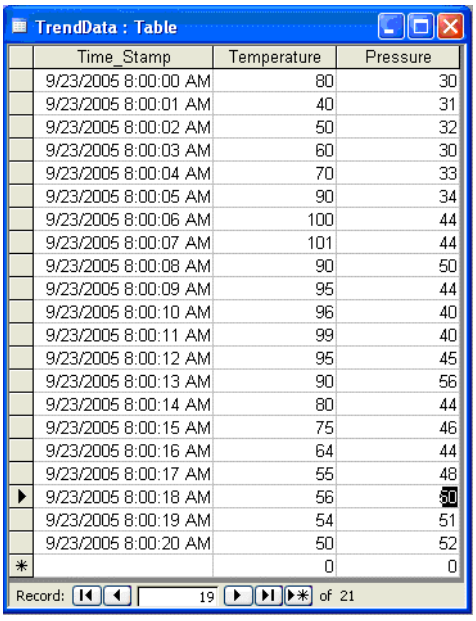


Fig. 9-49: Trend data table

The first step is to add the trend object to the screen. Double-click on the object to open then Object Properties and click on `Data Sources...`. The following window is displayed:

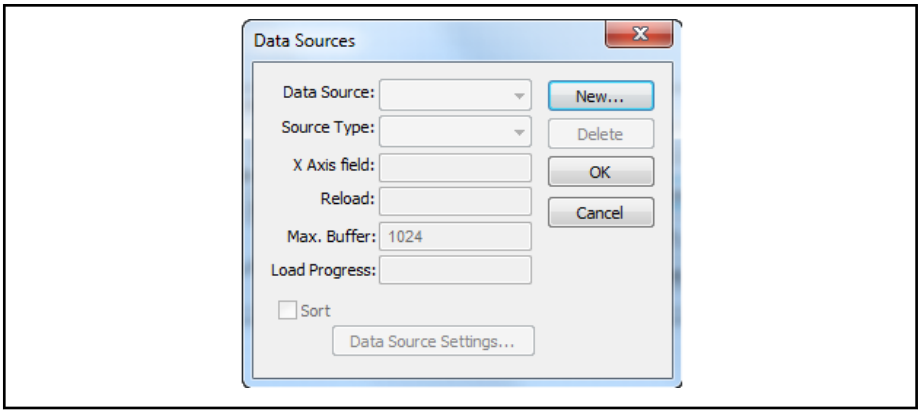


Fig. 9-50: Trend object – Data Sources dialog

To access the database, a data source has to be created. Click the **New...** button, enter the **Data Source** name **MyDB** and click **Create**. Now, the following information is displayed:

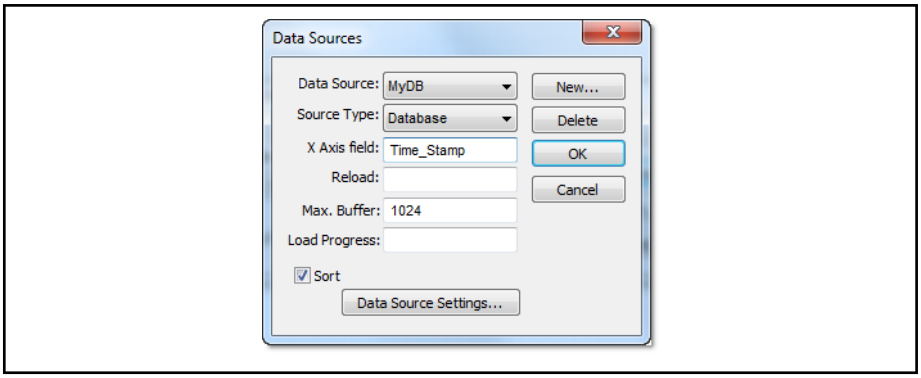


Fig. 9-51: Setting the X Axis field to Time_Stamp

Change the **Source Type** to **Database** and specify "Time_Stamp" in the **X Axis field**. Click on the **Data Source Settings** button. The following window is displayed:

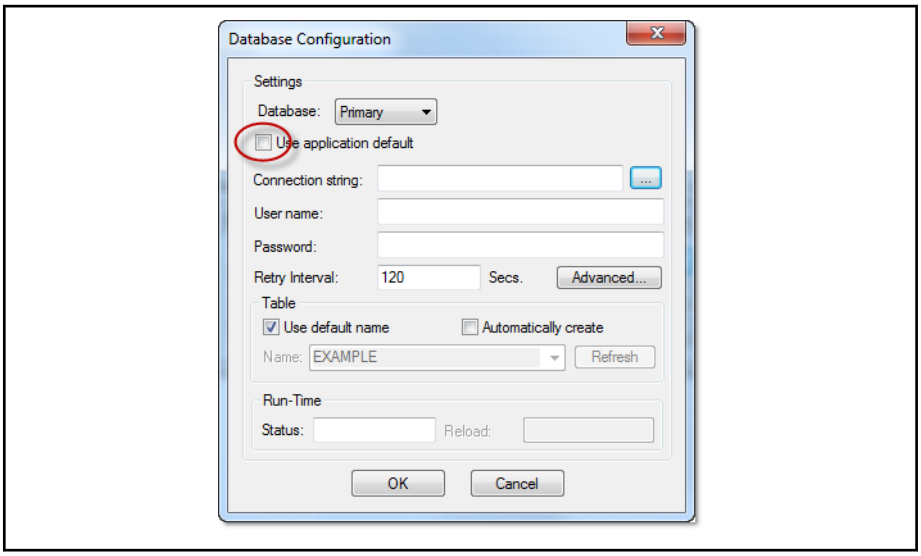


Fig. 9-52: Disabling the Use application default option

Disable the checkbox **Use application default** and click on the browse button **...** to configure the Connection string. The following window is displayed:

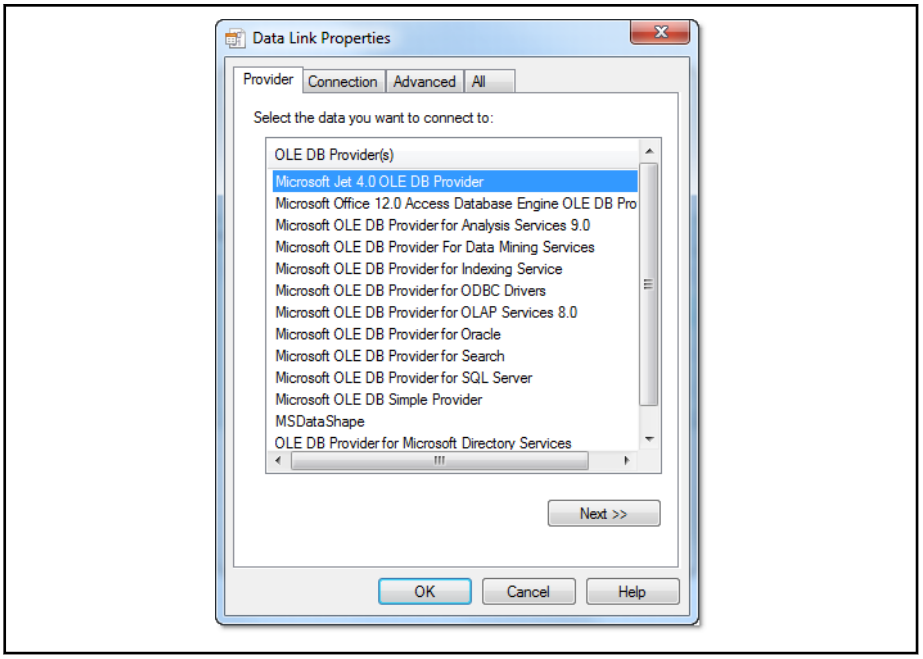


Fig. 9-53: Selecting the OLE DB Provider

Select the Microsoft Jet 4.0 OLE DB Provider and click on **Next »**. In the following window, the database path is to be specified:

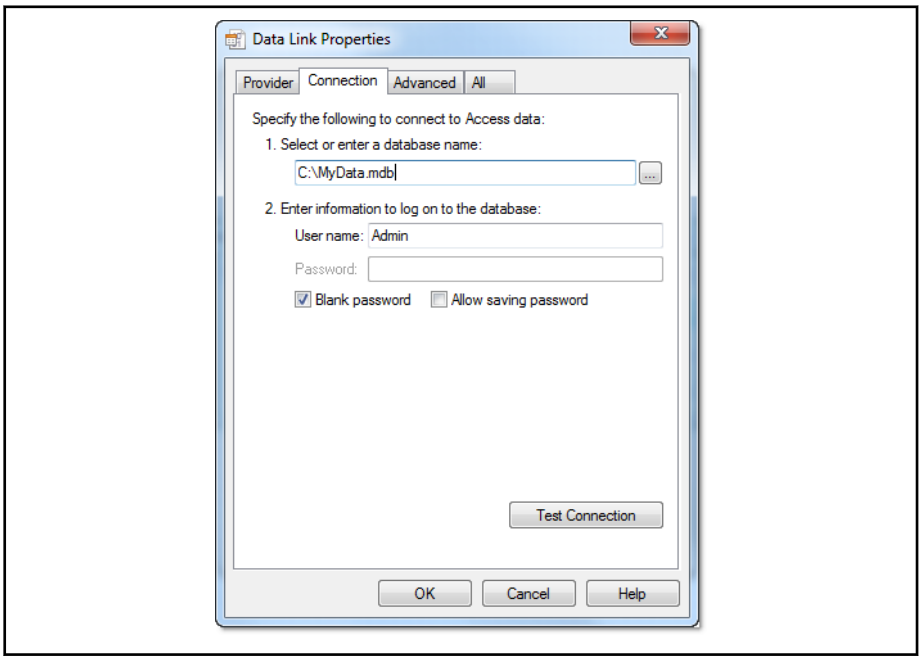


Fig. 9-54: Selecting the database file

Click **OK** to finish the Connection string configuration. Disable the option **Use default name** and select the table from the database as shown in the following:

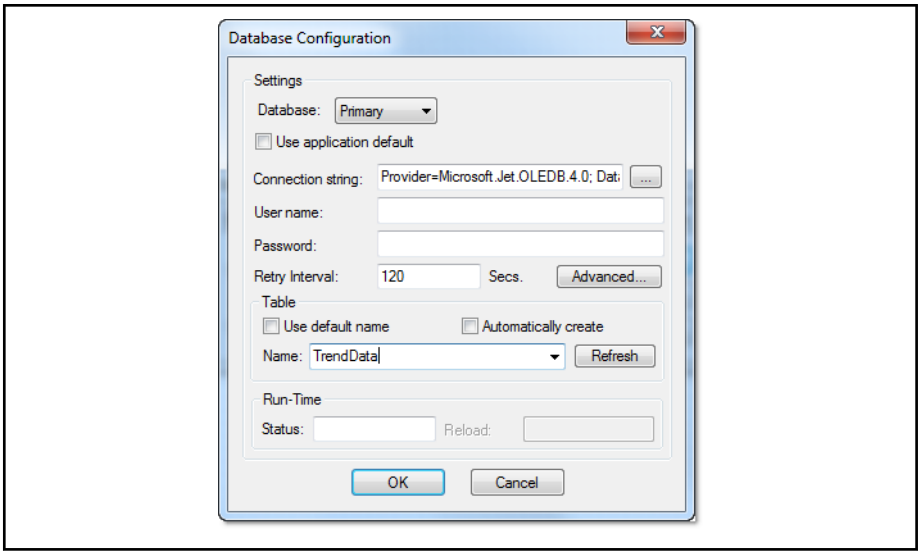


Fig. 9-55: Selecting the table in the database

Click on **OK** in this window and on **OK** again to finish the data source configuration and close the "Data Source" configuration window.

Now Temperature and Pressure have to be defined. These are represented by points on the trend display. Double-click on the trend object to open the Object Properties window and click on **Points...**. The next step is to define the points according to the following figure:

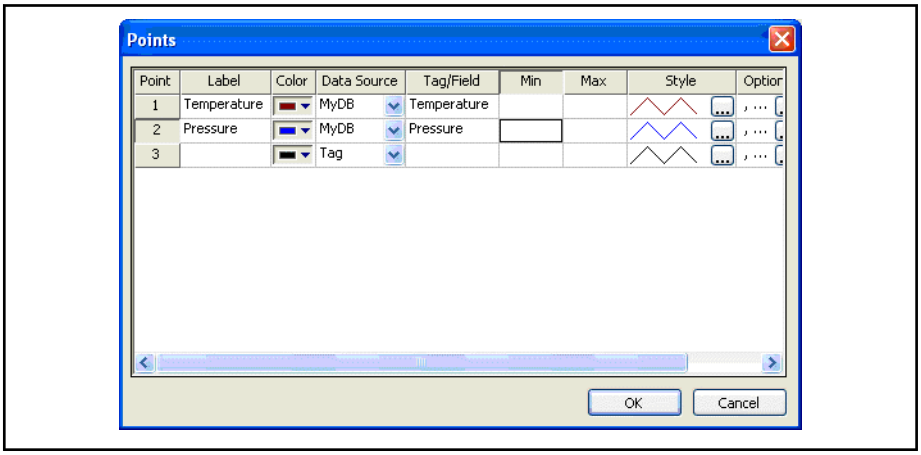


Fig. 9-56: Points table

If the trend display is executed, it starts with the current date/time. To display the data in the chart the start date/time has to be properly configured as follows:

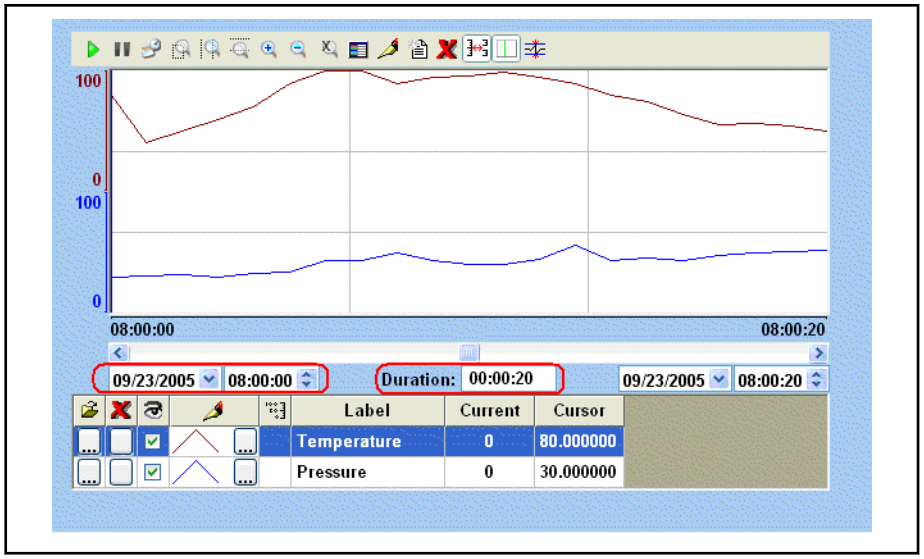


Fig. 9-57: Diagram

9.6 Grid object

The table object enables reading/writing data in the table format from/to the data source configured in the object.

To draw a table object, proceed as follows:

1. In the "WinStudio screen tools" tool window, click on the element "Table".
2. Click on the screen, left-click and drag the mouse across the screen to create a box of the desired size (while holding down the mouse button).
3. Release the mouse button to display the table object.

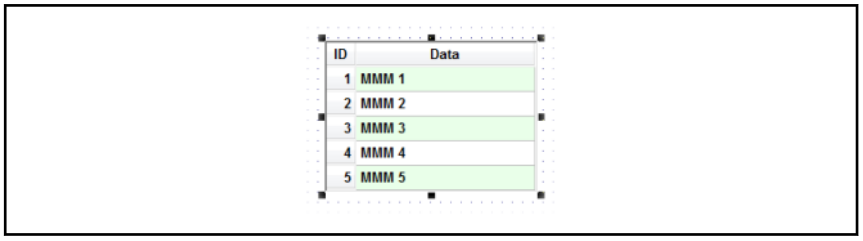


Fig. 9-58: Example of a table object

This object cannot be moved behind active objects or shapes on a project screen. In other words: it must always be at the front.

For additional information, refer to In the foreground/background, page 269.

Right-click on the table object and select **Properties** from the menu. The “Object properties” dialog is displayed. Use this dialog to configure the parameters of the table object:

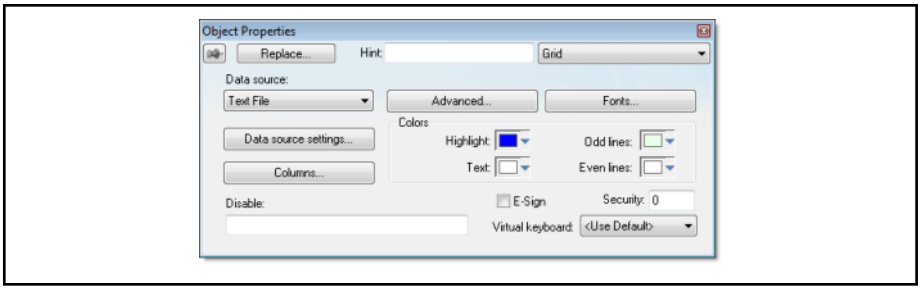


Fig. 9-59: Object properties: Table

- **Data source**: Select the data source type. The object supports three data sources:

Data source	Description
Text file	Displays data from a text file in the ASCII or Unicode format (e.g., CSV text files).
Class tag	Displays values from a class tag, where the class elements of the tag are fields (columns) of the table object and each array position represents one line of the table object.
Database	Displays data from an SQL relational database, using ADO (ActiveX Database Object) to exchange data with the database.

- **Data source settings**: Click to launch the Data dialog on page 383, where a data source for the grid object can be specified.
- **Columns**: Click to launch the Columns dialog page 385, where the settings (such as label, column, width, etc.) for the columns of the grid object can be configured.
- **Advanced**: Click to launch the Advanced dialog on page 388, where verious settings for the grid object can be specified.
- **Fonts**: Click to launch the "Fonts" dialog, where you can configure the font settings for the text displayed in the Grid object.



By default, the same text color is used for the header as for the main part of the grid. To change the text color for the header, edit your project file (`projectname.APP`) manually or add the following setting:

```
[Objects]
GridHeaderTextColor=value
```

`value` has to be a hexadecimal RGB color value, for example FF0000 for red. Please note that this setting is applied to all headers of grids in your project.

- **Highlight**: Select a background color for the selected line at runtime.
- **Text**: Select a text color for the selected line at runtime.
- **Odd lines**: Select a background color for the odd lines.
- **Even lines**: Select a background color for the even lines.
- **Disable**: In this field, an expression can be entered to disable data input or action by the user.
- **E-Sign**: If this option is enabled, the user is prompted to enter the electronic signature before entering or modifying data on the object.
- **Security**: Enter the security access level required for the object/animation.

- **VK**: Select the "Virtual keyboard" type used for this object. The option <Use default> selects the default virtual keyboard configured in the "Viewer" settings (**Viewer** tab in the Properties dialog of the "WinStudio" folder).

  You can also specify a different virtual keyboard for this grid object.

9.6.1 Data dialog

Using this dialog, the data source for grid objects can be configured.

Grid data – Text file

If the **Data source** type is set to text file, the following settings are available for configuration:

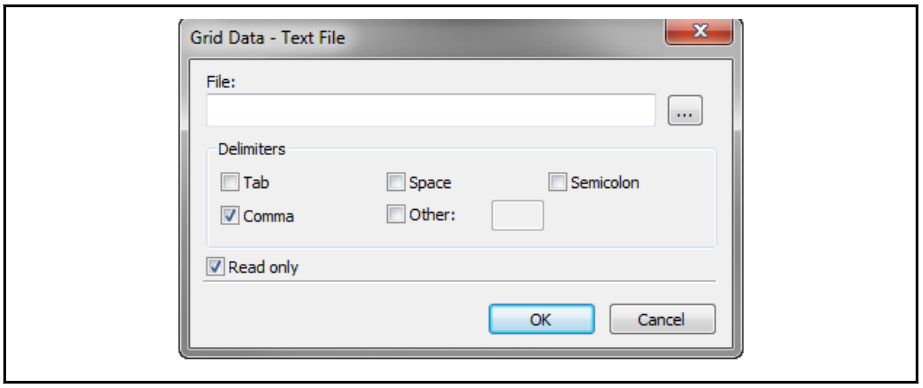


Fig. 9-60: Grid data - Text file

- **File**: Enter the name of the source text file. Enter the file name and path or click the **...** button to browse for it. (If the file is stored in the project folder, the path does not need to be specified in the name.)

Tag names can be entered in curly brackets (e.g. **{TagName}**) in the `File` field

- **Delimiters**: Set the delimiter(s) used in the data source file. For instance, if the data is to be read from a CSV (comma separated values) file, select the **Comma** option. A custom delimiter can be selected by checking the **Other** option and typing it in the field beside it.
- **Read-only** checkbox: When this option is checked, the grid object will only read data from the specified file. The object does not write anything to the file.

Grid data – Class tag

If the **Data source** type is set to database, the following settings are available for configuration:

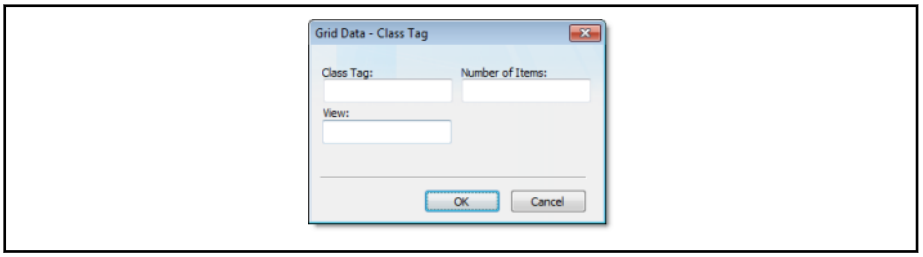


Fig. 9-61: Grid data – Class tag

- **Class tag**: Enter the name of the main class tag source. (Do not specify a specific member of the class tag.) The initial array position can be specified in this field (e.g., "Mytag[10]"); otherwise, 0 (zero) is used as initial position by default.

- **Number of items**: Enter the number of Array positions of the **Class tag** to be displayed.
- **View**: When the tag configured in the optional field changes value (e.g., toggles) at runtime, the grid object launches a dialog, allowing the user to show/hide each column or modify their positions.

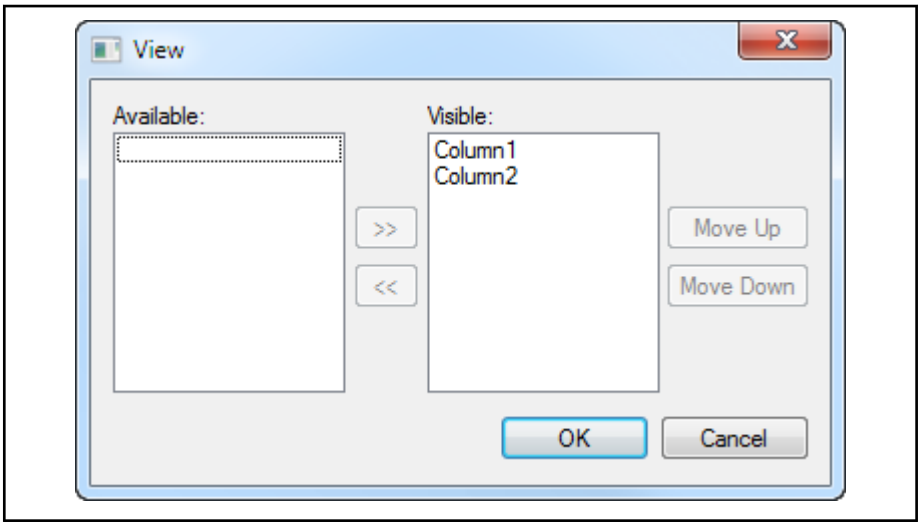


Fig. 9-62: Dialog: View

Grid data – Standard database configuration

If the **Data source** type is set to database, the following settings are available for configuration:

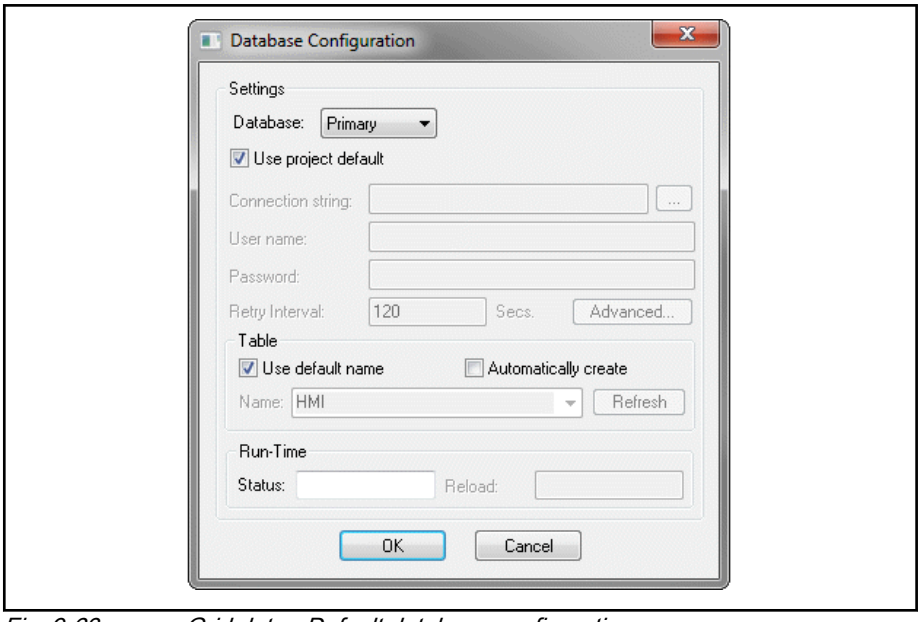


Fig. 9-63: Grid data - Default database configuration

Please refer to the Database configuration dialog page 563 for further information on this dialog.

The configuration of a secondary (redundant) database for a grid object ensures that the content of the object is read only - this means that the object can read and display data from the primary or secondary database as required but changes in the object cannot be written in any of the databases. This also means that the **Insert trigger** and **Save on data change** options do not have any effect.

For further information on these options, refer to Advanced dialog, page 388.

9.6.2 Columns dialog

To configure the settings for each column displayed by the grid object on page 381 at runtime, proceed as follows:

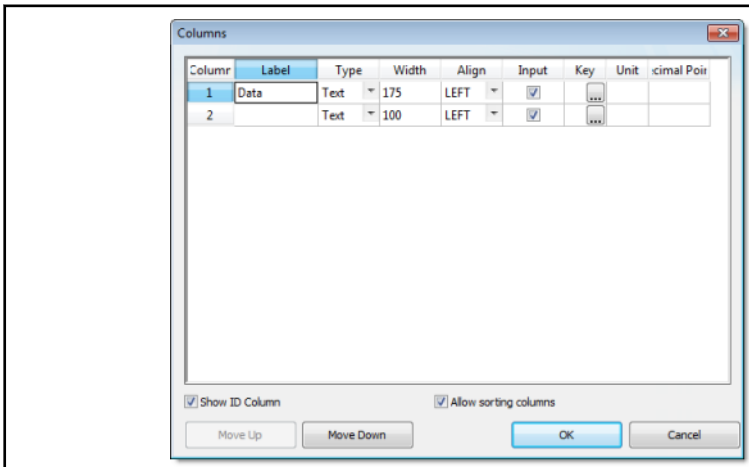


Fig. 9-64: "Columns" dialog

- **Column**: The ID number defines the position of the column in the table.
- **Label**: Enter a title for each column, which will display as the heading (first) row of the Grid object.

To dynamically change the title during runtime, configure a project tag in curly brackets (e.g. `{MyTag}`). If the label is left blank, the width of the entire column is set to 0. This option is useful to hide columns at runtime.

- **Member**: Enter the name of the class member to connect this column to. If this field is left blank, the text configured in the **Label** field is used as standard member name.

  This setting is only available if **Data source** is set to **Class tag**.
- **Field**: Enter the name of the field (column) in the SQL relational database to which the grid object is linked. If this field is left blank, the text configured in the **Name** field is used as field name.

  This setting is only available if the **Data source** type is set to database.
- **Type**: Select the type of interface that is used in the column. The following options are available:

Type	Description
Text	Displays alphanumeric values
Numeric	Displays numeric values
Screen	Displays the picture (`*.bmp` or `*.ico` format) of the data source. For instance, if the value from the data source is `MyFile.bmp`, the grid object will display the picture from the file `MyFile.bmp` stored in your project folder. The picture will be automatically resized to fit the cell of the grid object. The picture file(s) has/have to be stored in the `Web` sub-folder of the project folder to support this feature on thin client stations. CEView projects support pictures in bitmap format (`*.bmp`), however, not in icon format (`*.ico`).
Checkbox	Displays a checkbox. This checkbox is disabled if the value read from the file is 0, <ZERO> or "FALSE"; otherwise, the checkbox is enabled. By default, "WinStudio" uses the value 0 for disabled and the value 1 for enabled.
Time	Displays the value in the time format (e.g., `HH:MM:SS`). This setting is only available if the **Data source** type is set to **database**.
Date	Displays the value in the date format (e.g., `MM/TT/JJJJ`). This setting is only available if the **Data source** type is set to **database**.
Date/Time	Displays the value in the date/time format (e.g., `MM/TT/JJJJ HH:MM:SS`). This setting is only available if the **Data source** type is set to **database**.
Date - UTC	Displays the value in the date format (e.g., `MM/TT/JJJJ`). For the date, the Coordinated Universal Time is assumed and it is automatically converted to the current time zone in the Viewer / Thin client module. This setting is only available if the **Data source** type is set to **database**.
Date - UTC	Displays the value in the date format (e.g., `MM/TT/JJJJ`). For the date, the Coordinated Universal Time is assumed and it is automatically converted to the current time zone in the Viewer / Thin client module. This setting is only available if the **Data source** type is set to **database**.
Date/Time - UTC	Displays the value in the date/time format (e.g., `MM/TT/JJJJ HH:MM:SS`). For the date and the time, the Coordinated Universal Time is assumed and they are automatically converted to the current time zone in the Viewer / Thin client module. This setting is only available if the **Data source** type is set to **database**.



- If the **Data source** type is set to database, it has to be ensured that the **Type** for each column configured in the object matches the type of the respective field in the database.
- If the **Data source** type is set to database, valid SQL statements can be configured directly in the field (e.g., `List(DISTINCT [Cell_Name]) AS [Cell_Name]`).

  Tag names can also be configured in curly brackets to modify this setting at runtime (e.g., `{MyFieldName}`).



If the column type is set to Picture, the grid object displays a default icon ( ) in case the picture file is not found at runtime. To configure a different picture to be displayed when the file is not found, copy the picture file to the "Web" sub-folder of the project folder and configure its name in the "project_name.app" file as follows:

```
[Objects]
GridDefaultPicture=PictureFileName
```

- **Width**: Enter a column width in pixels

  If the width is to be dynamically adjusted during project runtime, enter the name of the Integer tag in curly brackets (e.g. `{Column1Width}`). If the value of the tag changes, the width is changed accordingly.
- **Align**: Select an alignment for the data shown in the column. There are three options: Left, Right or Center
- **Input**: Clicking on (check) the checkbox enables users to enter data in this column at runtime
- **Key**: Use this field to designate a shortcut for sorting the values. A shortcut is a combination of keys simultaneously pressed on a keyboard (e.g., <CTRL>+<C>, <CTRL>+<V>, etc.). This option is especially useful when creating projects for runtime devices that do not provide a mouse or touchscreen interface and only have a keyboard to access the project at runtime.
- **Unit**: Enter the name of the engineering unit (i.e., the unit of measurement), if any, that applies to the data displayed.

  String tags can also be entered using the `{tagname}` syntax to change the **Unit** value during runtime (e.g. `{Column1Unit}`). If the value of the tag changes, the unit is changed accordingly.
- **Decimal points**: Enter the number of decimal points to be displayed.

  Integer tags can also be entered using the `{tagname}` syntax to change the **Decimal points** value during runtime (e.g. `{Column1Decimals}`). If the value of the tag changes, the number of decimal points is changed accordingly.
- **Show column ID**: Check to automatically display the number of each line.
- **Allow sorting columns**: Check to enable the user to sort the values in the columns at runtime, either by clicking on the label or by using the shortcut configured for each column.

  This option is disabled if the **Show header** option in the Advanced dialog on page 388 is disabled.



If the **Data source** type is set to **Class tag** and the Columns dialog is left blank, the object displays the values from all members of the class tag with the following default column settings:

- Label = <The name of the member from the class tag>
- Type = Text
- Width = <Minimum size for the display of the member name of the class tag in the header of the grid object>
- Align = Center
- Input = Enabled (checked)
- Key = <None>
- Unit = <Unit of the class tag member>

The unit of a class member or tag is set using the tag properties on page 122 tool.



Use the **Move up** and **Move down** buttons to reorder the display of the columns.

9.6.3 Advanced dialog

In this dialog, the advanced settings for table objects can be configured.

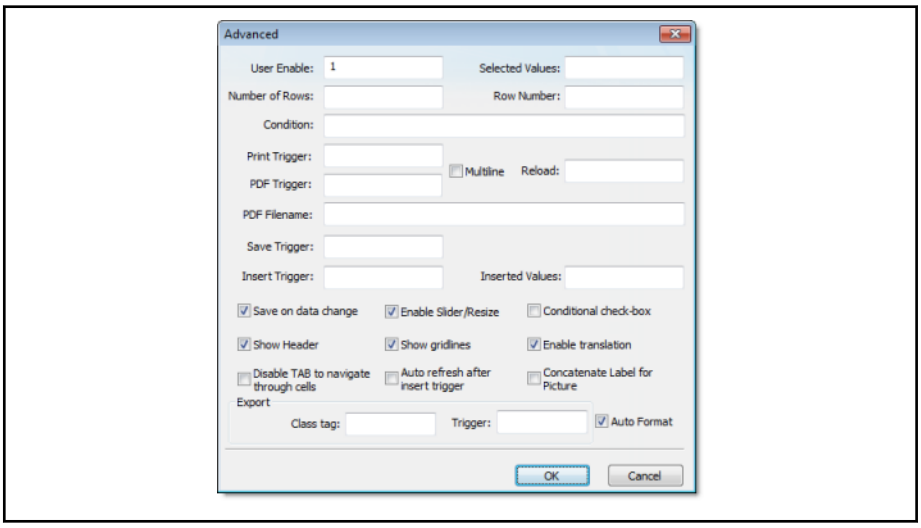


Fig. 9-65: "Advanced" dialog

- **User enable**: If the value of this tag is TRUE (different from 0), the user can select different lines of the object by clicking on them at runtime. This field can be configured with a tag or with a numeric value.
- **Selected values**: The values from each column of the selected line are written to each position of the array tag configured in this field. Additionally, the value of the cells currently selected in the table object can be modified by changing the value of the array tag configured in this field. In this field, the initial position (offset) can be configured.
- **Number of rows**: The table object writes the number of rows currently available in the table object to the tag configured in this field.
- **Row number**: The table object writes the number of the row currently selected at runtime. In addition, different rows can be selected by writing their values in this tag.
- **Condition**: Enter an expression to filter the table data. Only rows that match the expression will be displayed. The expression has to comply with the following syntax:

  ```
  Column Operator Value
  ```

  For example…

  ```
  ColumnX > 200
  ```

  If **Data source** (in the Table object properties dialog on page 381) is set to text file or class tag, the "Column" contains the value specified in the Label. If **Data source** is set to database, the column contains the value specified in the field. (In this case, if the field is left blank, the column value specified is the label.)

  Database expressions have to be formatted like SQL Where statements. The following table illustrates the operators to be used:

Condition expression operators

Comparison	Data source is text file…	Data source is database…
equal to	=	LIKE
not equal to	<>	NOT LIKE
Placeholder, single character	?	_
Placeholder, unlimited characters	*	%

As such, the following expression for text file…

```
C1 = 'ab?d'
```

... means the same as the following expression for database...

```
C1 LIKE 'ab_d'
```

Finally, several expressions can be combined simultaneously in the **Condition** field using the logic operators "AND", "OR" and "NOT".

```
ColumnAge > '10' OR ColumnName = 'John' AND ColumnDate >
'05/20/2003'
```



Tags can be configured between curly brackets `{TagName}` in the **Condition** field to change the filtering condition at runtime.

- **Print Trigger**: If the tag configured in this field is toggled, the current state of the table object is sent to the default printer.
- **PDF trigger** field: If the tag configured in this field is toggled, the current state of the table object is saved as a PDF file at the location specified under **PDF Filename**.
- **PDF file name** field: Enter a complete file path and name where the PDF file is to be saved. Tag names using the **{tag}** syntax can also be entered.



PDF trigger and **PDF file name** are not supported in projects running on Windows Embedded or thin client systems.

- **Multiline** checkbox: If this option is enabled, the print output or PDF is formatted according to the available column space, and the text within each cell is wrapped so that it is completely displayed.
- **Reload**: If the tag configured in this field is toggled, the object reloads the data from the data source and displays it.
- **Save trigger**: If the tag configured in this field is toggled, the data source (text file or database) is updated with the current values of the grid object. This field is not available if the data source type is set to class tag, as the values are automatically updated in the tags as cells are changed in the table.
- **Insert trigger**: If **Auto refresh after insert trigger** is enabled (checked), the tag configured in this field is used as a trigger to refresh the database table. If the value of the tag changes, a new row is added to the table and the values of the array configured in the **Inserted Values** field are automatically inserted.

- **Inserted values**: If the **Insert trigger** is used, the array tag configured in this field provides the values to be inserted. This field may only contain one array tag, however, it can be of any size.
- **Save on data change**: If this option is enabled, the values are updated on the data source (text file or database) as soon as the user enters a new value into the table at runtime. This option is disabled if the data source type is set to class tag, as the values are automatically updated in the tags as the cell values are changed in the table.
- **Enable slider/Resize**: If this box is disabled, it is neither possible to scroll the list by dragging the slider button nor to change the cell size at runtime.
- **Conditional check-box**: If this option is enabled, checkboxes on the Grid cannot be disabled at runtime, unless all preceding checkboxes in the same column are already disabled. This option is particularly useful if users are required to follow a pre-defined sequence. This field is not available if the data source type is set to class tag.
- **Show header**: If this option is enabled the header of the table object is visible at runtime, displaying the label of each column.
- **Show grid lines**: If this option is enabled, the gridlines of the table object are visible at runtime.
- **Enable translation**: If this option is enabled, the text displayed by the grid object is provided to the translation tool at runtime.
- **Disable TAB to navigate through cells**: If this option is enabled, it is possible to navigate the cells of the table object with the arrow keys instead of the Tab key. The Tab key is to be disabled for navigation if it is used for switching to the next screen object that can be activated.
- **Concatenate label for picture**: If this option is enabled, the reference name for the picture is the result of the concatenation of the name in the Field column with the value of the Label column. The result is <Designation>_<Field value>.
- **Export**: This interface enables exporting data from the table object to a class array tag, regardless of the data source selected for the object. The following fields have to be configured to support this function:

Field	Description
Class tag	Enter the main tag name of the class array tag that is to receive the exported values. Each row in the grid object is exported to one array position of the array tag by the respective column labels. The initial array position can be configured in this field; 0 is the default.
Trigger	If the tag configured in this field changes its value (e.g., toggles), the data is exported from the table object to the class array tag configured in the Class Tag field.



The export feature is an uncomplicated and powerful tool to transfer data from different data sources to tags. After exporting the data to tags, different tasks are available to manipulate the data, such as the FileWrite on page 736 function or the Recipe or Report tasks to save data in text files (e.g., CSV files).

- **Auto format**: If this option is enabled, decimal values in **Numeric** columns are formatted according to the virtual table created by the function SetDecimalPoints on page 767: This option only works in columns in which no **decimal points** are yet configured. For further information, refer to Grid object: Columns dialog on page 385.

10 Background tasks

Background tasks are, as the name implies, project features that run in the background, as opposed to the graphical screens for user interaction.

The background tasks are executed by the background tasks module (see execution tasks, page 91) and they are defined by task worksheets in the Project Explorer.

10.1 Alarm worksheet, general information

The Alarms folder enables the configuration of alarm groups and tags related to each group. The Alarm worksheet defines the alarm messages generated by the application. The primary purpose of an alarm is to inform the operator of any problems or abnormal condition during the process so he can take corrective action(s).

The Alarm worksheet is executed by the background task module (refer to Execution tasks, page 91). It manages the state of all alarms and saves the alarm messages to the history if configured. The Alarm worksheet itself, however, does not display alarm messages at runtime. To display alarms, the Screen object “Alarm/event control” on page 331 in the "WinStudio screen tools" tool window has to be inserted into a WinStudio screen and configured.

To create a new worksheet, open the context menu of the "Tasks" folder and select **Add ▸ Alarm**.

To edit an existing alarm worksheet, double-click the element in the Project Explorer.

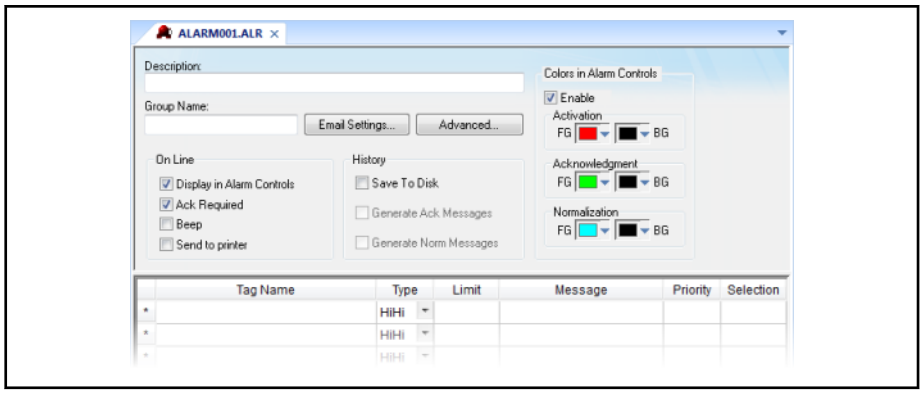


Fig. 10-1: Alarm worksheet

Multiple alarm groups (worksheets) can be created and each group can be configured with independent settings, such as message colors, history log enabled/disabled, etc.

Each alarm worksheet is composed of two areas:

- **Header:** Settings applied to all tags and alarms configured in the same alarm group. These settings enable configuration of the formatting of the messages and actions that have to be triggered based on alarm events (e.g., print alarms, send alarms by email, etc.). For further information, refer to header settings on page 314.
- **Body:** Selecting tags and defining conditions on which alarm messages are created. For further information, refer to body settings on page 319.

- Alarm groups can be configured to send notifications by email automatically, based on alarm events. For further information, refer to email settings on page 315.
- The alarm properties associated with each tag (configured in the body of the alarm group) can also be edited on the Tag properties on page 150 dialog (**Properties** in the "Cross references" tool window). However, before associating a tag with an alarm group, it is necessary to create the alarm group and to configure the settings on its header. These will be applied to all tags associated with the group.
- As of "WinStudio" v6.1+SP2, the Alarm task has been modified to avoid automatically acknowledging alarms by another alarm. For example, the Hi (Lo) alarm is not to be automatically acknowledged when the HiHi (LoLo) alarm becomes active. To enable the this behavior, set the following key in the project file (`.APP`):

```
[Alarm]
UseLegacyPriorityAck=1
```

The settings configured in the body of each Alarm worksheet are stored in the Tags Database archive(s). Therefore, changes to the tags database may affect the content of the alarm worksheets (body).

Notice that each tag/type cannot be available in more than one alarm group simultaneously because the alarm group is a property associated to each tag/alarm type (e.g., `tag: Level; alarm type: Hi; alarm group: 2`).

10.2 Trend worksheet, general information

Data sets to record data series can be configured in the "Trend" folder. They can be used to represent trend curves. If required, declare in a driver worksheet from which variables the values are to be saved in a file and about the configuration of historical data for trend curves. WinStudio saves the recorded values in a binary history file (*.hst). The values can be displayed together with the current variable values in a trend curve.

The Trend worksheet is executed by the background task module (refer to Execution tasks, page 91). It manages saving of trend data in the history, however, does not display this data to the operator. To display trend data, the Trend control object on page 353 in the "WinStudio screen tools" tool window has to be inserted into a WinStudio screen and configured.

To create a new trend worksheet, open the context menu of the "Tasks" folder and select "Trend" in the "Add" menu.

To edit an existing trend worksheet, double-click it in the Project Explorer.

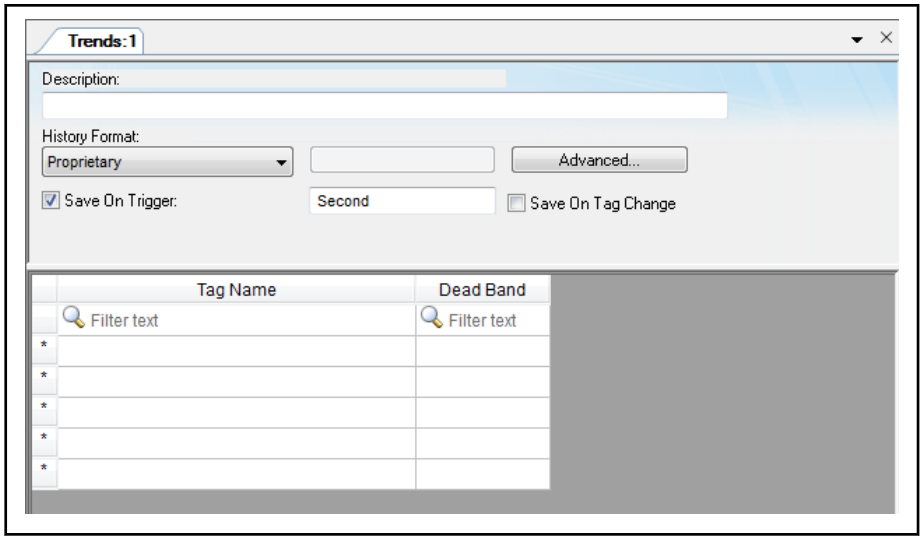


Fig. 10-2: Trend worksheet

The trend worksheet is divided into two areas:

- Header area (top section) containing information for the whole group
- Body area (bottom section) to define each tag in the group. This section contains several columns (only two are shown in the preceding figure).

Use the header parameters on this worksheet as follows:

- **Description** field: Enter a description of the worksheet for documentation purposes.
- **History format**: Click to select a trend history format. The following options are available:
  - **Proprietary**
    - **File format:** Binary
    - **Default path:** …\project_name\Hst\GGJJTTMM.HST, where:
      - YY = two last digits of the year
      - MM = month
      - DD = day

☞ "WinStudio" provides the HST2TXT.EXE and TXT2HST.EXE programs to convert trend history files from binary (.hst) to text (.txt) and vice versa. For further information on these programs, refer to Converting trend history files from binary to text on page 345 and Converting trend history files from text to binary on page 346.

  - **Database**
    - **Database type:** Selected by the user
    - **Default table name:**TRENDGGG (GGG = trend worksheet number; e.g., TREND001 for the trend worksheet 001)

    For further information on the structure of the database table that "WinStudio" uses to save history files, refer to Database interface on page 553.

String tags can be specified in many fields of the trend worksheet to change these values at runtime. However, this may influence how these values are stored in the trend history:

- If the history format is **Proprietary**, the value of the String tag is converted to a numerical value (if possible) and then saved to the history file. If numeric conversion is not possible, a value of 0 is saved.
- If the history format is **Database**, the actual value of the String tag is saved in the database.

- **Database Configuration**: Opens the Database configuration on page 563 dialog, to enter the required settings to link the project to an external SQL relational database for the purpose of saving the trend history.
- **Save on trigger** checkbox and field: Click on (enable) the checkbox and type a tag name to save trend samples if someone changes the specified tag. (The tag change can be an event from the Scheduler.)
- **Save on tag change** checkbox: Click (enable) to always save the trend sample in case of a value change in any of the tags from that group.
- **Advanced**: Click to display the "Trend advanced settings" dialog. For information on completing the fields in this window, refer to Batch history configuration on page 346.

Use the body parameters on this worksheet as follows:

- **Tag name** field: Enter the variable name to be saved in the history file.
- **Dead band** checkbox: Enter a value to filter acceptable changes when **Save on tag change** is used.

  Example:

  The dead band has a value of 5. If the tag value is set to 50 and changes to 52, the system does not register this change in the database as it is smaller than 5. If the change is at least 5, the new value is saved in the history file.
- **Field**: Name of the field in the database where the tag is stored. If this field is left blank, the name of the tag is used as the tag name. For array tags and classes, the characters "[", "] " and " . " replaced by " _ ".

  Examples:

Tag name	Default field
MyArray[1]	MyArray_1
MyClass.Member1	MyClass_Member1
MyClass[3].Member2	MyClass_3_Member2

The trend task can accept only up to 240 tags in a single worksheet. If more than 240 tags are manually configured in the same worksheet, the Trend task generates an error on running the finished project.

Also refer to:

- Converting trend history files from binary to text on page 345

- Converting trend history files from text to binary on page 346
- Creating batch history on page 346
- Configuring a default database for all task histories on page 567

10.3 Recipe worksheet

Recipe worksheets are used to load tag values from or save tag values to an external file at runtime. It is typically used to execute process recipes, however, any type of information (such as operation logs, passwords, etc.) can be stored in the external file.

To create new recipe worksheets, open the context menu of the "Tasks" folder and select **Add...** ▸ **Recipe**

To edit an existing recipe worksheet, double-click on it in the Project Explorer.

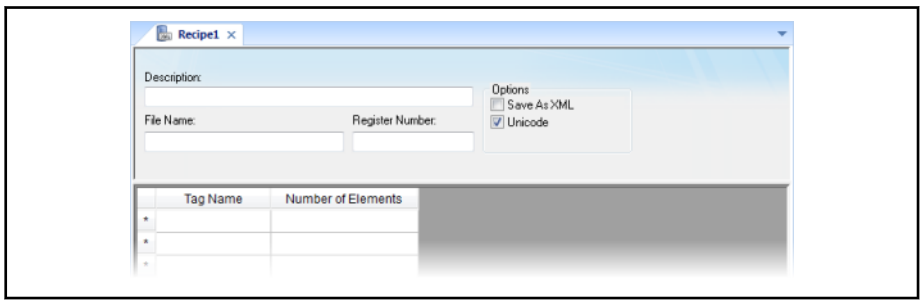


Fig. 10-3: Recipe worksheet

The worksheet is divided into two areas:

- "Header" area (top section) containing information for the whole group
- "Body" area (bottom section) to define each tag in the group.

Use the header parameters on this worksheet as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.
- **File name**: Enter the name of the external file, using static text (`File1`) or indirect tag (`{FileNameTag}`).
- **Register number**: Enter a tag to define the register number to be read or written into a DBF file. (This field is for legacy systems only and is no longer used.)
- **Save as XML**: Enable (check) to save information in XML format or disable (uncheck) to save in DAT format. Use a second recipe worksheet to load information from a **.DAT** file to different tags. In contrast, information in **.XML** files have to be loaded to tags with the same name as the tag containing the original data.

☞ Similar to HTML pages, the web server has to be executed to view XML data from the web. Unlike the HTML pages in the runtime system, XML pages do not require the project to be running to view the XML data. (To view XML data, Internet Explorer version 5.0 or higher is required.)

- **Unicode**: Click (enable) to save the recipe in Unicode format (two bytes per character) or (disable) to save the recipe in ANSI format (one byte per character).

Worksheets can be saved under any name (a predefined file name is not required). A configuration file using the default extension **.RCP** (or **.XSL** if **Save as XML** is selected) contains the recipe configuration and the **File name** field contains the data file name to be read or written.

Use the body parameters on this worksheet as follows:

- **Tag name**: Enter tag names to be updated with file contents or with values that are to be written to a file. If the tag is an array, the first position to be used has to be specified. If the tag is an array or a class (or both), all array positions and class elements are automatically activated by default. To configure a specific array position and/or class element, enter it in the **Tag name** field as usual.

  For example `level[3].member`.
- **Number of Elements** field: Specify how many positions of the array tag are in use.

In this field, tag names can be configured in curly brackets (`{tagname}`) to dynamically change the number of elements configured in the recipe for each array tag at runtime.

To execute a recipe worksheet, the Recipe on page 704 function can be used anywhere an expression is allowed.

10.4 Report worksheet

Report worksheets are used to design reports that are dynamically generated at runtime (using the current values of the included tags) and then either sent to a printer or saved to a file.

To create new report worksheets, open the context menu of the "Tasks" folder and select **Add...** ▸ **Report**

To edit an existing report worksheet, double-click on it in the Project Explorer.

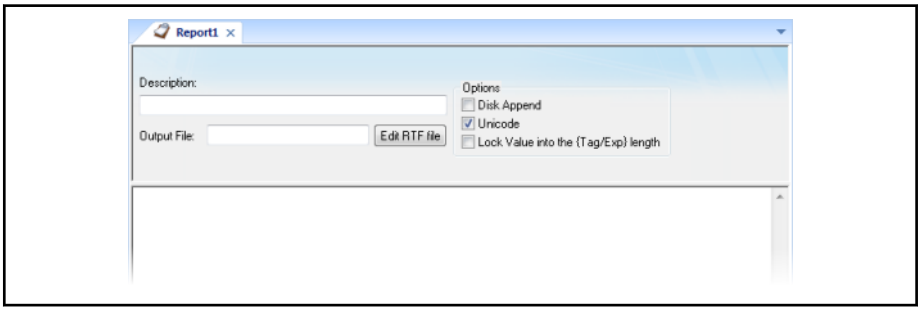


Fig. 10-4: Report worksheet

"Report" worksheets are divided into two areas:

- "Header" area (top section) containing information for the whole group; and
- "Body" area (bottom section) to define each tag in the group.

Use the header parameters on this worksheet as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.

- **Output file**: Enter a tag name for the output file (using the **{tag}** syntax) where data is stored during output for printing to a file. The tag value is part of the file name.

  For example: `report{Day}.out`. The generated file might be `report1.out`, `report2.out`, `report3.out`, etc., according to the tag day value.



Report configuration files use `.RCP` as default extension. The data is saved to the file specified in the **Output file** field.

- **Edit RTF file** button: Click to access the report as an RTF file for editing the layout, etc.
- **Disk append** checkbox: When printing to a file
  - Enable the checkbox to add (change) the new report at the end of an existing file
  - Disable the checkbox to replace the existing report in that file by the new report
- **Unicode** checkbox: Click (enable) to save the report in Unicode format (two bytes per character) or (disable) to save the report in ASCII format (one byte per character).
- Checkbox **Lock value into the** `{Tag/Exp}` **length**: Click (enable) to automatically limit the values of tags/expressions in the report to fit between the curly brackets, as they are positioned in the body of the report (see below). This helps to preserve the layout of the report. If this option is left unchecked, the full values of tags/expressions in the report are displayed.

Use the "body" of this worksheet for report formatting. Reports can be configured using data in the system and indicating where to print the tag values. Each tag name replaces the `{tag_name}` tag name. For Real type tags, use the following syntax: `{tag_name n}`, where **n** is the number of decimal places to be printed.



If the standard report editor (text only: ASCII or Unicode) is used, the number of characters reserved for the tag value corresponds to the number of characters used to type the tag name (including the two "curly" brackets). For example, if `{TagA}` is configured in the report body, six characters are to be reserved for the tag value in the report file. This functionality does not apply to reports in RTF format.

To execute a report worksheet, the Report on page 705 function can be used anywhere an expression is allowed.

10.5 ODBC worksheet

The ODBC interface runs in a network environment and uses the default Windows ODBC configuration. The "ODBC" task can be used for data exchange between "WinStudio" and any database supporting this interface.



In addition to configuring the "ODBC" worksheet, the Windows ODBC default driver has to be configured. "WinStudio" refers to the User DSN (Data Source Name) configured by means of the Control Panel. For further information, refer to the Windows documentation.

Additionally, the ODBC interface is not available for projects running on Windows Embedded target systems.

To create new ODBC worksheets, open the context menu of the "Tasks" folder and select **Add... ▸ ODBC worksheet**

To edit an existing ODBC worksheet, double-click on it in the Project Explorer.

A new worksheet is displayed as follows.

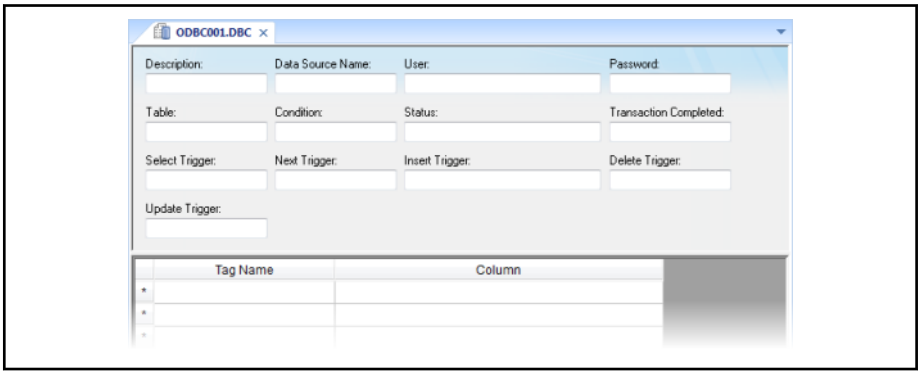


Fig. 10-5: ODBC worksheet

ODBC worksheets are executed under the "ODBC runtime" task. However, creating a new worksheet does not automatically activate the task. The task has to be activated in the task configuration (in the WinStudio context menu under **Tools ▸ Status**) to start at runtime. For more information, see Execution tasks, page 91.

"ODBC" worksheets are divided into two areas:

- "Header" area (top section) containing information for the whole group, defining tags to start read and write events, setting return values, handling database access parameters, etc.; and
- "Body" area (bottom section) to define each tag in the group and relate tags to fields in the current tab from the database table.

Use the header parameters on this worksheet as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.
- **Data source name**: Enter the same data source name (DNS) specified in the Windows Control Panel containing information on specific database access.
- **User**: Enter a user name to access the database.
- **Password**: Enter the user password.
- **Table**: Enter a table name in the database.
- **Condition**: Enter a search condition or filter.
- **Status**: Enter a tag in which the status value is to be written on ODBC access. The tag is to report 0 for success and use another value for an error code.

- **Transaction completed**: Enter a tag that changes its value when the transaction is executed.
- **Select trigger**, **Next trigger**, **Insert trigger**, **Delete trigger** or **Update trigger**: Enter a tag to work as a trigger, where each value change causes WinStudio to execute the respective command. At least one trigger field is required.

Use the body parameters on this worksheet as follows:

- **Tag name**: Enter the names of tags to update with file contents or tags whose values are to be written to a file.
- **Column**: Enter the data location in the file (for example, "R3CH" corresponds to Row 3, Column H of an Excel sheet)

Use the Windows Control Panel to set up the ODBC interface for Excel files. Proceed as follows:

1. Click the **Start** ▸ **Settings** ▸ **Control Panel**.
2. In the "Control Panel" window, double-click on the ODBC icon to open the "ODBC data source administrator" dialog.
3. In the "ODBC data source administrator" dialog, click on Excel files in the “User data sources” list and on the “Configure” button.
4. In the "ODBC Microsoft Excel setup" dialog, enter the Windows configuration name to be used in the DSN field on the "ODBC" worksheet into the “Data source name” field.
5. Click the “Select workbook” button to configure the Excel file to be used.
6. Return to the “ODBC data source administrator” dialog and check whether the user DSN is displayed in the list. Click on “OK” to close the dialog.
7. After configuring the ODBC Windows interface, the "ODBC" worksheets of the project have to be configured.
8. From the "Tasks" tab, insert a new ODBC worksheet.
9. Ensure that the automatic start of the ODBC runtime is specified in the task configuration (menu item **Tools** ▸ **Status** in the WinStudio context menu)

To start this configuration, the project simply needs to be executed. The project handles the **Select**, **Next**, **Insert**, **Delete** and **Update** triggers to enable data exchange in rows in Excel and tags configured in the worksheet.

The meaning of specific error codes can be found in the Windows documentation.

The following is a list of "WinStudio" error codes:

- *Select command*
  - 1 - Error in the ODBCPREPARE function
  - 2 - Error in the ODBCBINDCOL function
  - 3 - Error in the ODBCEXECUTE function
  - 4 - Error in the ODBCSETCH function
- *Next command*
  - 5 - Error in the ODBCSETCH function
- *Insert command*
  - 6 - Error in the ODBCPREPARE function
  - 7 - Error in the ODBCEXECUTE function

  - 8 - Error in the ODBCCOMMITE function
- *Update command*
  - 9 - Error in the ODBCPREPARE function
  - 10 - Error in the ODBCEXECUTE function
  - 11 - Error in the ODBCCOMMITE function
- *Delete command*
  - 12 - Error in the ODBCPREPARE function
  - 13 - Error in the ODBCEXECUTE function
  - 14 - Error in the ODBCCOMMITE function

10.6 Math worksheet

Math worksheets are used to implement program logic (using the internal script language) that is to be continuously executed at runtime, rather than on specific actions like the user pressing a button on a screen.



Math worksheets are functionally similar to script on page 401 worksheets, except that the internal scripting language is used instead of VBScript.

To create new math worksheets, open the context menu of the "Tasks" folder and select **Add...** ▸ **Math worksheet**

To edit an existing math worksheet, double-click on it in the Project Explorer.

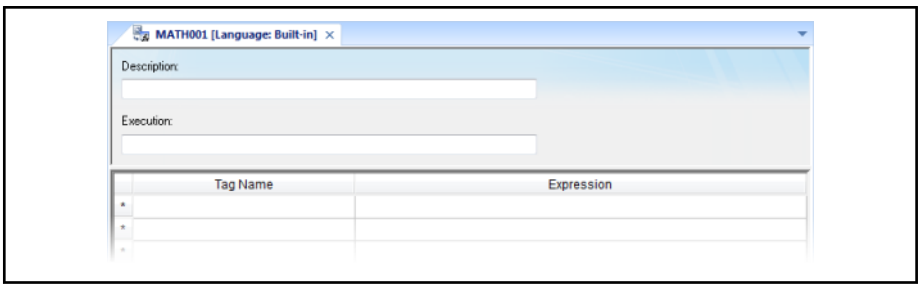


Fig. 10-6: Math worksheet

Math worksheets are divided into two areas:

- "Header" area (top section) containing information for the whole group; and
- "Body" area (bottom section) to define each tag, expression and programming lines (logical routines and mathematical calculations through functions and logical operations) in the group.

Use the header parameters on this worksheet as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.
- **Execution**: Enter an expression, a single tag or a constant value to determine when the worksheet is to be executed.



The project executes the worksheet only if the **Execution** field result is "not zero". If the worksheet is to be executed always, enter 1 (constant value) into the **Execution** field.

Use the body parameters on this worksheet as follows:

- **Tag name** field: Enter a tag to receive a return value from the specified calculation in the **Expression** column.
- **Expression** field: Enter an expression to receive a return value from the specified tag in the **Tag name** column.

To execute a math worksheet at a specific time independently of the configuration in the **Execution** field, use the Math on page 703 function where expressions are allowed.

10.7 Script worksheet

10.7.1 Script worksheet, general information

Script worksheets are used to implement program logic (using VBScript) that is to be continuously executed at runtime, rather than on specific actions like the user pressing a button on a screen.

Script worksheets are functionally similar to math on page 400 worksheets, except that VBScript is used instead of the internal scripting language.

To create new script worksheets, open the context menu of the "Task" folder and select **New script**

To edit an existing script worksheet, double-click on it in the Project Explorer.

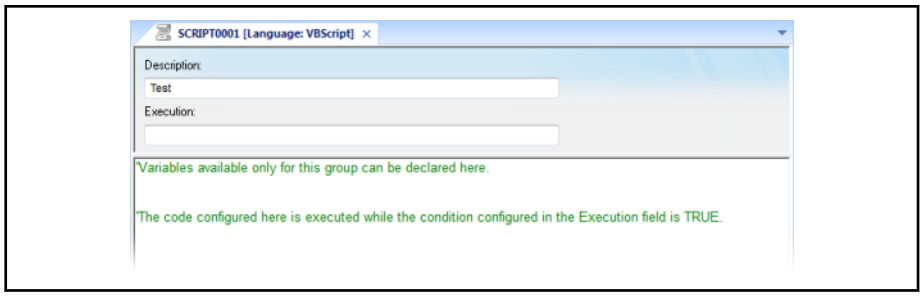


Fig. 10-7: Script worksheet

The code configured in each Script worksheet is executed by the background task. The project scans the worksheets sequentially (based on the worksheet number) and executes only the groups in which the condition configured in the **Execution** field of the worksheet is TRUE (i.e., non-zero).



The syntax supported by the internal scripting language in the **Execution** field has to be used. VBScript is only supported by the body of the worksheet.

Variables declared in the worksheet have local scope for that specific group only. They are not available for any other VBScript interface.

Procedures (i.e., functions and subroutines) cannot be defined in the script worksheet. However, procedures defined in the global procedures, on page 927 or in the startup script on page 402 can be called.

Example:

```
'Variables available only for this group can be declared here
Dim myVar, myTest
myTest = 1

'The code configured here is executed while the condition configured in the Execution field is TRUE
myVar = $FindFile("c:\*.txt")
If MyVar > 0 Then
```

```
  $TagNumOfFiles = myVar
End If
```

NOTICE

If a script worksheet is saved at runtime (online configuration) the startup script is executed again.

The current value of the local variables of all script worksheets are reset.

10.7.2 Script Worksheet Startup

The startup script worksheet is a VBScript interface that is automatically executed when the project is run.

To edit the startup script worksheet, double-click on it in the Project Explorer. (It can be found in the folder "WinStudio\Tasks\Scripts")

The worksheet is displayed:

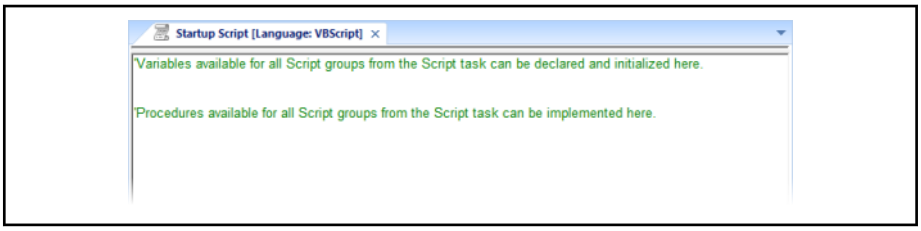


Fig. 10-8: Startup script worksheet

The code configured in this worksheet is executed just once if the Background Task module (BG_Task) is started. This interface is useful for initializing variables or executing logics that have to be implemented when running the project.

Variables can be declared and initialized and procedures defined. However, variables or procedures declared in this interface are available ONLY to the script worksheets executed by the Background Task module. They are not available to any VBScript interface from the Graphic module.

Example:

```
'Variables available for all Script groups from the Script task can be declared and initialized here
Dim MyVar, Counter
MyVar = 100

'Procedures available for all Script groups from the Script task can be implemented here

Function AreaEquTriangle(base, high)
  AreaEquTriangle = (base * high) / 2
End Function

Sub CheckLimits(myValue, myHiLimit, myLoLimit)
  If (myValue > myHiLimit Or myValue < myLoLimit) Then
    MsgBox("Value out of range")
  End If
End Sub

'The code configured here is executed just once when the Background task is started
If $GetOS() = 3 Then
  MsgBox ("Welcome! This project is running under Microsoft Windows Embedded operating system.")
Else
  MsgBox("Welcome! This project Is running under Microsoft Windows desktop operating system.")
End If
```

10.8 Scheduler Worksheet

Scheduler worksheets are used to execute program logic (using the internal scripting language) at a specific date/time, on a regular time interval or upon triggering an event.

To create new scheduler worksheets, open the context menu of the "Tasks" folder and select **Add...** ▸ **Scheduler**

To edit an existing scheduler worksheet, double-click on it in the Project Explorer.

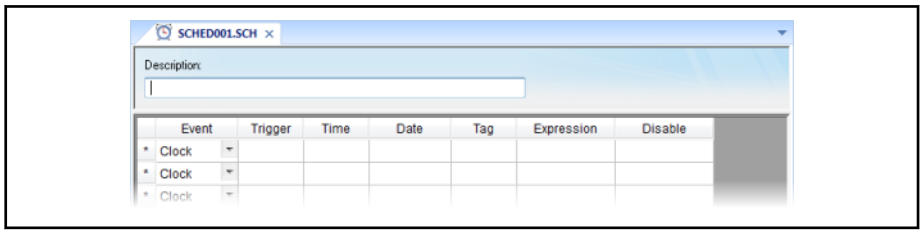


Fig. 10-9: Scheduler worksheet

"Scheduler" worksheets are divided into two areas:

- **Header** area (top section) containing information for the whole group
- **Body** area (bottom section) to define each tag, expression and condition for the group.

Use the parameters on this worksheet as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.
- **Event**

  Selection list: Click to select an event type from the following options:

  – **Calendar**: Generates time periods exceeding 24 hours. For example, it is possible to define events that print a report every Friday at a specific time.

☞ Be sure to complete the **Date** field to set a specific date for event execution.

  – **Clock**: Generates time periods smaller than 24 hours (intervals in minutes or seconds). This function is frequently used with trend graphics. For example, it is possible to define a tag that is incremented each hour.

  – **Change**: Event related to the change of a tag in the **Trigger** field.

☞ This only works for tag changes on the project server, regardless of the scope defined for the tag.

- **Trigger**: This field is used only with the **Change** Event type. Enter the name of a project tag into this field. The **Expression** is evaluated, if the value of the tag is changed.
- **Time**: This field is used with the **Calendar** and **Clock** Event types.

  If the Event type is **Calendar**, **Time** is a specific time of the day on **Date**. If **Date** and **Time** are reached, **Expression** is evaluated. If **Clock** is selected as Event type, **Time** represents a time interval from project execution. Every time the interval occurs, **Expression** is evaluated.

Either way, enter a time using the `HH:MM:SS.ms` format. Valid values are `00` to `23` for hours, `00` to `59` for minutes, `00` to `59` for seconds, and `1` to `9` for milliseconds. (Milliseconds are optional.) Examples: `03:00:00` is every three hours, `00:00:00.1` is every 100 milliseconds.

- **Date**: This field is used only with the **Calendar** Event type. Enter a specific date in `MM/TT/YYYY` format. If **Date** and **Time** are reached, **Expression** is evaluated.

  Valid values are `01` to `12` for months, `01` to `31` for days, and `1900` to `2099` for years. If the field is left blank, the event occurs daily at the specified **Time**.
- **Tag**: Enter a tag that is to receive the value returned by **Expression** (if any).
- **Expression**: Enter an expression to be evaluated. This field is used by all events.
- **Disable**: Contains a disable condition for the specified function. Leave this field blank or use an expression value equal to zero to execute the function. If an expression value equal to one is used, the function is not executed (Disable = 1).

10.9 Database/ERP worksheet

In addition to ODBC, "WinStudio" also supports Microsoft .NET ActiveX data objects (ADO.NET) for interfacing between the project tags database and other external databases. A database/ERP worksheet is used to associate project tags with external database fields.



For further information on ADO.NET support in "WinStudio" - including how to communicate with remote databases using the "WinStudio" Database Gateway software - please refer to Database interface on page 553.

To communicate with an external database, a connection on page 404 to the database has to be configured first and then, a worksheet on page 406 has to be created that associates project tags with the database fields.

Database connections

To create a new connection to a target database, proceed as follows:

1. In the Project Explorer on page 45, open the context menu of the **External databases** folder and select the entry **Database connection configuration**.
2. The "Database connection" configuration dialog is displayed listing all already configured database connections.

   Press **Add** to create a new database connection. The "Database connection" dialog is displayed.

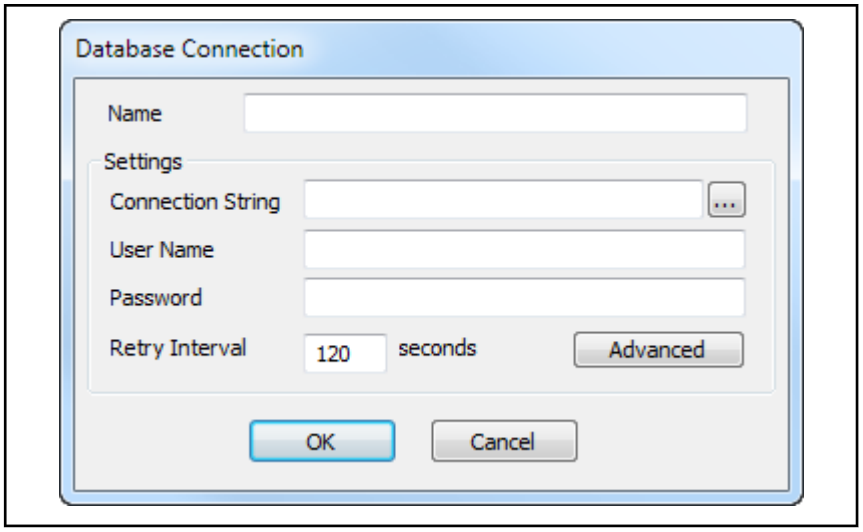


Fig. 10-10: "Database connection" dialog

3. In the **Name** field, enter the name to be used to reference the target database. Multiple database connections can be created, however, each connection requires a unique name.
4. In the **Connection string** field, click the browse button **...** to open a standard "Data link properties" dialog. Use the dialog to configure a connection string for the target database.

☞ The list of database providers displayed in the "Data link properties" dialog depends on the providers actually installed and available on the station where the development application is running. For further information on using the "Data link properties" dialog, please refer to the Windows help.

---

5. In the **User name** and **Password** fields, enter an appropriate login for the target database. The login has to have already been created on the database server and requires privileges to read from and write to the database tables.
6. If the connection is established to a remote database via the Studio Database Gateway on page 556, click the **Advanced** button to open the advanced settings dialog.

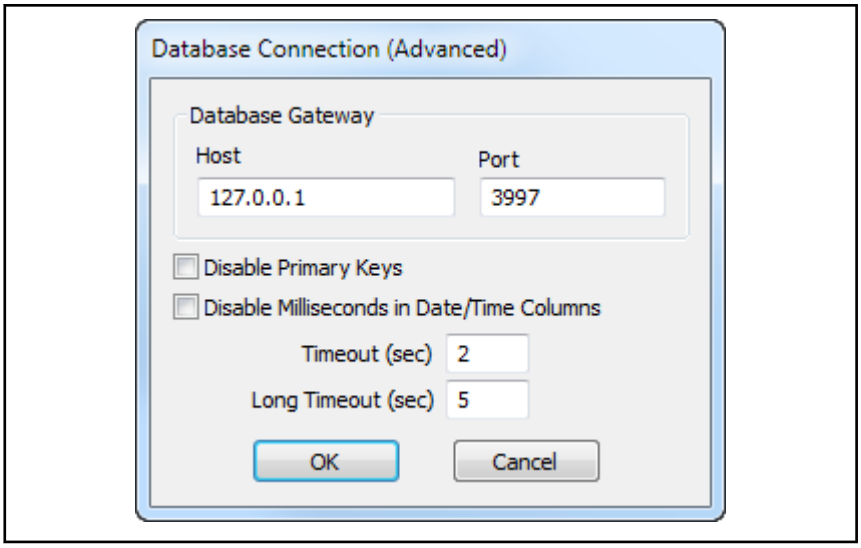


Fig. 10-11: "Database connection" dialog (Advanced) dialog

7. In the **Host** field, enter the IP address of the station that is running the "WinStudio" database Gateway software (`STADOSvr.exe`). In the **Port** field, enter the port number on which the software has been configured to run.

   Additional settings to be configured, if necessary:

   - **Disable primary keys** checkbox: "WinStudio" tries to define a primary key to the table in order to speed up the queries. If a database is used that does not support primary keys (e.g., Microsoft Excel), this checkbox is to be enabled.
   - **Disable milliseconds in Date/time columns** checkbox: "WinStudio" tries to include milliseconds when saving a date/time in the database. If a database is used that does not support milliseconds, this checkbox is to be enabled.

8. Click **OK** to save the connection configuration and close the dialog.

Database connections are saved as XML files in the `\project_name\Config` sub-folder. Each file is given the same name as the name of the connection (as entered in the **Name** field of the "Database connection" dialog) with the `.XDC` file extension. For example, the connection configuration "DB1" is saved in the file…

`\project_name\Config\DB1.XDC`

Database worksheet



This function emulates SQL (Structured Query Language) database operations. Before using this function, users have to familiarize themselves with the structure and execution of SQL commands.

Database worksheets allow asynchronous execution of database operations and offer a user-friendly interface for building SQL commands. To create new database worksheets select the entry **New database worksheet**in the "Tasks" or "External databases" context menu.

A new worksheet is displayed, as shown in the following figure:

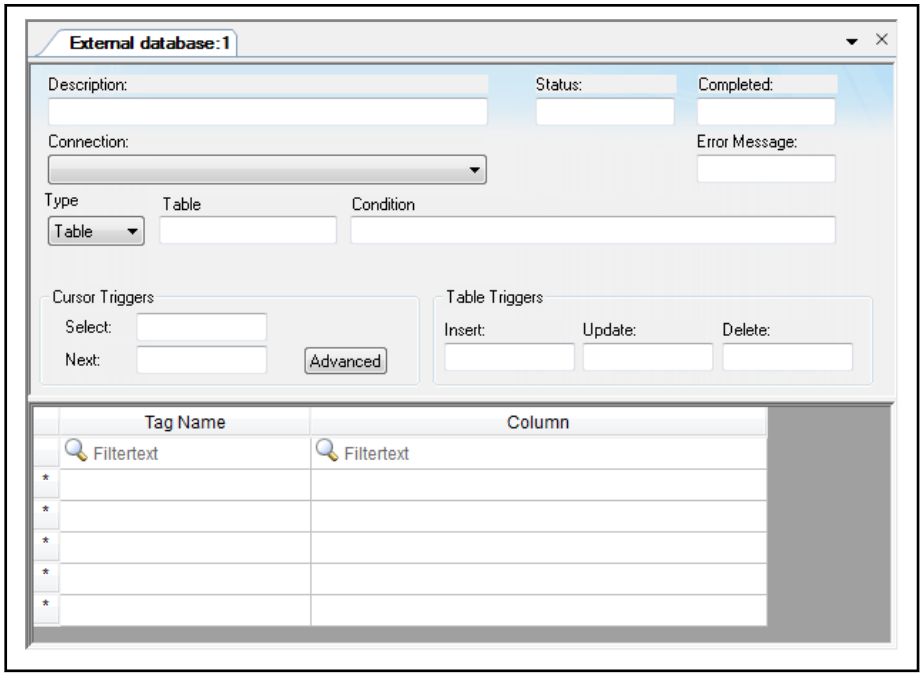


Fig. 10-12: Database worksheet

Database worksheets are saved in the `\project_name\Config` directory, with the `.XDB` file extension. Each new worksheet is automatically numbered in the order of its creation. For example, the first worksheet created is saved in the file…

`\project_name\Config\DB001.XDB`

Database worksheets are executed under the `Database/ERP Runtime` task. However, creating a new worksheet does not automatically activate the task. The task has to be configured in the task configuration (in the WinStudio context menu under **Tools** ▸ **Status**) to start at runtime. For further information, refer to execution task, page 91.

Database worksheets run only on the server and all triggers have to be configured with server tags.

Worksheet header

The header of the database worksheet is configured as follows:

- **Description**: Enter a description of the worksheet for documentation purposes.
- **Status**: Enter the name of a numeric tag that is to receive status codes for database operations at runtime:

Status codes for external database operations

Status code	Description
4	Result set is empty
3	Cursor released and query successfully closed
2	Beginning of result set reached, usually while moving cursor to the previous row
1	End of result set reached, usually while moving cursor to the next row
0	No errors; status normal
-1	Error while connecting to the specified database (see **Connection** below)

Status code	Description
-2	Error while selecting the result set
-3	Error while moving the cursor to the next row (see **Next** trigger below)
-4	Error while moving the cursor to the previous row (see **Previous** trigger below)
-5	Error while closing the query (see **Close query** trigger below)
-6	Error while inserting rows in the result set (see **Insert** trigger below)
-7	Error while updating the result set (see **Update** trigger below)
-8	Error while deleting the result set (see **Delete** trigger below)

- **Completed**: Enter the name of a numeric tag that is to be toggled when database commands are successfully executed.
- **Error message**: Enter the name of a String tag that is to receive detailed error messages if errors occur at runtime.
- **Connection**: Click to select a connection to the target database. All available connections are listed, as configured in the "Database connection" dialog described above on page 404.
- **Type**: Click to specify how the result set is to be selected for the worksheet:
  - **Table**: Enter a table name and an optional filter condition. (The filter condition is equivalent to the SQL "Where" clause.) All rows of the table that match the filter conditions are selected.
  - **SQL**: Enter a custom SQL "Select" statement.

☞ For **Table**, **Condition** and **SQL statement**, the names of project tags that contain the desired information can be entered. This enables programmatically changing the selection at runtime. However, tag names have to be enclosed in curly brackets ({ }) to distinguish them from actual Strings. Additionally, existing selections have to be released before opening new ones; see **Close query** below.

- *Cursor triggers pane…*
  - **Selection**: Enter any tag; if the value of the tag changes, a new cursor opens the first row of the result set and copies the values to the tags configured in the worksheet body.
  - **Next**: Enter any tag; if the value of the tag changes, the cursor moves to the next row of the result set and copies the values to the tags configured in the worksheet body.
  - **Advanced**: Click to open the "Advanced cursor options" dialog:

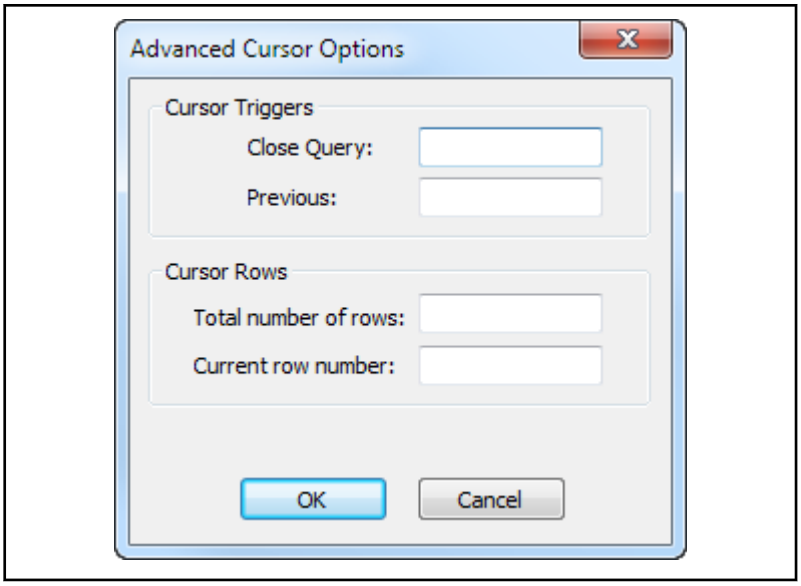


Fig. 10-13: "Advanced cursor options" dialog

- **Close query**: Enter any tag; if the value of the tag changes, the cursor releases the result set.
- **Previous**: Enter any tag; if the value of the tag changes, the cursor moves to the previous row of the result set and copies the values to the tag configured in the worksheet body.
- **Total number of rows**: Enter a numeric tag that is to receive the total number of rows in the result set.
- **Current row number**: Enter a numeric tag that is to receive the number of the current row (i.e., the position of the cursor). When a result set is first opened using the **Select** trigger, this number is 1. Each **Next** trigger increases this number and each **Previous** trigger decreases it.

- *"Table triggers" pane*
  - **Insert** field: Enter any tag; if the value of the tag changes, a new row is inserted with the current values of the tags configured in the worksheet body.
  - **Update** field: Enter any tag; if the value of the tag changes, all rows of the result set are overwritten with the current values of the tags configured in the worksheet body.
  - **Delete** field: Enter any tag; if the value of the tag changes, all rows of the result set are deleted.

☞ Table triggers are available only if **Type** is set to **Table**, as these operations work on the entire table row.

Worksheet body

In the body of the worksheet, you can map project tags on page 104 to the columns (fields) of the result set. For each row of the body, enter a **tag name** and its corresponding **Column**. The available columns depend on how the result set is selected.

This selection can change at runtime. Thus, all required columns have to be mapped.

There can be up to 2048 rows per worksheet. If more are required, try to create additional worksheets and to adjust the result for each worksheet.

10.10 Sorting or filtering the lines of a worksheet

Sort or filter the lines of a worksheet to facilitate searching the lines or finding a particular element.

Before starting this task, insert a worksheet and open it for editing. In addition it should be clear how sorting and filtering works in general spreadsheet applications.

Please note that the lines can only by sorted and filtered in the following types of worksheets:

In Project tags, Shared tags and System tags data sheets;

In the Translation table worksheet;

In all Task worksheets except for report and script which do not have lines;

In all Communication worksheets.

The other worksheets do not have lines to sort or filter.

The worksheets are sorted alphanumerically by selected column either in ascending ((0-9, A-Z) or descending order (Z-A, 9-0).

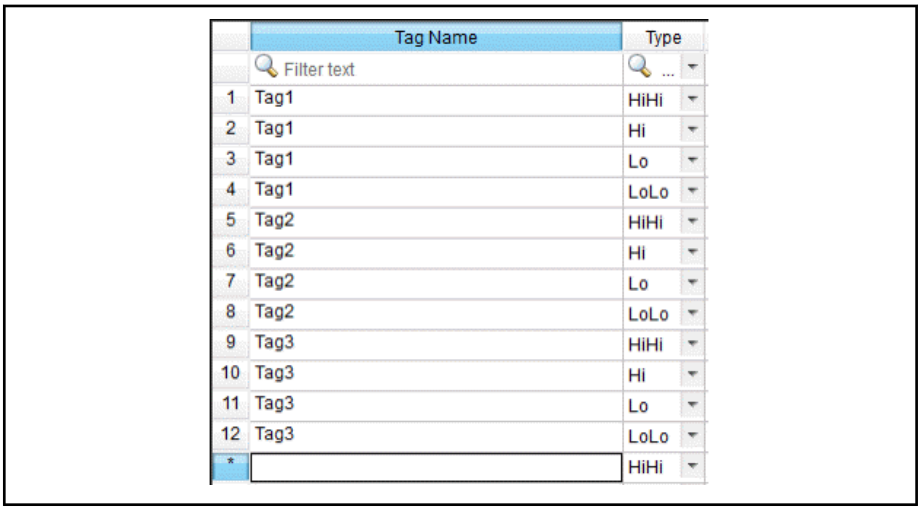
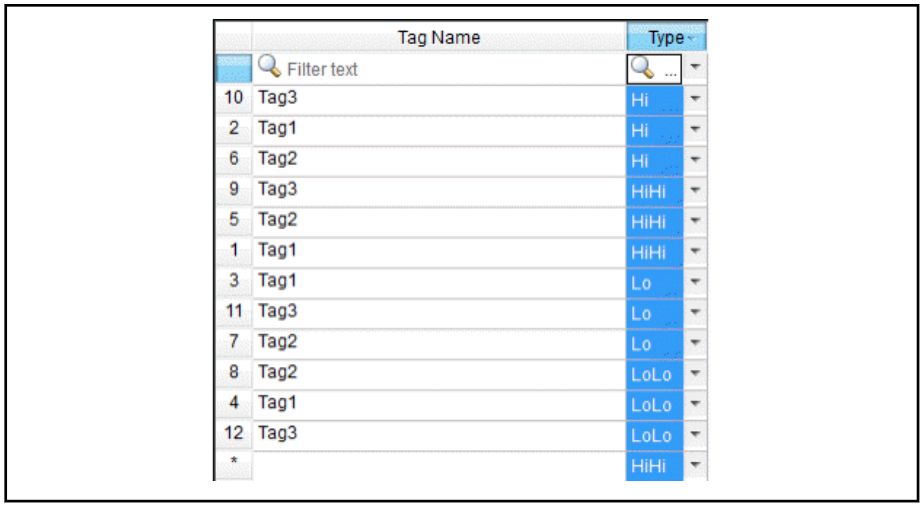


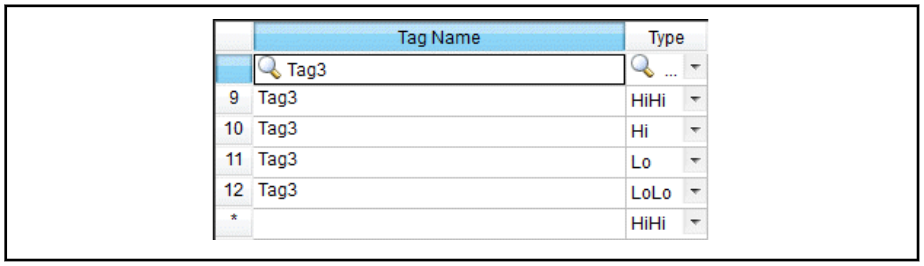
Fig. 10-14: Alarm worksheet lines in their original order



	Tag Name	Type
	Filter text	...
10	Tag3	Hi
2	Tag1	Hi
6	Tag2	Hi
9	Tag3	HiHi
5	Tag2	HiHi
1	Tag1	HiHi
3	Tag1	Lo
11	Tag3	Lo
7	Tag2	Lo
8	Tag2	LoLo
4	Tag1	LoLo
12	Tag3	LoLo
*		HiHi

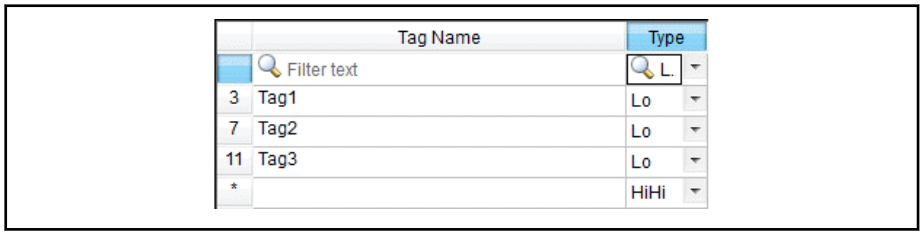
Fig. 10-15: Alarm worksheet lines sorted by type

The worksheets are filtered according to the string entered in the selected column. Only lines that match the string will be displayed.



	Tag Name	Type
	Tag3	...
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

Fig. 10-16: Alarm worksheet lines filtered by tag name is "Tag3"



	Tag Name	Type
	Filter text	L.
3	Tag1	Lo
7	Tag2	Lo
11	Tag3	Lo
*		HiHi

Fig. 10-17: Alarm worksheet lines filtered by type is "Lo"



It is still possible to delete lines while they are sorted or filtered.

Sorting or filtering lines:

1. To sort the lines, click on the column header by which you would like to sort. Click once to sort in ascending order and again to sort in descending order. The current order (i.e. the sorting direction) is indicated by the arrow to the right of the column name.



It is not possible to sort by several columns.

2. To cancel the sorting and restore the original order of the lines, click on the header of the first (numbered) column.
3. To filter the lines, enter the sting by which you would like to filter in the top line (zero) of the worksheet and press either **Tab** or **Enter**.

*It is possible to use * and ? as wildcard characters in your string:*

- * stands for any number of characters or no character. Tag* for example corresponds to `Tag`, `Tag3`, `Tag34567`, `TagA` and `Tag_TEMP`.
- ? stands for exactly one character. `Tag?`, for example, corresponds to `Tag3` and `TagA`, while `Tag?????` corresponds to `Tag34567` and `Tag_TEMP`.

It is possible to filter several columns. Only lines that match the filter string in all columns will be displayed.

4. To cancel filtering and restore the original order of the lines, delete the string you entered and either press **Tab** or **Enter**.

Please note that sorting or filtering the lines of a worksheet only facilitates the editing of this worksheet. How the worksheet is executed during runtime does not change. The lines are executed in their original order according to the numbering (column on the left), unless a line is moved or deleted.

11 Communication with other devices

Communication tasks/worksheets are used to exchange tag values with remote devices (e.g., PLC controls), other "WinStudio" projects and all systems implementing supported protocols like OPC and DDE.

The communication between tags and a data source is configured using communication drivers and the respective worksheets in the folder **WinStudio ▸ Variables ▸ Communication ▸ <DriverName>**.

11.1 Configuring direct communication with a remote device

11.1.1 Configuring direct communication with a remote device, general information

Communication drivers are DLLs implementing the communication protocol of a specific remote device to read and write its data. WinStudio provides various standard communication drivers. For communication with VPx and VEx devices, the BR_WS driver is additionally available. For numerous devices, additional drivers are available that can be added to systems to exchange data with these devices.

In the properties dialog of drivers, the communication interface between the project and remote devices can be defined.

☞ Consult the **Help** menu for a description of the default functions and characteristics for all drivers. If a project is to be developed, also refer to the specific documentation provided with each communication driver. This documentation can usually be found in the **DRV** directory.

For communication driver configuration, the interface parameters (for example, the station address and the baud rate) as well as device addresses are to be specified and then linked to project tags.

To add or remove drivers, proceed as follows:

- Open the context menu of the **WinStudio ▸ Variables ▸ Communication** folder and select the entry "Add/Remove driver"

The following dialog with a list of available drivers is displayed.

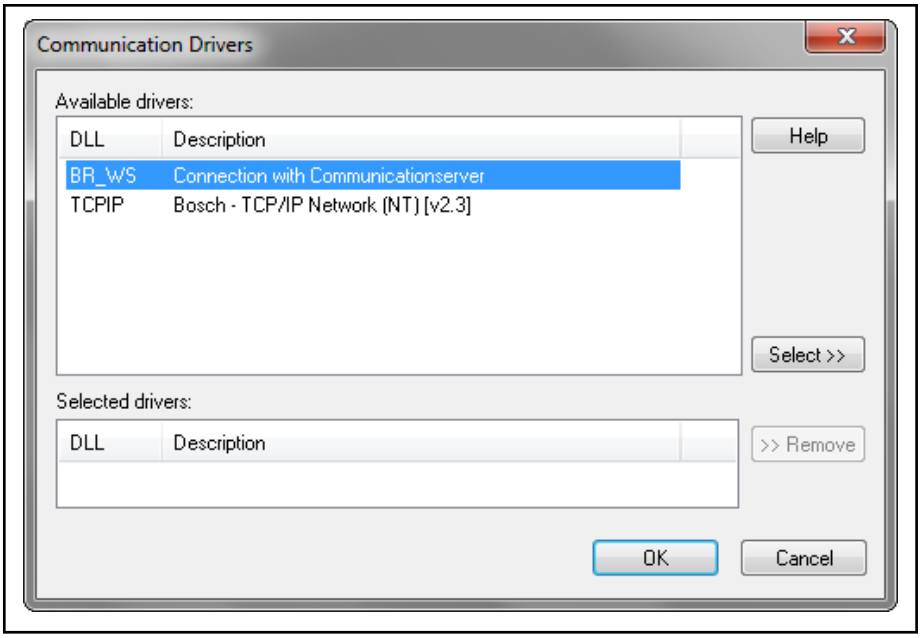


Fig. 11-1: Communication Drivers dialog

The properties in this dialog have the following function:

- **Available drivers**: Lists all available drivers and a brief description of each
- **Help**: With the **Help** button, a detailed description for driver configuration can be opened for the drivers selected under "Available drivers" or "Selected drivers". If additional drivers are installed, no help may be available for these. In this case, the "Help" button is deactivated.
- **Selection**: Click on this button to use the driver selected in the **Available drivers** field in the project.
- **Selected drivers**: Lists all selected drivers and their descriptions (if available).
- **Remove**: Click on this button to stop using the driver selected in the **Available drivers** field in the project and to remove it from the project.

Changed settings can be applied by pressing **OK**. In the "Communication" folder, a respective subfolder for added drivers is created or removed.

The context menu of a driver subfolder contains the entry "Properties". This entry opens the configuration dialog for the respective driver to configure the following settings:

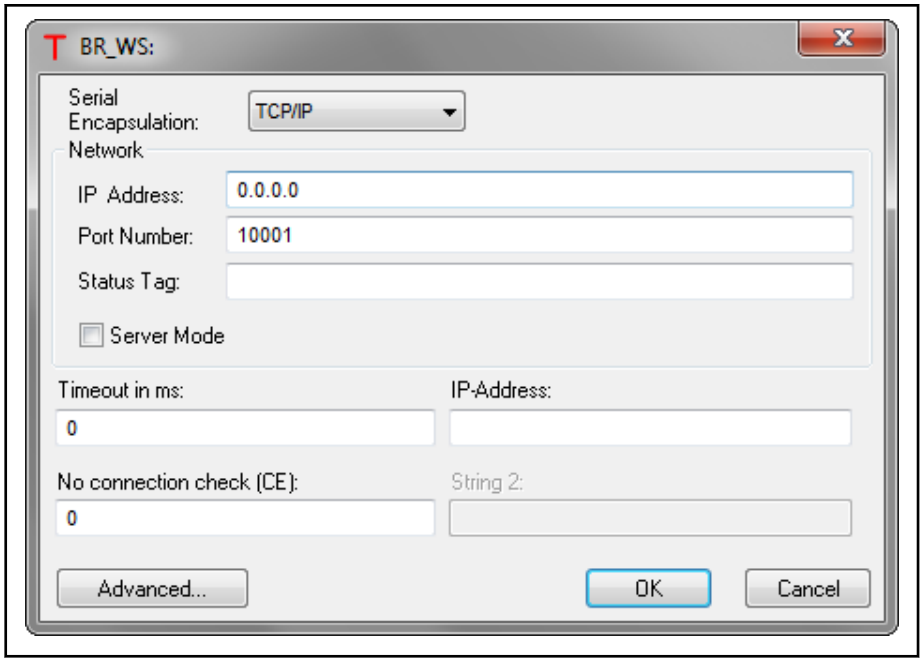


Fig. 11-2: Example of Communication Parameters dialog

The individual driver settings have the following meaning:

- **Serial encapsulation**: Enables serial drivers to communicate with modem, TCP/IP or UDP connections. This setting is only supported for serial drivers developed with the UNICOMM library. This applies to most serial drivers available in this product.

☞ This section covers only the **"None"** option, which enables the driver to connect via a normal serial channel. Please refer to "Using TCP/IP and UDP encapsulation" and "Using modem connections" below for more information on other encapsulation modes. "Serial encapsulation tests" below lists drivers that have been tested with modem, TCP/IP and UDP modes.

- **COM**: Select a serial communication port.
- **Baud rate**, **Data bits**, **Stop bits** and **Parity**: These parameters are required for the configuration of a serial port.
- **Long1**, **Long2**, **String1**, and **String2**: These settings are driver-specific. In the example above, the driver uses **Long1** to set up the error detection method and **String1** to define the PLC family type.
- **Advanced**: Click on this button to open a dialog to configure advanced settings: Use this dialog to change the default driver parameters.

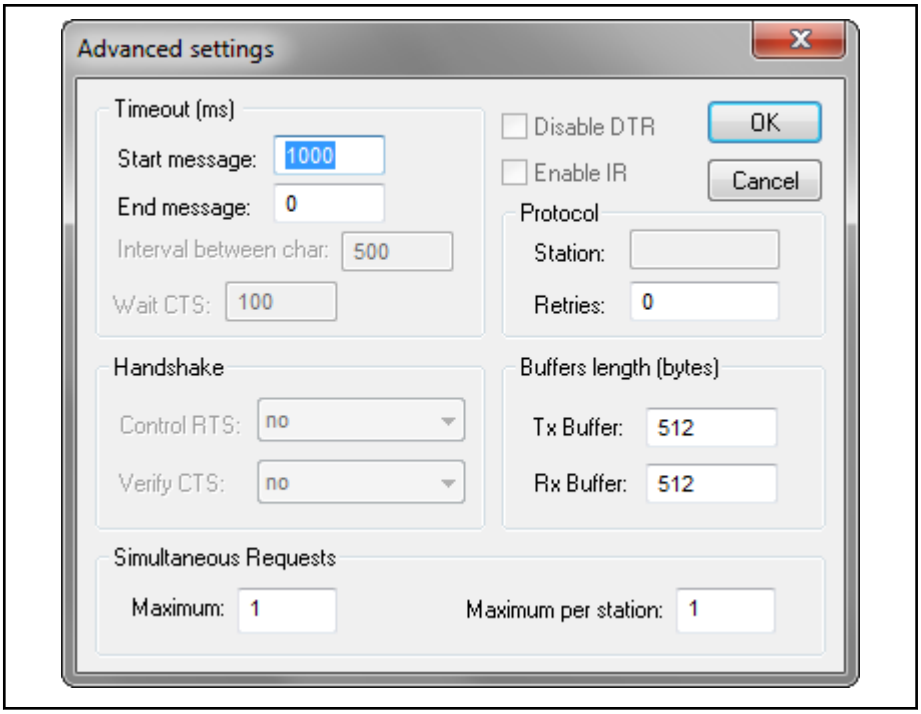


Fig. 11-3: Advanced Settings dialog

To specify or change default driver parameters, proceed as follows:

- *Timeout (ms) pane:*
  - **Start message**: Specify the timeout for the message start.
  - **End message**: Specify the timeout for the message end.
  - **Interval between char**: Specify the timeout between each character.
  - **Wait CTS**: Specify the timeout for the "Clear to Send" (CTS) waiting time.
- *Handshake pane:*
  - **Control RTS**: Specify whether to use the "Request to send" (RTS) control.
  - **Verify CTS**: Specify whether to use the "Clear to send" type of verification.
  - **Disable DTR**: Click (enable) this checkbox to disable the DTR (data terminal ready) function (the DTR signal is not set by the driver before starting the communication).
  - **Enable IR** ("only available on Windows Embedded target systems"): Click (enable) this checkbox to enable the serial driver to use an infrared interface (COM2 port) instead of a standard serial port to communicate with the device (such as the PLC, I/O, handheld computers, etc.).
- *Protocol pane:*
  - **Station**: Some slave drivers, such as the Modbus Slave (MODSL), require a slave network address. Use this field to specify the slave address.
  - **Retries**: Enter a numeric value to specify how many times the driver attempts to execute the same communication command before considering a communication error for this command.

- *Buffers length (bytes) pane:*
  - **Tx Buffer**: Specify the transmission buffer length (in bytes):
  - **Rx Buffer**: Specify the reception buffer length (in bytes).
- *Simultaneous requests pane (available only for selected drivers):*
  - **Maximum**: Specify the maximum number of requests that may be sent simultaneously to all connected devices.
  - **Maximum per station**: Specify the maximum number of requests that may be sent simultaneously to a single device.



The maximum number of simultaneous requests depends on the device and protocol specifications. Please consult the device manufacturer's documentation.

The development application provides two interfaces to configure the driver (associating project tags to device addresses):

- <MAIN DRIVER SHEET> Provides the easiest method for configuring communication between project tags and device addresses. With this interface, tags can be automatically grouped to provide the best performance during runtime. This interface cannot be used to control the time needed to scan a group of tags individually.
- STANDARD DRIVER SHEETS on page 421 Enables controlling the time needed to scan a group of tags individually.

Both sheets can be used at the same time.

Using TCP/IP and UDP encapsulation

Most of the serial drivers allow the use of TCP/IP or UDP/IP encapsulation. The encapsulation mode has been designed to provide communication with serial devices connected to terminal servers on Ethernet or wireless networks. A terminal server can be seen as a virtual serial port. It converts TCP/IP or UDP/IP messages on Ethernet or wireless networks to serial data. Once the message has been converted to a serial format, standard devices that support serial communication to the terminal server can be connected.

The following figure illustrates one example of applying this solution:

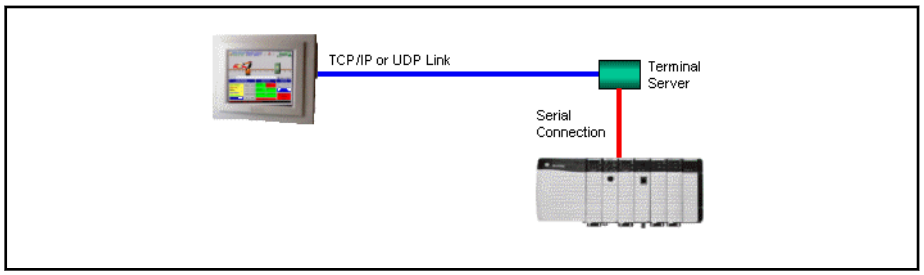


Fig. 11-4: TCP/IP encapsulation

To enable encapsulation, proceed as follows:

1. Right-click on the “Driver” folder and select **Settings** from the context menu. By this means, the communication parameters are accessed.
2. In the **Serial encapsulation** field, select **TCP/IP** or **UDP/IP**:

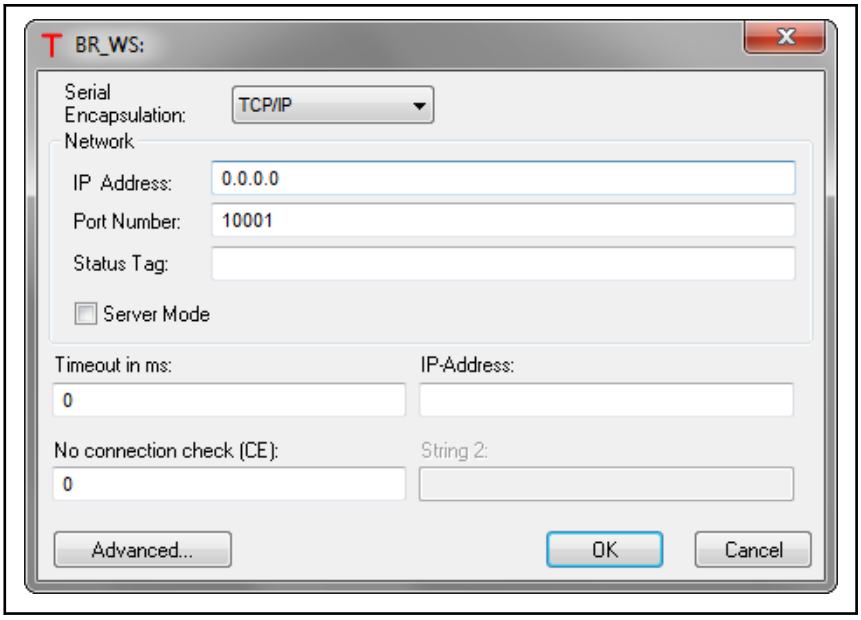


Fig. 11-5: Dialog: Encapsulation

The following fields are available:

- **IP address**: Specify the IP address for the terminal server. In this field, tags can be entered between curly brackets.
- **Port number**: Enter the TCP/IP or UDP/IP port number.
- **Status tag**: This field is available only when using TCP/IP. The tag on this field receives the value 1 when the TCP/IP connection is established; otherwise, it receives 0.
- **Server mode**: TCP/IP encapsulation enables the Server mode, making the remote client responsible for establishing the connection to enable the communication.

Using modem connections

Most of the serial drivers enable the use of modem connections. Modem connections have been designed to enable communication with remote serial devices connected via phone lines.

The following figure illustrates one example of applying this solution:

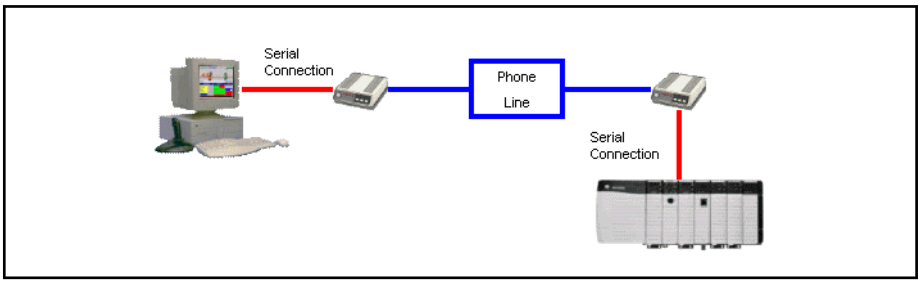


Fig. 11-6: Modem connection

To enable the modem connection, proceed as follows:

1. Right-click on the “Driver” folder and select **Settings** from the context menu. By this means, the communication parameters are accessed.
2. In the **Serial encapsulation** menu, select **Modem**:

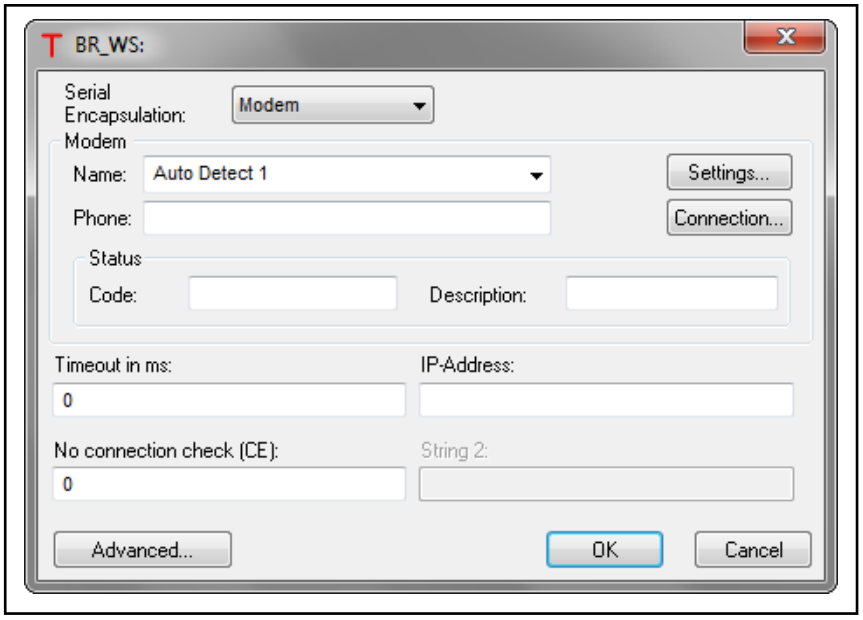


Fig. 11-7: Dialog: Serial encapsulation

NOTICE **The Modem option is not supported for Pocket PC v3.00 or older.**

The following settings are available:

- **Name**: Select the modem that the driver is to use to establish the connection.

  If the modem name is unknown, use the “Auto detect” option:

  – **Auto Detect 1** uses the first modem available
  – **Auto Detect 2** uses the second modem available
  – **Auto Detect 3** uses the third modem available
  – **Auto Detect 4** uses the fourth modem available

- **Phone**: Enter a phone number for the driver to connect to the remote device. In this field, tags can be entered in curly brackets to change the pone number during runtime.

- **Settings**: Click on this button to configure the modem settings. The window displayed after clicking on this button depends on the operating system and the modem type.

  NOTICE **The settings configured by clicking on this button are not saved with the project.**

  The information is saved on the operating system registry and is valid only on the computer that is interacted with. If the project is installed on a different computer, these settings have to be reconfigured.

- **Connection**: Click to open the "Connection control"

  The default connection settings are sufficient for most projects. However, the connection can be fully controlled and incoming calls can be enabled by clicking on this button.

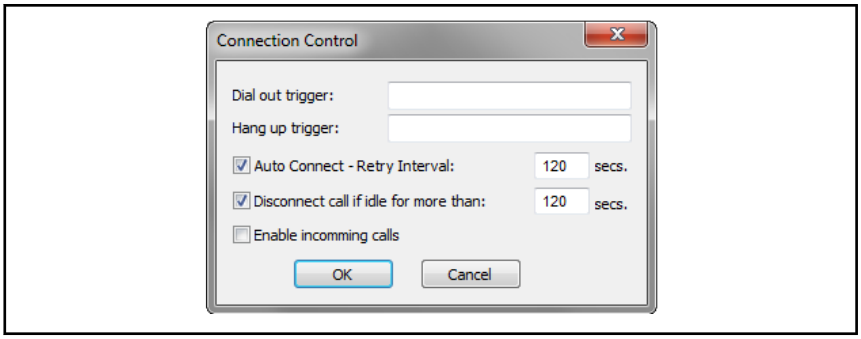


Fig. 11-8: "Connection control" dialog

- **Dial out trigger**: If the value of the tag configured in this field changes, the driver tries to connect to the remote device. If the connection has already been established, the command is ignored. This field can be ignored if "Auto connect" is used.
- **Hang up trigger**: If the value of the tag configured in this field changes, the driver disconnects from the remote device. If the device is disconnected, the command is ignored. This field can be ignored if "Disconnect call if idle for more than" is enabled.
- **Auto connect**: If this option is enabled, the driver tries to connect to the remote device before sending any information. If the connection fails, the next attempt is made after the retry interval has expired.
- **Disconnect call if idle for more than**: If this option is enabled, the driver automatically disconnects from the remote device if no communication takes place after the specified time.
- **Enable incoming calls**: Enable this option to enable the driver to receive calls from the remote device. Hang up trigger can be used to drop calls once they have been established. Please note that drivers can use both incoming and outgoing calls.

• *Status pane*

- **Code**: Enter a tag to receive the status value of the connection.

  The status values have the following function:

  - 0 = Disconnected
  - 1 = Connected
  - 2 = Dialing
  - 3 = Dropping
  - 4 = Closing line
- **Description**: Enter a tag that is to receive a complete description of the current status. The description is associated with the **Code** field; however, it provides additional information on the current status.

Serial encapsulation tests

Most serial drivers work with every serial encapsulation mode. However, most of the drivers had been developed before the encapsulation modes were created. The following table lists the drivers tested with specific encapsulation modes. If a driver is to be used that is not listed and if it is not sure if it works, please contact the distributor.

Driver	Modem	TCP/IP	UDP/IP
MODSL	X	X	X
ABKE	X	X	X
MODBU	X		X
OMETH	X		

X = Item has been tested

11.1.2 Driver sheets

Multiple driver sheets can be created for each driver to enter tags and to link them to an address on the remote device. Additionally, each driver sheet offers various configuration options to control execution during runtime.

To create a new driver sheet, open the context menu of the “Driver” folder and select "New driver sheet". Each driver sheet has an unique number that can be changed at any time by means of the "Rename" context function. To open driver sheets, double-click on the new element in the Driver folder.

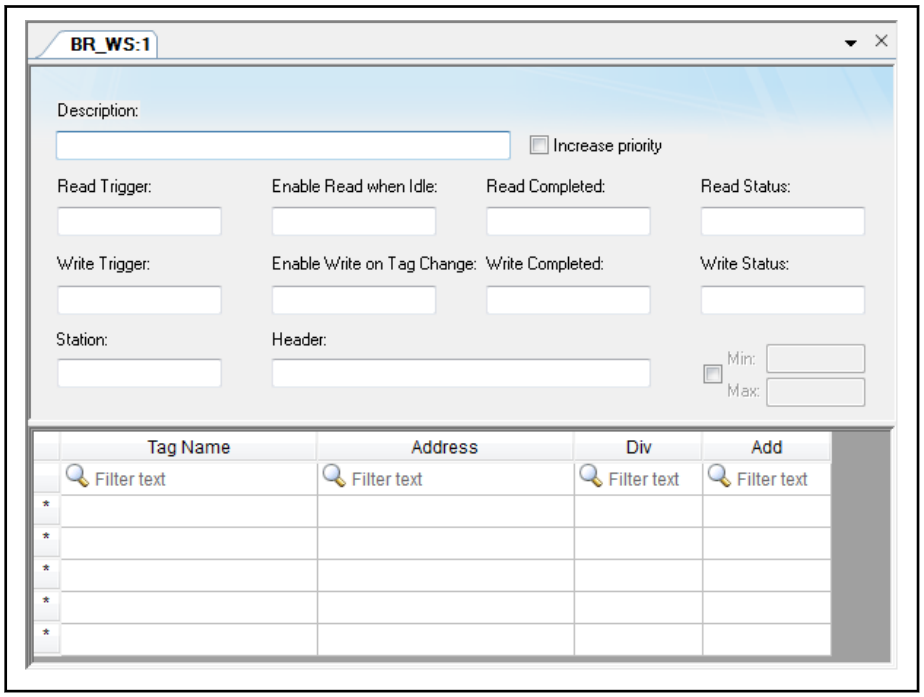


Fig. 11-9: Driver sheet editor

The driver sheet editor is divided into two sections:

- The "header" area (top section) contains parameters that apply to the communication of all tags configured in the body area of the respective worksheet.
- The "body" area (bottom section) links tags available in the project with device addresses.

To use the "header" parameters, proceed as follows:

- **Description**: Enter a description for documentation purposes.
- **Increase priority**: Enable this option if this driver sheet is to be processed first in every execution cycle.

NOTICE **If the Increase priority option is enabled, special attention has to be given to this worksheet.**

If the worksheet keeps triggering communication commands, the project may never be able to execute the other driver sheets.

- **Read trigger**: Enter a tag if a read command for all tags is to be executed in the body area for every value change of the tag.
- **Enable read when idle**: Enter a tag or constant. Use a tag value (or constant) greater than zero to enable reading from the equipment.

NOTICE **If a constant (not zero) is used, check whether the project requires continuous reading.**

Continuous reading may affect execution of the project.

- **Read completed**: Enter a tag to monitor execution of the read command. The communication driver changes the tag after each "read" command.
- **Read status**: Enter a tag that is to be updated with the status on page 977 of the last "read" command by the communication driver.
- **Write trigger**: Enter a tag if a write command for all tags is to be executed in the body area for every value change of the tag.
- **Enable write ontag change**: Enter a tag value or constant (not zero) to activate the communication driver to continuously check the tag value for changes. If a change occurs at this tag value, this value is written by the project to an address in the remote device.
- **Write completed**: Enter a tag that is to be updated by the communication driver if a "write" command is completed.
- **Write status**: Enter a tag that is to be updated with the status on page 977 of the last "write" command by the communication driver.
- **Station**: Enter the device station number within the network. The syntax in this field varies according to the communication driver. For further information, refer to the documentation of the respective driver.



If the driver is configured for serial encapsulation via TCP/IP or UDP/IP, the station can be specified for some drivers by means of the following format:

```
IP_adresse:port_nummer|station
```

Example:

```
10.169.25.18:1234|Station5
```

For information whether this function is available for the selected driver, please refer to the documentation of the respective driver.

- **Header** field: Specify the data type and/or initial address to read or write in the device. The syntax in this field varies according to the communication driver. For further information, refer to the respective <driver documentation>.



In the **Station** and **Header** fields, text with tag values using the `text {tag}` syntax can be entered.

- **Min** and **Max**: Enable this option to specify the minimum and maximum values for remote device data.
- **Min** and **Max** (become active only if the **Min and Max** checkbox is enabled): Enter a range of values to be converted into an engineering unit. These fields determine the minimum and maximum range of values. These values affect all tags in the worksheet. For example, the memory contains values from 0 to 4095, i.e., 0% to 100% in the user interface. In this example, 0 to 100 has to be specified for the min and max tag parameters.

Use the "body" area parameters as follows:

- **Tag name**: Enter the name of a tag to be linked to the device address.
- **Address**: Enter a remote device address (or address offset) related to the project tag. The syntax in this field varies according to the communication driver. For further information, refer to the documentation of the respective driver.
- **Div**: Specify a division constant to use in case scale adjustment is required. The project uses this value as a division factor in a read operation and a multiplication factor in a write operation. Do not use this field if **Min** or **Max** is used in the configuration body.
- **Add**: Specify an addition constant to use in case scale adjustment is required. The project uses this value as an addition factor in a read operation and a subtraction factor in a write operation. Do not use this field if **Min** or **Max** is used in the configuration body.



The maximum number of tags in each driver communication worksheet is 512. For some drivers, this number may be smaller. For more information, please refer to your <driver documentation>.

For read operations, the following applies:

```
Tag = (value in the device) / Div + Add
```

For write operations, the following applies:

```
Value in the equipment = (tag - Add) * Div
```

If the cells in the **Div** and **Add** fields are left empty, this function is ignored.

11.2 Inserting tags using the tag wizard

When configuring driver sheets, the tag wizard provides support. Process variables can thus be inserted into a WinStudio driver sheet and the corresponding WinStudio tags (variables) can be created in a few steps. The tag wizard suggests a WinStudio tag name for each variable to be inserted into the driver sheet and selects the corresponding data type for the tag.

Requirements

To select variables by using the process variable browser, proceed as follows:

1. Generate the symbol configuration for the PLC variables of the corresponding control.

   The control configuration has to be in the same project.
2. Create a new OPC-UA connection and select the corresponding control as server.

3. Open the OPC-UA driver sheet or select the created OPC-UA configuration under "Connection".

The tag wizard opens automatically if one or multiple variables are dragged from the process variable window and dropped to the WinStudio driver sheet using the mouse. A prerequisite is that the control, whose process variables should be used, is located in the same IndraWorks project as the visualization device.

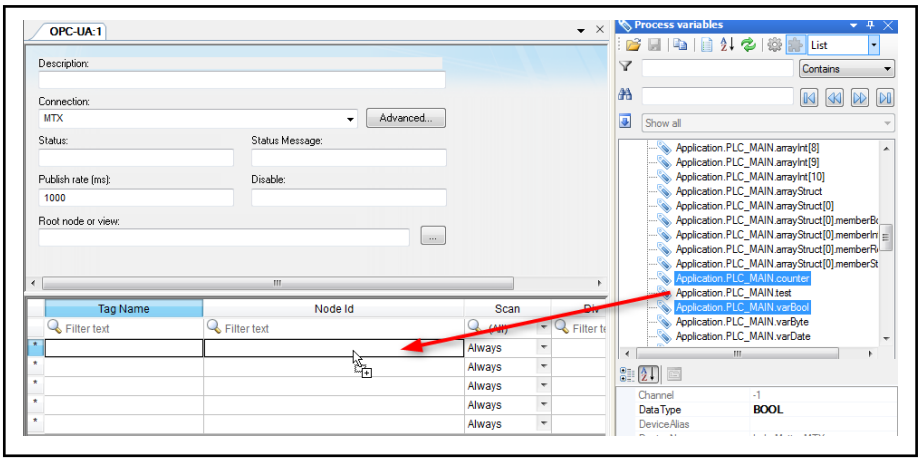


Fig. 11-10: Inserting process variables into the WinStudio driver sheet

The tag wizard displays all variables selected in the process variable browser in a list with further information. The names of the inserted variables are displayed in column (3). The name of the process variable without namespace is suggested as name of the WinStudio variable. To change the name, click on the corresponding field and edit the text. When leaving the input field or confirming the entry with <RETURN>, the status in column (1) is updated. The displayed icon indicates whether the variable is valid. The tooltip of the icon shows more information on the entered variable.

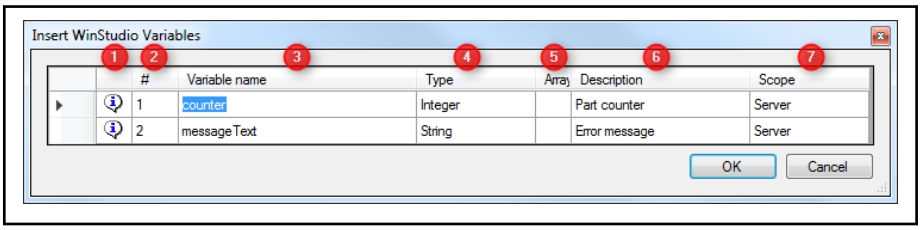


Fig. 11-11: Tag wizard

If one or multiple lines display an error, the tag wizard can only be closed after the error was recovered or by discarding all entries with "Cancel". The error icon displayed can be caused by the following:

- The name contains invalid characters (spaces and special characters)
- The variable is already used in the driver sheet.
- The variable already exists in the WinStudio variable list, but the data type differs from the data type displayed in column (4).

The data type displayed in column (4) is a WinStudio basic data type (Bool, Integer, Real, String). The data type matching the corresponding process variable is automatically selected and cannot be changed. If the inserted process variable is a one-dimensional array, column (5) displays the array size.



Multi-dimensional arrays as well as arrays of structures are not supported by the tag wizard.

Column (6) can be used to enter a description text for the respective WinStudio variables. This text is applied to the variable list and can subsequently be changed.

Column (7) displays the validity range of the variables (server, local).

Press "OK" to create the corresponding WinStudio variables and to apply the driver sheet. Additionally, the corresponding communication items can be inserted into the driver sheet.

☞ Delete lines from the variable list of the tag wizard by selecting the line to be deleted. Then, right-click to open the context menu and select the item "Delete Line". This function can for example be used to delete lines with name or data type conflicts and to insert only the remaining variables.

☞ The tag wizard supports the driver sheets for the OPC-UA, BR_WS and OPC (Iw.SCP.OPC) communication.

☞ For array variables, an individual line is inserted for each array element into the driver sheet.

11.3 Adding classes using the class wizard

As for the tag wizard, the class wizard supports you when inserting structural variables. A WinStudio class matching the structural variable can be selected or created in a few steps. Additionally, the communication items for the individual class elements can be inserted into the driver sheet.

The class wizard opens automatically if a **single** structural variable from the process variable browser is inserted into an open driver sheet via drag&drop.

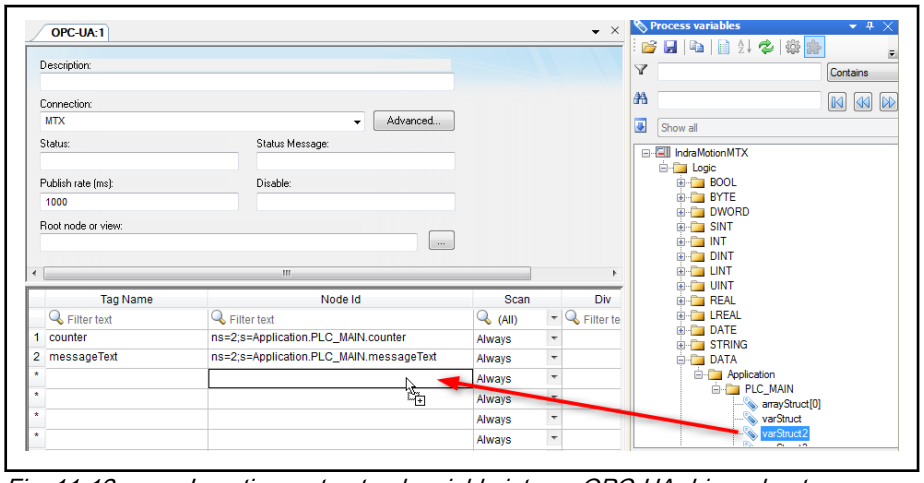


Fig. 11-12: Inserting a structural variable into an OPC-UA driver sheet

The complete name of the inserted structural variable including the namespace is displayed in the "Name" (1) field of the class wizard. The PLC variable name was specified in the PLC program code and cannot be changed in the class wizard anymore. The names of the WinStudio variables to be created for these structural variables are displayed in the "Name (Tag)" (2) field. It corresponds to the default setting of the basic name of the PLC variable. The name of the class variables to be created can be freely changed. Empty and special characters as well as already existing variable names are not permitted.

The name of the classes to be created for the current PLC structural variables is displayed in the "Type" (3) field. The field is pre-assigned with the basic name of the PLC structural variables and the prefix "class".

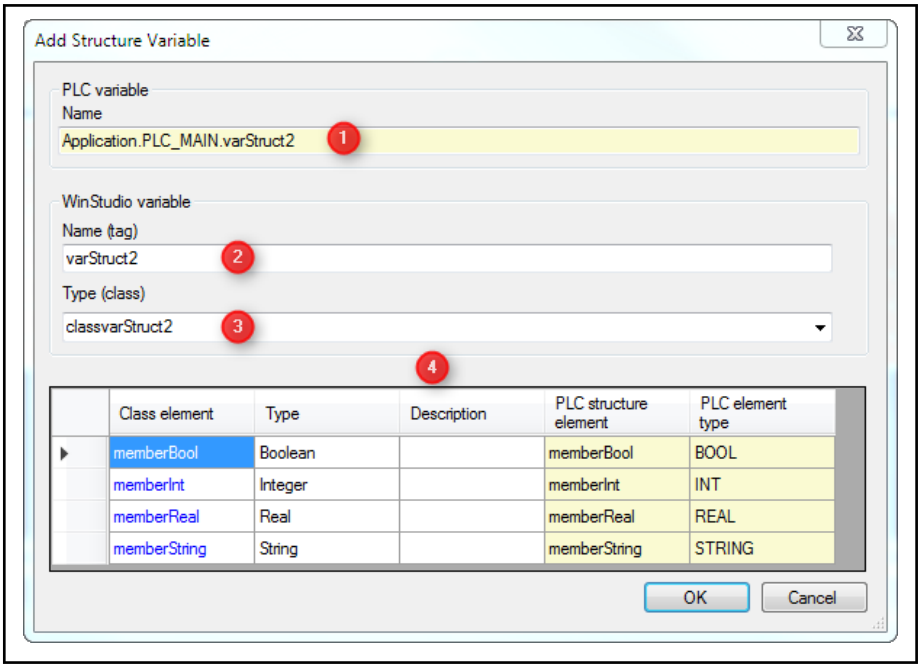


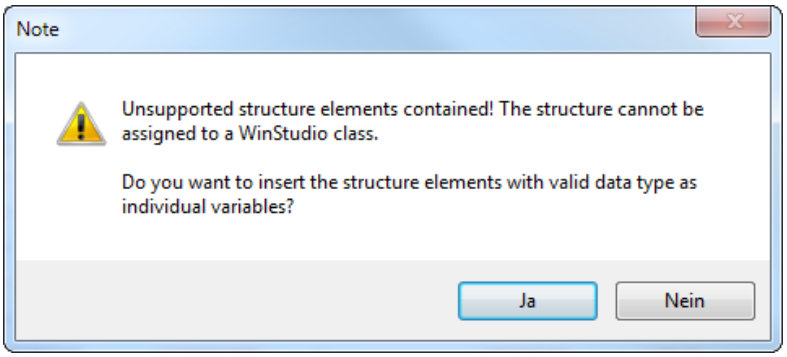
Fig. 11-13: Class wizard

A table (4) with all elements of the PLC structural variables as well as the respective data types is located in the lower section of the class wizard. The suggested names and data types of the class elements are displayed in the columns "Class Element" and "Type". When a new class name is entered into (3), the names of the class elements can be edited in table (4) and a description text can be assigned to each class element. When adding more instances of the same PLC structure to the driver sheet, it is often reasonable to use the previously created WinStudio class again. In this case, select the already existing WinStudio class in the drop-down list (3). The names an data types of the existing class elements are displayed in the columns "Class Element" and "Type". If the selected WinStudio class does not match the PLC structure with regard to the number and data types of the class elements, an error icon is displayed next to the field "Type".

Press "OK" to create the following entries:

- A WinStudio class in the folder "WinStudio\Variables\Classes"
- A WinStudio variable of the type of the added class
- One line each in the driver sheet for each structural element with corresponding communication item.

Only WinStudio classes for PLC structural variables containing elements with simple data types can be created. For nested structures as well as for structures with arrays, no class can be created. In this case, the following note is displayed:



Confirm with "Yes" to open the tag wizard. In this case, the tag wizard displays the name of a new variable for each structural element in the "Variable Name" column.

11.4 Configuring an OPC client connection to an OPC server (OPC)

OPC worksheets are used to communicate with any system that implements the OPC Server protocol.

The OPC client implements the OPC standard as described in document "OLE for process control data access standard version 1.0A". This document is available on the OPC Foundation website.

Before using the OPC client task/worksheet in the project, make sure that the OPC server software is properly installed and configured on the system to which the connection is to be established.

To configure a new connection, insert a new OPC worksheet in the **WinStudio** ▸ **Variables** ▸ **Communication** ▸ **OPC** folder in the Project Explorer.

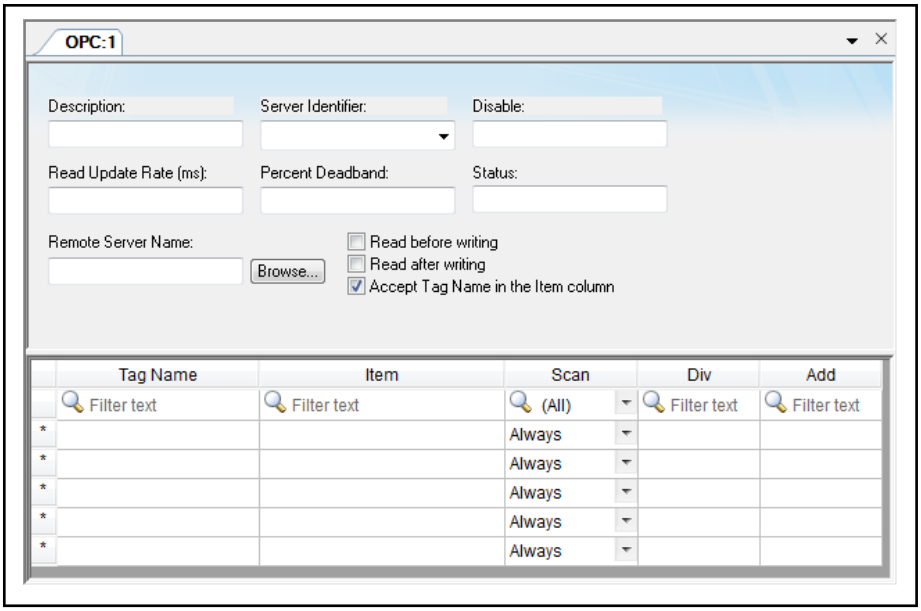


Fig. 11-14: Example of an OPC worksheet

To configure an OPC driver sheet, use the following parameters:

- **Description**: Enter a description of the OPC worksheet for documentation purposes.
- **Server identifier**: Enter the name of the server to connect to. If the server is already installed on the computer, the server name can be selected from the list.
- **Disable**: Enter a tag or a constant with a value other than "0" to disable communication with the OPC server. Enter "0" or leave the field blank to enable communication.
- **Read update rate (ms)**: Specify how often the server is to update this group (in milliseconds). Enter "0" to specify the fastest practical rate for the server.
- **Percent dead band** (available for analog items only): Specify how much percent change in an item value is to cause a notification by the server.
- **Status**: Enter the name of a tag to receive the status of the connection. The status "GOOD" corresponds to a value of 1.
- **Remote server name**: Node name or IP address of the server on the node network.
- **Read before writing**: If this option is enabled, the original values are read from the OPC server directly before writing new variable values to the server. The project does this by first buffering the new values to be written and then reading the original values from the server. The new values are not written from the buffer to the server before the project is synchronized with the server.
- **Read after writing**: If this option is enabled, the values from the OPC server are read back directly after writing new variable values to the server.

NOTICE

The "Read before writing" and "Read after writing" options are offered as the OPC client/server specifications define that the value of an element on the client – in this case, the project – is not to change unless the change is initiated by the server. This way, the client always stays in sync with the server.

The project, however, may be designed to change the values according to runtime processes or user input. Thus, the best way to change the values while staying in sync with the server is to make it seem like the changes are initiated by the server. If both options are enabled, the following sequence of events is performed on every scan of the OPC worksheet:

1. The new values on the client are buffered.
2. The original values on the server are read by the client, i.e., the client is synchronized with the server.
3. The new values are written from the buffer to the server.
4. The new values on the server are read by the client, i.e., the client is synchronized with the server again.

At the end of each reading/writing cycle, the values reflect the project status even though, technically speaking, the project is just staying in sync with the server.

In most projects, both options are to be enabled. In some projects, however, this may cause items to bounce between the original values and the new values. If this is a problem, try moving those items to another OPC worksheet where the **Read before writing** and **Read after writing** options are disabled.

- **Accept tag name in the item column**: If this option is enabled, the text enclosed in curly brackets in the "Item" field is interpreted as a tag name (String type). In this case, the value of this tag is used as item name by the OPC server. This enables changing the link between a tag and an OPC item during runtime by changing the values of the configured tag. If the **Accept tag name in the item column** option is disabled, all characters configured in the "Item" column are considered part of the item name (including the characters in curly brackets).
- **Tag name**: Enter the names of tags to be linked to the server items.
- **Item**: Enter the name of the server item. After selecting the OPC server, items can be selected from the server using the OPC browser. Right-click in the **Item** field and select the **OPC browser** option.



In this field, tag names can be entered between curly brackets (e.g., `{TagName}`). This way, item names can be changed dynamically during runtime.

- **Scan**: Specifies the condition under which the tag value is read from the remote device or server and then updated in the project database. For this, the following options apply:
  - **Always** means the tag is read and updated during every scan of the communication worksheet, regardless of whether the tag is used in any other WinStudio screens, scripts, or worksheets. This option is recommended for tags that have to be monitored continu-

ously in the background, such as tags that trigger alarms, tags used in recipes, tags that are recorded in the historical database, etc.

- **Screen** means the tag is read and updated only if it is being used in at least one open screen, either locally or on another client station. This option is recommended for tags that are exclusively used in screens as these do not need to be updated frequently. Selecting this option can improve project performance.
- **Auto** means the project automatically selects either **Always** or **Screen**, depending on where the tag is used in the project. If the tag is only used in one WinStudio screen, the scan is set to **Screen** by default. If the tag is used in different worksheets (e.g., script, math, alarm, trend, recipe, report, scheduler), the scan is set to **Always** at runtime.

If it is not clear, which option is to be selected, select **Always**. This guarantees that the tag is read and updated.

- **Div**: Specifies the division constant if scale adjustment is required. This value is a division factor in a read operation and a multiplication factor in a write operation.
- **Add**: Specifies the addition constant if scale adjustment is required. This value is an addition factor in a read operation and a subtraction factor in a write operation.

To execute the OPC client task during project runtime, open the task configuration (**Tools** ▸ **Status** in the "WinStudio" folder context menu) and select the default setting "Automatic" for "OPC client runtime". In this dialog, the task can also be started manually for test purposes by clicking on **Start**. After running this program, a small icon is displayed in the system tray.

To close the OPC Client runtime task, right-click the icon in the system tray and click **Exit**.



"WinStudio" and CEView also provide an OPC Server communication task named "Studio.Scada.OPC". This task starts automatically if any OPC client (local or remote) attempts to connect to the "Studio.Scada.OPC" server. An OPC client can exchange data with the tags database (project tags, system tags, and shared tags) using the OPC interface.

In addition, the OPC Server task can be started automatically on running the project. Select the OPC server task in the task configuration (**Status** entry in the WinStudio context menu), click on **Start** and select **Automatic**.

Reading Array Items on the Server

The OPC Client runtime task is capable of reading array items on the specified OPC server. However, it cannot read only certain elements of an array item; the entire item is read during every scan. Also, the way in which the values of the individual array elements are written to project tags in WinStudio depends on what tag you configure.

Scenario 1: The project tag is an array (i.e., the tag's Array property is greater than 0)

In this scenario, each element of the array item on the server is written one-for-one to an element of the project tag. If you specify an array index (e.g., `MyArrayTag[3]`), then writing will begin with that element. If you do not specify an array index, then writing will begin with element 0 by default.

You must make the **Array** property of the project tag large enough to accomodate the elements of the array item on the server. If it is not large enough, then elements will be lost.

Scenario 2: The project tag is not an array, but the tag name ends with a number

In this scenario, you can specify the first of a series of sequentially numbered project tags, and the values of the array elements are written to those tags.

When Tag Name is configured as…	…the values of the array element are written to the following tags:
`MyTag1`	`MyTag1, MyTag2, MyTag3`
`MyTag3`	`MyTag3, MyTag4, MyTag5`
`MyTag001`	`MyTag001, MyTag002, MyTag003`

Tab. 11-1: Examples of sequentially numbered project tags



Actually you have to create enough project tags to accomodate the elements of the array item on the server. If there are not enough tags, then elements will be lost. Project tags cannot be created during runtime.

Scenario 3: The project tag is a class, and the class member ends with a number

This scenario is similar to Scenario 2 above - you can specify the first of a series of sequentially numbered class members, and the values of the array elements are written to those members.

When Tag Name is configured as…	…the values of the array element are written to the following members:
`MyClass.Member1`	`MyClass.Member1, MyClass.Member2, MyClass.Member3`
`MyClass.Member3`	`MyClass.Member3, MyClass.Member4, MyClass.Member5`
`MyClass.Member001`	`MyClass.Member001, MyClass.Member002, MyClass.Member003`

Tab. 11-2: Examples of sequentially numbered class members



Actually you have to create the class with enough members to accomodate the elements of the array item on the server. If there are not enough members, then elements will be lost. Class members cannot be created during runtime.

Scenario 4: The project tag is both a class and an array

If the class member ends with a number, then it is handled the same as in Scenario 3 above, regardless of the fact that the tag is also an array.

If the class member does not end with a number, then it is handled the same as in Scenario 1 above.

When Tag Name is configured as…	…the values of the array element are written to the following tags:
`MyArrayClass.Member`	`MyArrayClass[0].Member, MyArrayClass[1].Member, MyArrayClass[2].Member`
`MyArrayClass[3].Member`	`MyArrayClass[3].Member, MyArrayClass[4].Member, MyArrayClass[5].Member`
`MyArrayClass[3].Member1`	`MyArrayClass[3].Member1, MyArrayClass[3].Member2, MyArrayClass[3].Member3`
`MyArrayClass[3].Member3`	`MyArrayClass[3].Member3, MyArrayClass[3].Member4, MyArrayClass[3].Member5`
`MyArrayClass[3].Member001`	`MyArrayClass[3].Member001, MyArrayClass[3].Member002, MyArrayClass[3].Member003`

Tab. 11-3: Examples of tags that are both classes and arrays

This functionality is not supported in projects running on Windows Mobile or Windows Embedded Compact.

OPC debugging

If the use of OPC causes problems establishing the communication, the messages in the LogWin are to be checked first.

If the project is executed on a Windows Embedded target system, there are two ways to check the log:

1. Remote LogWin
2. Local log

For information on these logs, refer to Using the LogWin task on page 503.

If error messages are found in the log, look them up in the manual/help system and follow the documented steps for solving the problems. (Use <CTRL>>+ <F> to search the manual or use the Index to search in the context-related help system.)

To request technical support from the distributor, please provide the following information:

1. Log file
2. Software vendor and product name of the OPC server/client that is used
3. If possible, a copy or an evaluation version of the OPC server for testing purposes
4. The contact information of the technical support of the OPC server/client

Three possible errors and their solutions are listed below…

Security

Error code: `16#80070005 or -2147024891`

Error cause: If the OPC client tries to connect to the OPC server, the DCOM layer usually requires authentication. The computer that is running the OPC

server needs to recognize the user logged on to the OPC client computer, and the respective user requires access rights to the OPC server.

Solution: The first step is to create a user on both computers that has administrator privileges and the same password. Log on with this user to both computers and then try to establish the connection.

Resolving name

Error: `Could not connect to AdviseSink, error: -2147022986 (16#80070776)`

Error cause: There is a problem resolving the computer name.

Solution: This problem can be solved by specifying the IP address of the server instead of specifying the computer name.

Proxy for Windows CE

Error: `OPC server: IIndCP::Advise - Could not query callback interface: 16#80040155`

Error cause: The Windows Embedded device is missing the `OPCCOMN_PS.dll`.

Solution: Download the .dll to the device and register it. The `.dll` is with the "WinStudio" version, most probably at

`[…]\WinStudio\Redist\Wince x.x\processor\`

If the `.dll` cannot be found in the folder of the respective processor, contact the "WinStudio" distributor.

11.5 Configuring an OPC UA client connection to an OPC UA server

11.5.1 Configuring an OPC UA client worksheet

OPC UA driver sheets are used to communicate with any system that implements the OPC server protocol.

About OPC UA

OPC UA is intended as a platform and language-independent protocol that is also backwards-compatible with "classic" OPC systems. For more information, refer to http://www.opcfoundation.org/UA/.



This function includes cryptographic software written by Eric Young (eay@cryptsoft.com).

Configuring a new OPC UA connection

To communicate with a device via the OPC UA server, the connection to this server has to be configured first. To do so, proceed as follows:

1. Open the context menu of the "OPC UA" folder and select the entry "OPC UA Connection Configuration".

   A dialog is opened displaying all previously created configurations. To create a new connection to an OPC UA server, press "Add".

   The "UA server configuration" dialog is displayed.

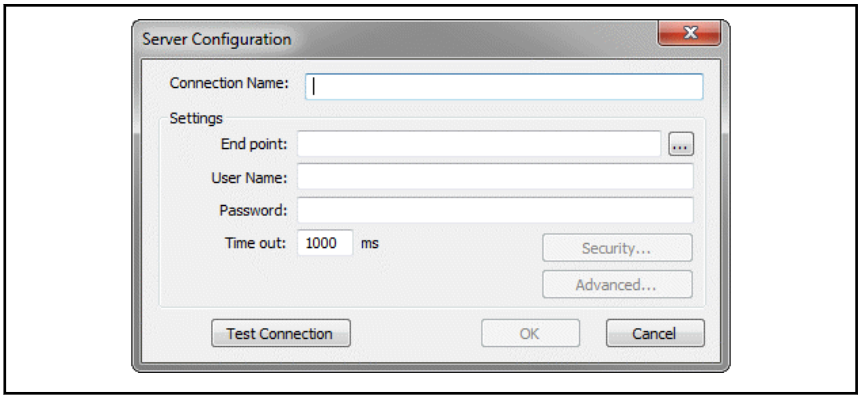


Fig. 11-15: "UA server configuration" dialog

2. In the **Connection name** box, enter a name for the connection. This name is displayed in the OPC UA connections folder in the Project Explorer and is the name to be looked for when configuring the OPC UA client worksheet.
3. In the **End point** box, enter the URL of the OPC UA server to be connected to.

☞ In Rexroth WinStudio, OPC UA at the moment only supports end points with OPC.TCP (e. g. `opc.tcp://host:8000/server`). HTTP and HTTPS are not supported.

If the URL is unknown, click the browse button to the right of the box. The "UA discovery" dialog is displayed.

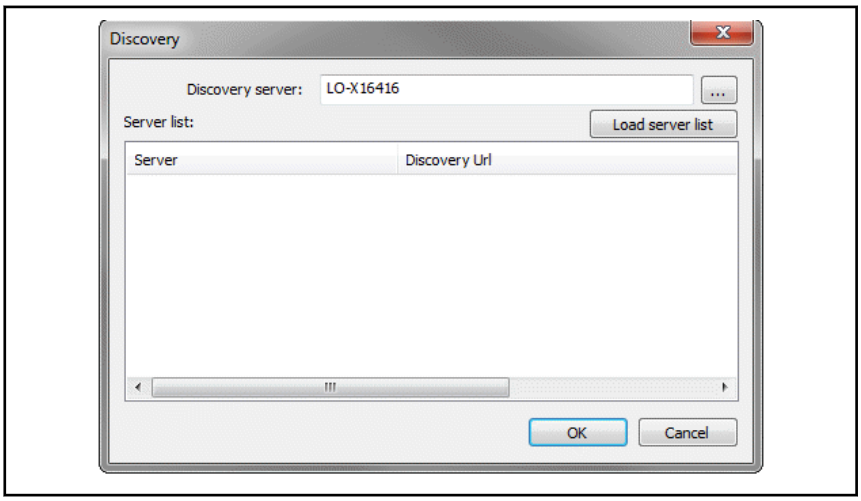


Fig. 11-16: "UA discovery" dialog

Use this dialog to find the discovery server, which publishes a list of OPC UA servers on the network and select the server to connect to.

4. In the **User name** and **Password** field, enter the login data for the OPC UA server.
5. If the OPC UA server is configured in a way that requires secure communication (also referred to as "SecureChannel" in the OPC UA specification), additional steps have to be taken to exchange certificates between this server and the OPC UA client task in Rexroth WinStudio. WinStudio uses a file-based certificate memory. This means that the

certificates are saved as files in the project folder and not as entries in a database or a system registry.

The steps to exchange the certificates and to configure the security settings are slightly different depending on whether you use self-signed certificates or certificates signed by a trusted certification authority (CA):

- Connecting to an OPC UA server using self-signed certificates, page 442
- Connecting to an OPC UA server using CA-signed certificates, page 444

Additional information in the “Certificate Management” section of the document "OPC Unified Architecture Specification, Part 2: Security Model".

Download the document from the OPC foundation website: (www.opc-foundation.org).

6. In certain situations, it may be necessary to adjust the server connection by modifying the different timeouts. In this case, click **Advanced** to open the following dialog box:

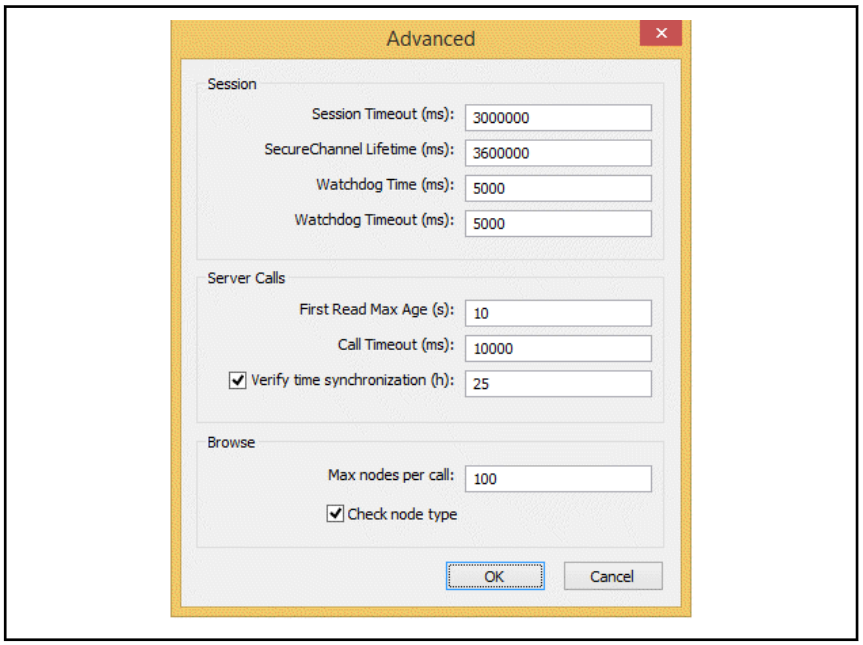


Fig. 11-17: Advanced dialog box

Review the settings configure them as needed:

Session

Session timeout

The session timeout (in milliseconds) for the server connection. If a session times out due to inactivity, a new session must be started to resume communication.

SecureChannel lifetime

The time (in milliseconds) after which the SecureChannel between client and server is renewed automatically.

For more information about the SecureChannel services see the OPC UA specification.

Watchdog time

The time (in milliseconds) between the Watchdog checks.

Watchdog timeout

The timeout (in milliseconds) for a specific Watchdog check.

Server calls

Max. first read age

The maximum acceptable age (in seconds) of the element values when the client connects to the server and requests the values for the first time. If the age of a value in the cache of the server is below the maximum age, the server returns this cached value to the client. If the age of a value in the cache of the server, however, exceeds the maximum age, the server updates the value from the field before returning it to the client.

In addition to an actual age in seconds, two special values may be in this field:

Value	Description
-1	Set the maximum age to a maximum possible value of an integer, i. e. 2147483647 seconds or approximately 68 years. This has the convenient effect that all cached values are accepted, independent of their age.
-2	Ignore the first reading altogether and wait until the server regularly updates the element values. Please note that the client cannot write on the server before the server has read.

Call timeout

The timeout (in milliseconds) for individual server calls.

Verify time synchronization

If this option is selected, the time stamp of each value which comes from the server is compared to the system time on the client. If the difference between the times corresponds to the indicated number of hours (i. e. the number of time zones between server and client), server and client are considered as synchronized and the value is accepted. Otherwise, if the difference does not correspond to the indicated number of hours, the value is rejected and the communication is interrupted.

The default number of hours is 25. This is a special setting which means that the hour part of the time stamp is to be ignored and only the minutes and seconds are compared. In this way, the client can receive values from the server without knowing the time zone of the server.

If this option is not selected, the time stamp of each value which comes from the server is ignored and the system time of the client is used instead to save the value in the project database.

Browse

Max. nodes per call

The maximum number of elements/nodes which are output per server call when browsing for a server element (see below).

Check node type

Check the data type of the element when browsing for a server element (see below). In some situations, the OPC UA server may not be able to receive data types from field devices. This may cause browsing to become slow and ineffective. If this is the case, deactivate the option.

7. Click on **OK**. The connection is saved in the OPC UA connections folder in the Project Explorer.

Configuring a new OPC UA Client worksheet

To create a new driver sheet for the OPC UA client, proceed as follows:

1. Open the context menu of the "OPC UA" folder and select "New OPC driver sheet".

   A new element is created in the UPC UA folder. Now, double-click the driver sheet editor to open it:

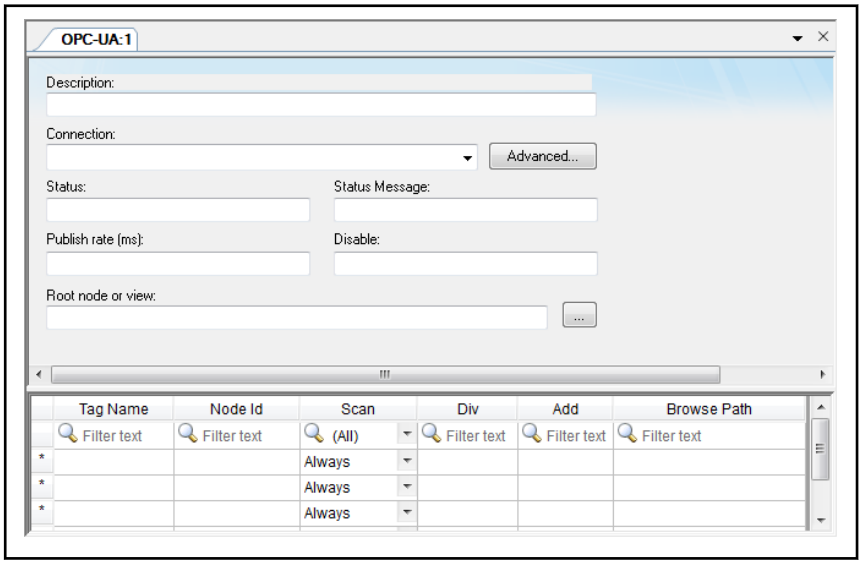


Fig. 11-18: OPC UA client worksheet

2. In the **Description** field, enter a description of the worksheet. This is for documentation purposes only and does not affect the execution of the worksheet.
3. In the **Connection** list, select the server connection configured earlier.
4. For further connection settings, click on **Advanced**. The advanced settings dialog is displayed:

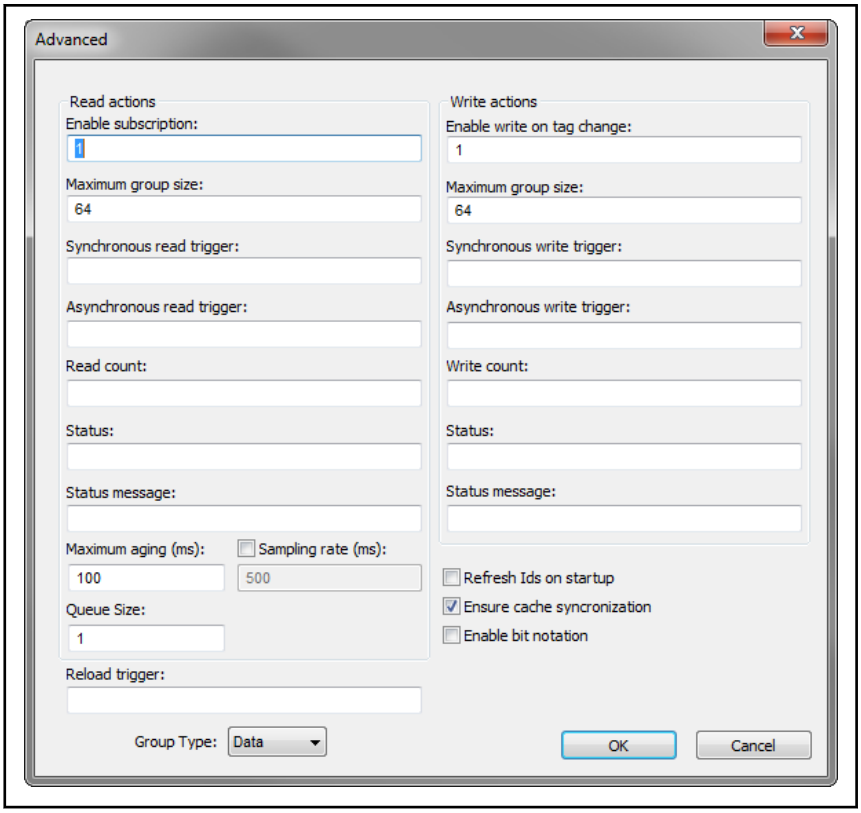


Fig. 11-19: OPC UA client worksheet, advanced

Review the options and configure as needed:

Read actions

Enable subscription

If this value is TRUE (not zero), the client subscribes to the server nodes to receive notifications if the node value changes.

Note:

This option is enabled by default. If it is disabled, read triggers (see below) have to be used.

Maximum group size

The maximum number of server nodes that can be read in a single read operation.

For example, if 1000 elements/rows are configured in the worksheet and the **Maximum group size** is set to 100, 10 read operations are performed when reading is triggered.

Synchronous read trigger

If the value of this tag/expression changes, the client reads all node values from the server. The read operation is performed synchronously. This means that all other client processes are blocked until the read operation is completed. When the read operation is completed, the client increases the tag which was configured under **Read count** step by step.

Asynchronous read trigger

As with the **Synchronous read trigger** but the read operation is performed asynchronously. This means that all other client processes can continue while the read operation is performed.

Read count

The name of a project tag (Integer type) that receives a count of the number of read operations performed since the project runs.

Status

The name of a project tag (Integer type) that is to receive a status code for the last read operation performed by a trigger. If the status code is 0, the read operation has been finished successfully. All other status codes can be found in the List of read/write status codes and messages for OPC UA, page 445.

Status message

The name of a project tag (String type) that is to receive the corresponding status message.

Maximum aging

The maximum age (in milliseconds) of values that are to be accepted from the server cache. If a value is older than this, the server is forced to get the latest value from the field device.

Sampling rate

If this option is selected, you can change the rate (in milliseconds) at which the OPC server reads from field devices. By default, the sampling rate is half the publishing rate.

Write actions

Enable write on tag change

If this value is TRUE (not zero), an asynchronous write operation is automatically performed whenever the value of the project tag changes. All tag changes since the last worksheet scan are written in a single write operation or in several write operations if the number of tag changes exceeds the maximum group size.

Note:

This option is enabled by default. If it is deactivated, write triggers (see below) are to be used.

Maximum group size

The maximum number of tag changes which can be written in a single write operation.

For example, if 1000 items/rows are configured in the worksheet and the **Maximum group size** is set to 100, 10 write operations are performed during each scan of the worksheet.

Synchronous write trigger

If the value of this tag/expression changes, the client writes all tag values on the server. The write operation is performed synchronously. This means that all other client processes are blocked until the write operation is completed. If the write operation is completed, the client increases the tag which was configured under **Write count** step by step.

Asynchronous write trigger

As with the **Trigger for synchronous writing** but the write operation is performed asynchronously. This means that all other client processes can continue while the write operation is performed.

Write count

The name of a project tag (Integer type) that is to receive a count of the number of write operations performed since the project runs.

Status

The name of a project tag (Integer type) that is to receive a status code for the last write operation performed by a trigger. If the status code is 0, the write operation has been finished successfully. All other status codes can be found in the List of read/write status codes and messages for OPC UA, page 445.

Status message

The name of a project tag (String type) that is to receive the corresponding status message.

Reload trigger

The name of a project tag (Boolean, Integer or Real type) which can be used as a trigger. If the value of this tag changes, the worksheet is reloaded. Tags in curly brackets (e. g. `{tag name}`), which are configured in the main part of the worksheet, are only renewed if the worksheet is reloaded. If the worksheet is reloaded, the value of this tag is reset to 0.

Group type

At the moment, this function is not supported.

Ensure cache synchronization

If this option is selected, the client waits for the confirmation of the server that the node values have actually changed after each write operation. If the client does not receive any confirmation, it retrieves the previous tag values.

Enable bit notation

If this option is selected, bit notation is admissible in the Element column of the main part of the worksheet.

☞ Normally, the "WinStudio" OPC client does not apply the “Trigger” mode that is defined in the OPC protocol. Instead, it allows any change in any tag/expression to be used as a trigger. If the “Trigger” mode is to be used, configure one worksheet to read the trigger values and configure another worksheet that specifies the read values as triggers values.

---

5. Click on **OK** to close the "Advanced" dialog and return to the worksheet.
6. In the **Status** field, enter the name of a tag (Integer type) that is to receive connection status codes at project runtime.
7. In the **Status message** field, enter the name of a tag (String type) that is to receive the corresponding status messages.
8. In the **Publish rate** field, enter the frequency (in milliseconds) at which the client is to request updates from the server.

9. In the **Disable** field, enter a tag/expression. If the value is TRUE (not zero), the worksheet is not executed.
10. In the **Root node or view** field, specify the server node that is to serve as the root for all items in the worksheet body. Specifying a root node makes it easier to find items and improves runtime performance.
11. For each row of the worksheet body, the following information is to be specified:
    - **Tag name**: The name of a tag. If indirect tags (e.g., `{MyTag}`) are to be used, the **Reload trigger** option described above has to be enabled.
    - **Item**: The server item the tag is to be associated with. To browse the server list of items, right-click in this field and select **Browse** in the context menu.

      Note:

      If **Enable bit notation** is selected in the advanced settings, an individual bit of a server node value can be selected by attaching the bit number to the element name (e. g. `nodepath.bit`). Please note that this is only supported for variables without prefixes and variables signed with 32 bits; if this is done using variables signed with 16 bits or 8 bits, the most important bit (MSB) does not work properly.
    - **Scan**: If **Always** is selected, the row is continuously scanned during runtime. If **Screen** is selected, the row is scanned only in case a WinStudio screen that uses the specified tag is opened.
    - **Div**: For scaling – the value is divided by this number when reading from the server and multiplied by this number when writing to the server.
    - **Add**: For scaling – this number is added to the value when reading from the server and subtracted from the value when writing to the server.
    - **Node ID**: The node ID is automatically generated from the full path of the server item.



If the OPC server element is a one-dimensional array, you can link it to an array tag in WinStudio using one single worksheet row without array indices. The elements of the OPC server element are automatically linked to the elements of the array tag, one after the other.

Please note that this function only supports the reading of the OPC server element.

12. Save and close the worksheet.

Enabling the OPC UA client task

To enable the OPC UA client task for runtime, proceed as follows:

1. Open the task configuration via **WinStudio** ▸ **Tools** ▸ **Status**. The "Execution tasks" dialog is displayed.
2. In the list of tasks, select **OPC UA client runtime**.
3. Click on **Start**.

   The "Start" dialog is displayed.
4. Select **Automatic** and click on **OK**.
5. Click on **OK** to close the "Execution tasks" dialog.

Debugging

If there are any problems with the operations of the OPC UA client, have the client create a trace log with more detailed information. To do this, change your project file (project_name.app) manually or add the following setting:

```
[OPC]
UaLogPath = Path and name of the file
UaTraceLevel=Level
```

For `UaLogPath`, the file path is relative to your project folder. For `UaTraceLevel`, select one of the following options:

Level	Description
0	No trace - Deactivates the trace. This is the default setting if no UaLogPath is configured.
1	Error – Internal system errors which require troubleshooting.
2	Warnings – Internal system warnings and external errors. This is the default setting if an UaLogPath is configured.
3	Info – More detailed information regarding system events.
4	InterfaceCall – Required information for troubleshooting.
5	CtorDtor – Required information for troubleshooting.
6	Program – The entire message content. 7 FlowData – All messages.

☞ The levels are cumulative, i. e. level 2 includes level 1 and level 3 includes levels 1 and 2 and so on.

11.5.2 Connecting to an OPC UA server using self-signed certificates

This task describes how to configure the OPC UA server and the client in a way that ensures secure communication via self-signed certificates.

This task supplements the previous task, Configuring an OPC UA client worksheet, page 433. It is assumed that a new OPC UA connection has already been established and now the security settings for this connection are to be configured.

A self-signed certificate is a certificate which is signed by the person or organization using it. In most cases, the use is only secure if you directly control both the server and the client. If you do not control both directly, certificates which are signed by a trusted certification authority (CA) are to be used.

Please note that in this task "OPC UA server" is a generic reference to all available hardware or software products which can run as an OPC UA server. For more information on how to perform individual steps, please refer to the manufacturer's documentation of the respective server. Our examples are based on the free OPC UA sample server and the UA configuration tool by the OPC Foundation. However, these examples are to be used as reference only.

Connecting to an OPC UA server using self-signed certificates:

1. Create a new self-signed certificate for the server application for your OPC UA server. If an old server certificate is in the certificate memory, you may be prompted to overwrite or delete it. This should be secure as long as no other clients are connected to the server. Example: Run the UA configuration tool. Click on **Manage application**.

Ensure that **Opc.Ua.SampleServer** is selected in the **Application to be managed** list and click on **Create application certificate**.

The Create certificate dialog box is displayed. Configure the certificate settings as desired but leave the **CA key file** and **CA password** fields empty; this makes the certificate self-signed. Click on **OK** to create the new certificate. Then restart the OPC UA sample server to ensure that it uses the certificate.

2. In the UA server configuration dialog box in Rexroth WinStudio click on security. The Security settings dialog box is displayed.

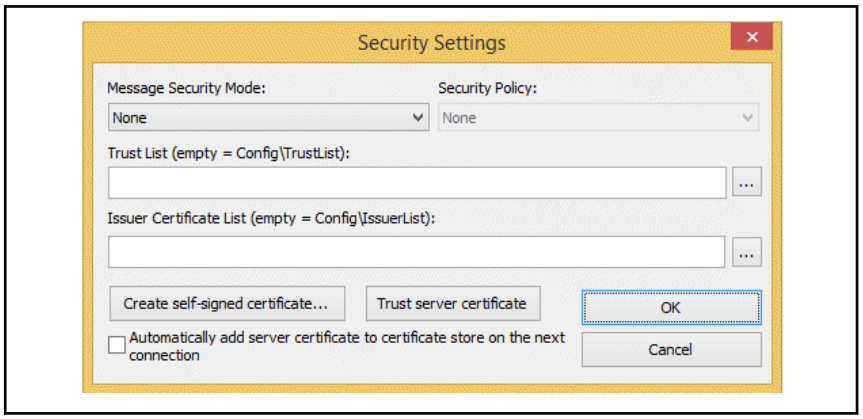


Fig. 11-20: Security settings dialog box

3. From the **Message security mode** and **Security policy** lists, select the required settings for your OPC UA server. In a typical configuration, the **Message security mode** is **Signing and encoding** and the **Security policy** is **Basic128Rsa15**.
4. Click **Trust server certificate**. In most cases, the **Trust list** or **Issuer certificate list** do not need to be changed. These are standard locations in your project folder where the certificates are saved. It is possible to change the locations if, for example, a single folder is used to save certificates of several different programs but we do not recommend it. The development applications attempts to establish a connection to the OPC UA server. If this is successful, it receives the server certificate. A warning dialog is displayed with the prompt to confirm your trust in the server certificate.
5. Click **OK** for confirmation. The server certificate file is copied in your project folder under: `[…]\Documents\WinStudio v7.4 Projects\projectname\Config\TrustList\Certs\connection-name.der`
6. Click on **Create self-signed certificate**. The Create certificate dialog box is displayed.
7. Fill in the required certificate information in the Create certificate dialog box and click **Create**. The development application creates a new client certificate file and saves it in your project folder under: `[…]\Documents\WinStudio v7.4 Projects\projectname\Config\UAClient-Certificate.der`
8. On your OPC UA server, import the client certificate (from the location indicated in the previous step) to the trust list of the server. Example: In the UA configuration tool click on **Manage security** and then on **Import trusted certificate**. A standard Open file dialog is displayed. Use the dialog box to find and select the client certificate and click **Open**. You are prompted to confirm the import. Click **Yes**.

When the certificates have been exchanged - this means when the server certificate is on the trust list of the client (i.e. your project) and the client certificate is on the trust list of the server - the OPC UA connection should be properly configured and ready for secure mutual communication.

To test the connection, add a new OPC UA client worksheet, select the connection you created and then try to browse for elements on the server.

11.5.3 Connecting to an OPC UA server using CA-signed certificates

This task describes how to configure the OPC UA server and the client in a way that ensures secure communication via CA-signed certificates.

This task supplements the previous task, Configuring an OPC UA client worksheet, page 433. It is assumed that a new OPC UA connection has already been established and now the security settings for this connection are to be configured.

In addition, this task assumes that a CA-signed certificate was purchased and you have the right to use it in your project. The certificate should be a Distinguished Encoding Rules file (`*.der`). In addition, the password for the certificate should be available.

A CA-signed certificate is a certificate which was signed by a trusted certification authority (CA). Using CA-signed certificates is secure in most situations even if you do not control both, the server and client, directly since you agree to trust the CA.

You can purchase certificates from certification authorities such as VeriSign, DigiCert and GeoTrust. Bosch Rexroth does not support any particular company.

On some servers it is possible to create your own certification authority. Do this, however, only if you are familiar with the related security problems. In particular, the difference between self-signed certificates and certificates signed by a self-created CA should be clear. However, this is beyond the scope of this documentation.

If a CA-signed certificate is available, it can be used to sign other certificates which are created for your server and client applications. Each subsequent certificate is considered as signed by the same trusted CA.

Please note that in this task "OPC UA server" is a generic reference to all available hardware or software products which can run as an OPC UA server. For more information on how to perform individual steps, please refer to the manufacturer's documentation of the respective server. Our examples are based on the free OPC UA sample server and the UA configuration tool by the OPC Foundation. However, these examples are to be used as reference only.

Connecting to an OPC UA server using CA-signed certificates:

1. On your OPC UA server, import the CA-signed certificate to the certificate memory of the server. The CA-signed certificate is used to sign server and client certificates which you create using the following steps. Example: Run the UA configuration tool. Click on **Manage certificates** and then on **Import certificate to be saved**. A standard Open file dialog is displayed. Use the dialog to find and select the CA-signed certificate and click **Open**. You are prompted to confirm the import. Click **Yes**.
2. Create a new, signed server certificate on your OPC UA server. If an old server certificate is in the certificate memory, you may be prompted to overwrite or delete it. This should be secure as long as no other clients are connected to the server. Example: Click on **Manage application** in the UA configuration tool. Ensure that **Opc.Ua.SampleServer** is selected

in the **Application to be managed** list and click on **Create application certificate**. The Create certificate dialog is displayed. Configure the desired certificate settings. In the **CA key file** field click on **Browse** to select the CA-signed certificate you imported in the previous step. Enter the password for the CA-signed certificate in the field **CA password**. Click on **OK** to create the new certificate. Then restart the OPC UA sample server to ensure that it uses the certificate.

3. Add the CA-signed certificate to the issuer list of your project - copy (do not drag and drop) the certificate file to your project folder under: `[…]\Documents\WinStudio v7.4 Projects\projectname\Config\IssuerList\Certs\` The issuer list determines which other certificates your project trusts. If your project is instructed to trust the signed server certificate, the certificate is compared to the CA-signed certificate in the issuer list to ensure that the signatures match.
4. In the UA server configuration dialog box in Rexroth WinStudio click on **Security**. The Security settings dialog is displayed.

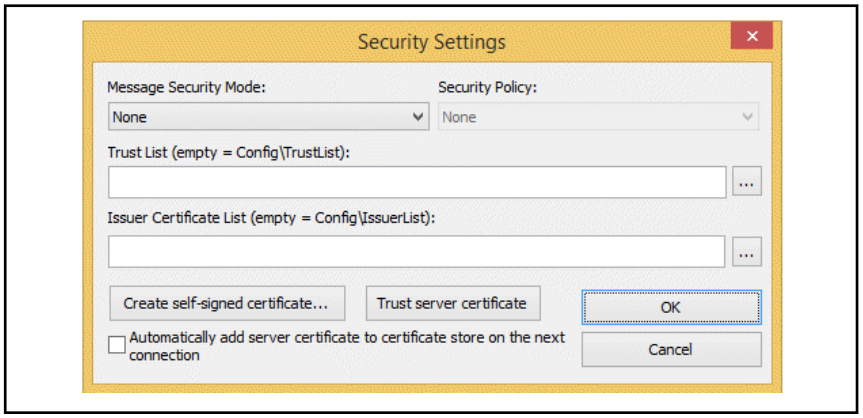


Fig. 11-21: Security settings dialog

5. From the **Message security mode** and **Security policy** lists, select the required settings for your OPC UA server. In a typical configuration, the **Message security mode** is **Signing and encoding** and the **Security policy** is **Basic128Rsa15**.
6.

11.5.4 List of read/write status codes and messages for OPC UA

This is a list of possible status codes and messages which can be generated by read/write operations in the OPC UA.

Status Code	Status Message
-2159476736	Bad – Max Connections Reached
-2159411200	Bad – Syntax Error
-2159345664	Bad – Would Block
-2159280128	Bad – Expected Stream To Block
-2159214592	Bad – Operation Abandoned
-2159149056	Bad – Waiting For Response
-2159083520	Bad – No Data Available
-2159017984	Bad – End Of Stream
-2147352576	Bad – Internal Error

Status Code	Status Message
-2147287040	Bad – Out Of Memory
-2147221504	Bad – Resource Unavailable
-2147155968	Bad – Communication Error
-2147090432	Bad – Encoding Error
-2147024896	Bad – Decoding Error
-2146959360	Bad – Encoding Limits Exceeded
-2146893824	Bad – Unknown Response
-2146828288	Bad – Timeout
-2146762752	Bad – Service Unsupported
-2146697216	Bad – Shutdown
-2146631680	Bad – Server Not Connected
-2146566144	Bad – Server Halted
-2146500608	Bad – Nothing To Do
-2146435072	Bad – Too Many Operations
-2146369536	Bad – Data Type Id Unknown
-2146304000	Bad – Certificate Invalid
-2146238464	Bad – Security Checks Failed
-2146172928	Bad – Certificate Time Invalid
-2146107392	Bad – Certificate Issuer Time Invalid
-2146041856	Bad – Certificate Host Name Invalid
-2145976320	Bad – Certificate Uri Invalid
-2145910784	Bad – Certificate Use Not Allowed
-2145845248	Bad – Certificate Issuer Use Not Allowed
-2145779712	Bad – Certificate Untrusted
-2145714176	Bad – Certificate Revocation Unknown
-2145648640	Bad – Certificate Issuer Revocation Unknown
-2145583104	Bad – Certificate Revoked
-2145517568	Bad – Certificate Issuer Revoked
-2145452032	Bad – User Access Denied
-2145386496	Bad – Identity Token Invalid
-2145320960	Bad – Identity Token Rejected
-2145255424	Bad – Secure Channel Id Invalid
-2145189888	Bad – Invalid Timestamp
-2145124352	Bad – Nonce Invalid
-2145058816	Bad – Session Id Invalid
-2144993280	Bad – Session Closed

Status Code	Status Message
-2144927744	Bad – Session Not Activated
-2144862208	Bad – Subscription Id Invalid
-2144731136	Bad – Request Header Invalid
-2144665600	Bad – Timestamps To Return Invalid
-2144600064	Bad – Request Cancelled By Client
-2144272384	Bad – No Communication
-2144206848	Bad – Waiting For Initial Data
-2144141312	Bad – Node Id Invalid
-2144075776	Bad – Node Id Unknown
-2144010240	Bad – Attribute Id Invalid
-2143944704	Bad – Index Range Invalid
-2143879168	Bad – Index Range No Data
-2143813632	Bad – Data Encoding Invalid
-2143748096	Bad – Data Encoding Unsupported
-2143682560	Bad – Not Readable
-2143617024	Bad – Not Writable
-2143551488	Bad – Out Of Range
-2143485952	Bad – Not Supported
-2143420416	Bad – Not Found
-2143354880	Bad – Object Deleted
-2143289344	Bad – Not Implemented
-2143223808	Bad – Monitoring Mode Invalid
-2143158272	Bad – Monitored Item Id Invalid
-2143092736	Bad – Monitored Item Filter Invalid
-2143027200	Bad – Monitored Item Filter Unsupported
-2142961664	Bad – Filter Not Allowed
-2142896128	Bad – Structure Missing
-2142830592	Bad – Event Filter Invalid
-2142765056	Bad – Content Filter Invalid
-2142699520	Bad – Filter Operand Invalid
-2142633984	Bad – Continuation Point Invalid
-2142568448	Bad – No Continuation Points
-2142502912	Bad – Reference Type Id Invalid
-2142437376	Bad – Browse Direction Invalid
-2142371840	Bad – Node Not In View
-2142306304	Bad – Server Uri Invalid

Status Code	Status Message
-2142240768	Bad – Server Name Missing
-2142175232	Bad – Discovery Url Missing
-2142109696	Bad – Sempahore File Missing
-2142044160	Bad – Request Type Invalid
-2141978624	Bad – Security Mode Rejected
-2141913088	Bad – Security Policy Rejected
-2141847552	Bad – Too Many Sessions
-2141782016	Bad – User Signature Invalid
-2141716480	Bad – Application Signature Invalid
-2141650944	Bad – No Valid Certificates
-2141585408	Bad – Request Cancelled By Request
-2141519872	Bad – Parent Node Id Invalid
-2141454336	Bad – Reference Not Allowed
-2141388800	Bad – Node Id Rejected
-2141323264	Bad – Node Id Exists
-2141257728	Bad – Node Class Invalid
-2141192192	Bad – Browse Name Invalid
-2141126656	Bad – Browse Name Duplicated
-2141061120	Bad – Node Attributes Invalid
-2140995584	Bad – Type Definition Invalid
-2140930048	Bad – Source Node Id Invalid
-2140864512	Bad – Target Node Id Invalid
-2140798976	Bad – Duplicate Reference Not Allowed
-2140733440	Bad – Invalid Self Reference
-2140667904	Bad – Reference Local Only
-2140602368	Bad – No Delete Rights
-2140536832	Bad – Server Index Invalid
-2140471296	Bad – View Id Unknown
-2140340224	Bad – Too Many Matches
-2140274688	Bad – Query Too Complex
-2140209152	Bad – No Match
-2140143616	Bad – Max Age Invalid
-2140078080	Bad – History Operation Invalid
-2140012544	Bad – History Operation Unsupported
-2139947008	Bad – Write Not Supported
-2139881472	Bad – Type Mismatch

Status Code	Status Message
-2139815936	Bad – Method Invalid
-2139750400	Bad – Arguments Missing
-2139684864	Bad – Too Many Subscriptions
-2139619328	Bad – Too Many Publish Requests
-2139553792	Bad – No Subscription
-2139488256	Bad – Sequence Number Unknown
-2139422720	Bad – Message Not Available
-2139357184	Bad – Insufficient Client Profile
-2139291648	Bad – Tcp Server Too Busy
-2139226112	Bad – Tcp Message Type Invalid
-2139160576	Bad – Tcp Secure Channel Unknown
-2139095040	Bad – Tcp Message Too Large
-2139029504	Bad – Tcp Not Enough Resources
-2138963968	Bad – Tcp Internal Error
-2138898432	Bad – Tcp Endpoint Url Invalid
-2138832896	Bad – Request Interrupted
-2138767360	Bad – Request Timeout
-2138701824	Bad – Secure Channel Closed
-2138636288	Bad – Secure Channel Token Unknown
-2138570752	Bad – Sequence Number Invalid
-2138505216	Bad – Configuration Error
-2138439680	Bad – Not Connected
-2138374144	Bad – Device Failure
-2138308608	Bad – Sensor Failure
-2138243072	Bad – Out Of Service
-2138177536	Bad – Deadband Filter Invalid
-2137587712	Bad – Refresh In Progress
-2137522176	Bad – Condition Already Disabled
-2137456640	Bad – Condition Disabled
-2137391104	Bad – Event Id Unknown
-2137325568	Bad – No Data
-2137194496	Bad – Data Lost
-2137128960	Bad – Data Unavailable
-2137063424	Bad – Entry Exists
-2136997888	Bad – No Entry Exists
-2136932352	Bad – Timestamp Not Supported

Status Code	Status Message
-2136276992	Bad – Invalid Argument
-2136211456	Bad – Connection Rejected
-2136145920	Bad – Disconnect
-2136080384	Bad – Connection Closed
-2136014848	Bad – Invalid State
-2135425024	Bad – Request Too Large
-2135359488	Bad – Response Too Large
-2135228416	Bad – Event Not Acknowledgeable
-2135097344	Bad – Invalid Timestamp Argument
-2135031808	Bad – Protocol Version Unsupported
-2134966272	Bad – State Not Active
-2134835200	Bad – Filter Operator Invalid
-2134769664	Bad – Filter Operator Unsupported
-2134704128	Bad – Filter Operand Count Mismatch
-2134638592	Bad – Filter Element Invalid
-2134573056	Bad – Filter Literal Invalid
-2134507520	Bad – Identity Change Not Supported
-2134376448	Bad – Not Type Definition
-2134310912	Bad – View Timestamp Invalid
-2134245376	Bad – View Parameter Mismatch
-2134179840	Bad – View Version Invalid
-2134114304	Bad – Condition Already Enabled
-2134048768	Bad – Dialog Not Active
-2133983232	Bad – Dialog Response Invalid
-2133917696	Bad – Condition Branch Already Acked
-2133852160	Bad – Condition Branch Already Confirmed
-2133786624	Bad – Condition Already Shelved
-2133721088	Bad – Condition Not Shelved
-2133655552	Bad – Shelving Time Out Of Range
-2133590016	Bad – Aggregate List Mismatch
-2133524480	Bad – Aggregate Not Supported
-2133458944	Bad – Aggregate Invalid Inputs
-2133393408	Bad – Bound Not Found
-2133327872	Bad – Bound Not Supported
-2133196800	Bad – Aggregate Configuration Rejected
2949120	Good – Subscription Transferred

Status Code	Status Message
3014656	Good – Completes Asynchronously
3080192	Good – Overload
3145728	Good – Clamped
9830400	Good – Local Override
10616832	Good – Entry Inserted
10682368	Good – Entry Replaced
10813440	Good – No Data
10878976	Good – More Data
10944512	Good – Communication Event
11010048	Good – Shutdown Event
11075584	Good – Call Again
11141120	Good – Non Critical Timeout
12189696	Good – Results May Be Incomplete
14221312	Good – Data Ignored
1080819712	Uncertain – Reference Out Of Server
1083113472	Uncertain – No Communication Last Usable Value
1083179008	Uncertain – Last Usable Value
1083244544	Uncertain – Substitute Value
1083310080	Uncertain – Initial Value
1083375616	Uncertain – Sensor Not Accurate
1083441152	Uncertain – Engineering Units Exceeded
1083506688	Uncertain – Sub Normal
1084489728	Uncertain – Data Sub Normal
1086062592	Uncertain – Reference Not Deleted
1086324736	Uncertain – Not All Nodes Available

Tab. 11-4: Status code - status message

11.5.5 Activating the OPC UA trace log

If problems occur with the OPC-UA communication during the project runtime, an OPC-UA trace log can be generated by the project runtime server.

The trace log complements the project runtime log displayed in the output window and the LogWin module.

To a large extent, the trace log contains the same information as the project runtime log (provided, the project runtime log is configured to contain OPC-UA messages). However, it is saved in an external file and can be configured to include even more detailed information.

☞ Make an entry in the WinStudio project configuration file to activate the trace log.

How to activate the trace log:

1. Open the project properties via the context menu of the IndraWorks project node.
   The temporary project directory is displayed in the Properties dialog.
2. Open the temporary project directory via the displayed links.
   The currently active project is displayed in the Windows Explorer.
3. One of the files is assigned the HMI node name in the IndraWorks project tree.
   Open the file.
4. Access the "Config" folder via the "User" folder.
5. Open the "HMI.app" configuration file with a text editor (e.g. Notepad++).
6. Add the following properties
   (For specifications to "UaLogPath" and "UaTraceLevel" see below):
   ```
   [OPC]
   UaLogPath=<file path and name>
   UaTraceLevel=<Level>
   ```
7. Save and close the configuration file.
8. To implement changes, close and restart the IndraWorks project.
   When starting the runtime the next time, the trace log is started and the log file is saved at the user-defined location.

☞ The log file size increases with the TraceLevel as long as runtime is running.

UaLogPath:

`<UaLogPath>` is the file path in the project folder.

If only a file name (e.g. opcualog.txt) is entered for `<UaLogPath>`, the trace log is saved in the project folder.

☞ The file path has to be compatible with the target device the runtime is running on.

UaTraceLevel:

Level	Description
0	**NoTrace** Trace log is deactivated Default setting if UaTraceLevel is not configured in the WinStudio configuration file
1	**Errors** Internal system errors which require troubleshooting

2	**Warnings** Internal system warnings and external errors Default setting if UaTraceLevel is configured in the WinStudio configuration file
3	**Info** Detailed information regarding system events
4	**InterfaceCall** Information required for debugging
5	**CtorDtor** Information required for debugging
6	**Program** All message contents
7	**FlowData** All messages

☞ The trace levels are cumulative, i.e. level 2 contains level 1, level 3 contains level 1 and 2 etc.

11.6 Configuring a TCP/IP Client Connection to Another Project

TCP/IP client task/worksheets are used to communicate with different "WinStudio" project servers.

TCP/IP client and server tasks enable two or more projects to keep their databases synchronized.

These tasks use the TCP/IP protocol to provide communication between projects.

Before using the TCP/IP client and server tasks, it has to be ensured that the TCP/IP (ethernet) communication is properly configured and running on both servers.

- **To configure the server, proceed as follows**: On the server itself, nothing has to be configured. Only the "WinStudio" TCP/IP Server task has to be executed. It can either be executed automatically on starting the runtime or manually in the task configuration (either local or by remote access). After running this program, a small icon is displayed in the system tray.
- To close the "WinStudio" TCP/IP Server task, proceed as follows: Right-click on the **TCP/IP Server** icon in the system tray and click **Exit** .
- **To configure the client, proceed as follows**: Use "TCP/IP worksheets" to specify the server IP address and the tags to be used with the server on the client system. These may be created in the following folder: **WinStudio ▸ Variables ▸ Communication ▸ TCP/IP**

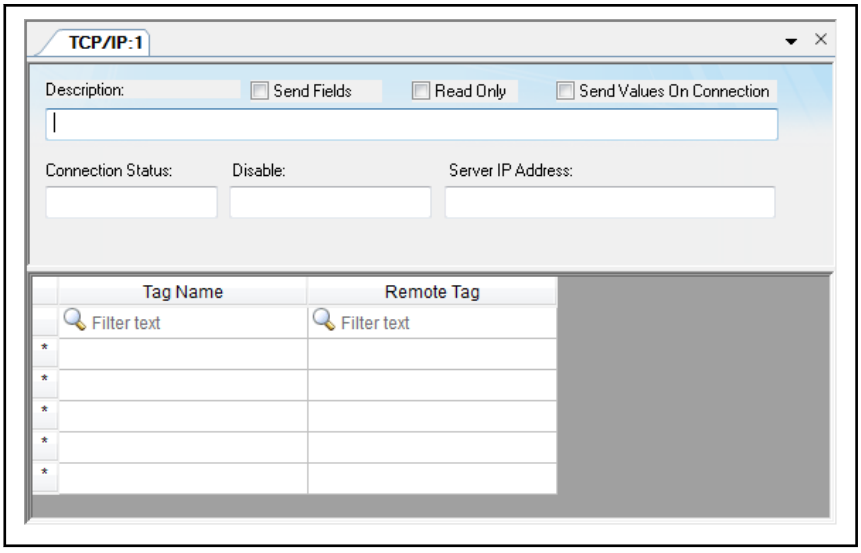


Fig. 11-22: TCP/IP driver sheet configuration

Use the following parameters to complete the TCP/IP client configuration:

- **Description**: Enter a description of the TCP/IP driver sheet for documentation purposes. The TCP/IP client task ignores this information.
- **Connection Status**: Enter a tag name to be updated according to its connection status by the TCP/IP client configuration task. A tag value of zero indicates that the connection is okay. Any other tag value indicates an error code returned by the "Windows Socket" library.
- **Disable**: Enter a tag name in this field. If this tag has any value other than 0, this TCP/IP worksheet is disabled. If this field is used, the TCP/IP client worksheet can be activated/deactivated during runtime.
- **Server IP Address**: Enter the IP address and Port (optional) of the target server, for example, "169.254.182.158:123". The Port is to be the same on both the client and server stations.

  String tags can be entered in curly brackets (e.g., `{tagname}`) to dynamically change this address during runtime.
- **Send Fields**:
  - If this checkbox is "disabled", TCP/IP client/server tasks "only" exchange tag values and their TimeStamp and Quality.
  - If this checkbox is "enabled", TCP/IP client/server tasks "also" exchange the Min, Max, Ack, Unit, LoLoLimit, LoLimit, HiLimit, HiHiLimit, RateLimit, DevSetPoint, DevpLimit, and DevenLimit tag fields.

☞ Additional fields can be added to the TCP/IP communication or fields can be disabled individually. For further information, contact the distributor.

- **Read Only**: If this option is enabled, the entire communication takes place in one direction and no tag values are returned to the specified server. This is useful if the TCP/IP client is only used to read data from other projects. Enabling this option may considerably improve safety and stability during runtime.
- **Tag Name**: Enter the tags to be shared with the server. If the tag is an array or class (or both), the project automatically activates all array and class elements for TCP/IP Communication by default. To configure a specific array and/or class element, enter the array index or class ele-

ment in square brackets followed by the tag name. For example,"level[3].member".

- **Remote Tag** ("optional"): Enter the name of a tag to be linked with the tag specified in the **Tag Name** field. If this field is left blank, the project uses the same tag name used in the client and in the server.

☞ If an array is to be shared, the tag in the server is to contain the same number of elements as the tag in the client. If the tag is a class, the class definition is to be the same in both server and client programs. If these rules are not adhered to, unexpected results can occur.

The TCP/IP client task can either be executed automatically on startup of the runtime environment or manually in the task configuration (**WinStudio ▸ Tools ▸ Status**). After starting the TCP/IP task, a small icon is displayed in the system tray.

The parameter `ConnectionRetryTimeout` is only used by the client.

11.7 Configuring a DDE Client Connection to a DDE Server

DDE worksheets are used to configure DDE client connections to DDE server applications such as Microsoft Excel (or any other Windows program supporting this interface).

Dynamic Data Exchange (DDE) is a protocol for dynamic data exchange between Windows applications, such as Excel. DDE communication is an interaction between server and client programs. "WinStudio" provides interfaces that run as clients or as servers Refer to DDE client runtime and DDE server in the Runtime tasks (refer to **Tools ▸ Status** in the context menu of the "WinStudio" folder.

- To run as a server, start the DDE or NetDDE server task as described in "Runtime tasks".
- To execute as client, the DDE driver sheet is configured in the **WinStudio ▸ Tags ▸ Communication ▸ DDE** folder.

Network Dynamic Data Exchange (NetDDE) is an extension of DDE that works across computers on a network.

- To run "WinStudio" as server for a NetDDE connection, start the DDE server application.
- To run "WinStudio" as a client for a NetDDE connection, use the same DDE driver sheets with the respective configuration to address a "WinStudio" project.

☞ When running NetDDE, "WinStudio" accepts only WRITE triggers. To read data, a write command has to be configured on the server computer.

To create a new "DDE" worksheet, select "New Driver Sheet" in the context menu of the "DDE" folder. Afterwards, double-click on the new element to open the driver sheet editor.

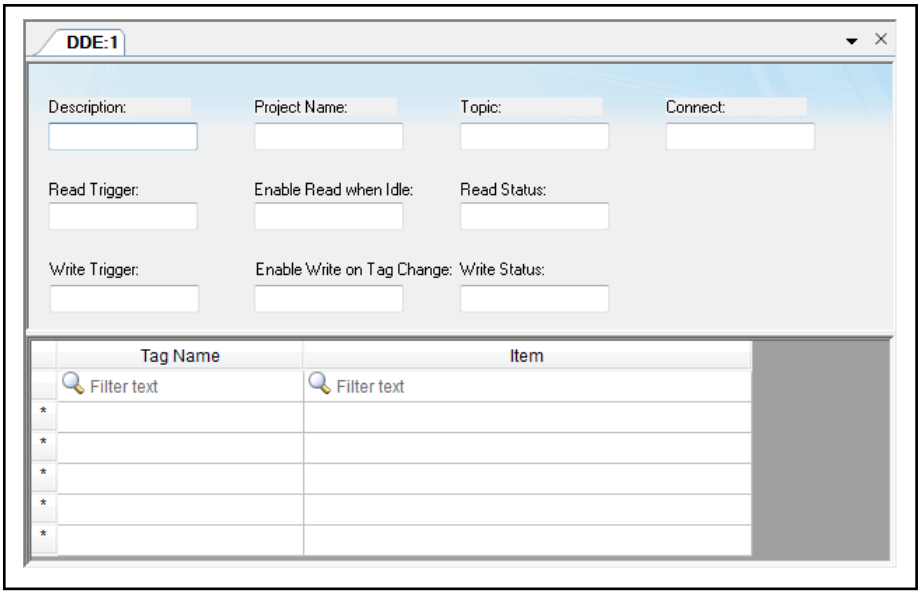


Fig. 11-23: DDE worksheet

The "DDE" worksheet dialog is divided into two areas:

- "Header" area (top section) containing information for the whole group and defining the tags to start the reading and writing process and to receive the connection status.
- "Body" area (bottom section), to define tags in the project and items related to the DDE server application.

Every DDE interface is based on addressing an application using the following three structures:

- Project name
- Topic
- Item

This information has to be determined before configuring the DDE driver sheet in the DDE server application.

Use the parameters in the DDE client worksheet header area as follows:

- **Description**: Enter a description of the DDE worksheet for documentation purposes.
- **Project Name**: Enter the DDE server application name.
- **Topic**: Specify a topic in the server application.
- **Connect**: Enter a tag to control the connection of the "WinStudio" DDE client and DDE server application. If this tag is set to 1, a connection to the server is requested. If the connection cannot be established or fails, "WinStudio" resets the tag to 0. If the connection is OK, this value remains set to 1.
- **Read Trigger**: Enter a tag as read command for the table. If this tag changes its value, "WinStudio" calls the variable values from the DDE server. This parameter can only be used with local DDE, not with NetDDE servers.
- **Enable Read when Idle**: Enter a tag value higher than zero to enable reading of the equipment.
- **Read Status**: Contains the status of the read command.
- **Write Trigger**: Enter a tag triggering the write command.

- **Enable Write on Tag Change**: Enter a tag value higher than zero to enable the communication driver to check continuously for changes in a tag value in the worksheet. If the driver detects a change, it writes the changed tag to the device, along with the tag address.
- **Write Status**: Contains the status of the writing command.

Use the DDE client body area parameters as follows:

- **Tag Name**: Enter a tag to read or write the "WinStudio" database from the DDE server application.
- **Item**: Enter the ITEM part of the DDE address on the server. Refer to the server software documentation for information about the proper syntax for **APP**, **TOPIC**, and **ITEM**.

  The **Topic** and **Item** fields can be configured with tags in the address according to the following syntax:

  `text{tag}`

  "WinStudio" evaluates the value of **{tag}** in a String and uses it in the address. For example:

  – `Topic: topic_{tag_topic_name}_example`
  – `Item: {tag_item_name}` or

    `A{tag_number}`

Configuring a NetDDE connection is similar to configuring a DDE connection, except for the "Name" and "Topic" in the header of the application. Before starting the tests, verify that the DDE server on the station with which data is to be exchanged is enabled.

☞ When connecting to servers other than "WinStudio", the proper syntax for **APP**, **TOPIC** and **ITEM** can be found in the server documentation.

To define the tags to initiate reading and writing as well as tags that receive the connection status, use the NetDDE client worksheet header parameters as follows:

- **Project Name**: Enter `computer_name\NDDE$`, where `computer_name` has to be a valid network computer name.
- **Topic**: Enter the topic name `UNISOFT$` to connect to another "WinStudio" station.

Use the DDE client worksheet body parameters to link each tag to each ITEM part of the DDE server address, as follows:

- **Tag Name**: Specify the "WinStudio" local database tag name that is related to a remote tag name.
- **Item**: Specify the remote tag name that is related to the local tag name.

☞ By default, the DDE client module supports DDE servers that handle String data in the UNICODE format. If the DDE server handles String data in the ASCII ANSI format, the following settings have to be configured manually in the `project_name.app` file (Notepad can be used to edit this file):

```
[Options]
DDEANSI=1
```

11.8 Sorting or filtering the lines of a worksheet

Sort or filter the lines of a worksheet to facilitate searching the lines or finding a particular element.

Before starting this task, insert a worksheet and open it for editing. In addition it should be clear how sorting and filtering works in general spreadsheet applications.

Please note that the lines can only by sorted and filtered in the following types of worksheets:

In Project tags, Shared tags and System tags data sheets;

In the Translation table worksheet;

In all Task worksheets except for report and script which do not have lines;

In all Communication worksheets.

The other worksheets do not have lines to sort or filter.

The worksheets are sorted alphanumerically by selected column either in ascending ((0-9, A-Z) or descending order (Z-A, 9-0).

	Tag Name	Type
	Filter text	...
1	Tag1	HiHi
2	Tag1	Hi
3	Tag1	Lo
4	Tag1	LoLo
5	Tag2	HiHi
6	Tag2	Hi
7	Tag2	Lo
8	Tag2	LoLo
9	Tag3	HiHi
10	Tag3	Hi
11	Tag3	Lo
12	Tag3	LoLo
*		HiHi

Fig. 11-24: Alarm worksheet lines in their original order

	Tag Name	Type
	Filter text	...
10	Tag3	Hi
2	Tag1	Hi
6	Tag2	Hi
9	Tag3	HiHi
5	Tag2	HiHi
1	Tag1	HiHi
3	Tag1	Lo
11	Tag3	Lo
7	Tag2	Lo
8	Tag2	LoLo
4	Tag1	LoLo
12	Tag3	LoLo
*		HiHi

Fig. 11-25: Alarm worksheet lines sorted by type

The worksheets are filtered according to the string entered in the selected column. Only lines that match the string will be displayed.

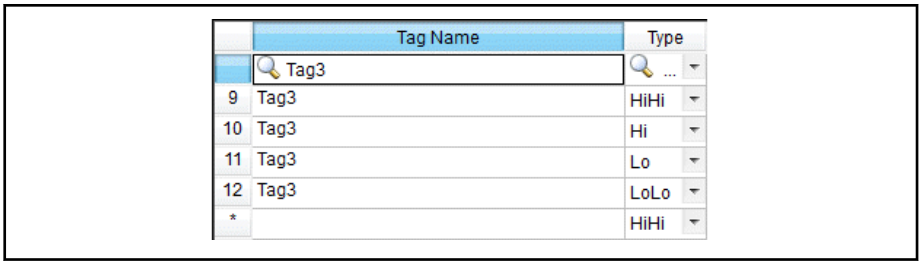


Fig. 11-26: Alarm worksheet lines filtered by tag name is "Tag3"

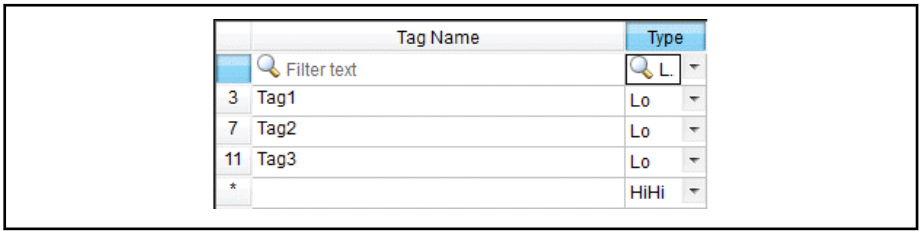


Fig. 11-27: Alarm worksheet lines filtered by type is "Lo"



It is still possible to delete lines while they are sorted or filtered.

Sorting or filtering lines:

1. To sort the lines, click on the column header by which you would like to sort. Click once to sort in ascending order and again to sort in descending order. The current order (i.e. the sorting direction) is indicated by the arrow to the right of the column name.



It is not possible to sort by several columns.

2. To cancel the sorting and restore the original order of the lines, click on the header of the first (numbered) column.
3. To filter the lines, enter the sting by which you would like to filter in the top line (zero) of the worksheet and press either **Tab** or **Enter**.

   *It is possible to use * and ? as wildcard characters in your string:*

   - * stands for any number of characters or no character. Tag* for example corresponds to `Tag`, `Tag3`, `Tag34567`, `TagA` and `Tag_TEMP`.
   - ? stands for exactly one character. `Tag?`, for example, corresponds to `Tag3` and `TagA`, while `Tag?????` corresponds to `Tag34567` and `Tag_TEMP`.

   It is possible to filter several columns. Only lines that match the filter string in all columns will be displayed.
4. To cancel filtering and restore the original order of the lines, delete the string you entered and either press **Tab** or **Enter**.

Please note that sorting or filtering the lines of a worksheet only facilitates the editing of this worksheet. How the worksheet is executed during runtime does not change. The lines are executed in their original order according to the numbering (column on the left), unless a line is moved or deleted.

12 Project security

IT security

To operate systems and machines, a holistic IT security concept, corresponding to state-of-the-art technology is required. Bosch Rexroth products are part of the holistic IT security concept. Bosch Rexroth product properties have to be taken into consideration in a holistic IT security concept. The properties to be taken into consideration are documented in the IT security guideline.

See Rexroth media directory:

DE: DOK-IWORKS-SECURITY***-PRRS-DE-P

EN: DOK-IWORKS-SECURITY***-PRRS-EN-P

Additionally to IT security measures, it is explicitly recommended to use the WinStudio project security system. The project security sytem manages how users and user groups can access projects during development and at runtime.

The WinStudio security system is only available if the **application type** "WinStudio application" is selected in the properties of the visualization device. For other application types, functions are protected during runtime by the IndraWorks function "Access protection".

Contains information about the configuration of the project security system, see chapter 12.7 "Security system dialog" on page 480

12.1 Security Modes

In addition to managing users and groups locally, entirely within a single project, pre-defined users and groups from other "WinStudio" projects or an LDAP-compliant domain server can also be applied.

"WinStudio" supports four security modes:

Local Only

This is the standard mode for most projects: Users and groups are created in the development environment and apply only to the project for which they are created.

Distributed – Server

This mode is similar to **Local Only**, except that the project's security system configuration is also made available to other "WinStudio" projects (that are set to **Distributed – Client**) on the same network.

Additionally, if the project loses its security system configuration for some reason, it can be reimported from one of its client projects.

Distributed – Client

If this mode is selected, the project receives its entire security system configuration from another "WinStudio" project (that is set to **Distributed – Server**) on the same network.

The project saves this configuration and can continue to run even if communication to the server project is lost.

Domain (LDAP)
The Lightweight Directory Access Protocol (LDAP) is a recognized standard for managing users and groups across many different applications on a network. If this mode is selected, the project receives its users and groups from an LDAP-compliant domain server, such as Microsoft Active Directory for Windows or OpenLDAP for Linux. However, only user names, passwords, and group memberships are taken from the domain. Specific rights still have to be configured within the project.

12.2 Security access levels

Almost all items of a project – screen objects, object animations, project screens, task worksheets – can be assigned a security access level. This access level determines the user groups that can edit the item during development and/or use it at runtime.

There are 255 possible access levels, allowing fine grading. Each user group is configured with ranges of levels for both development and runtime and the ranges of the group may overlap.

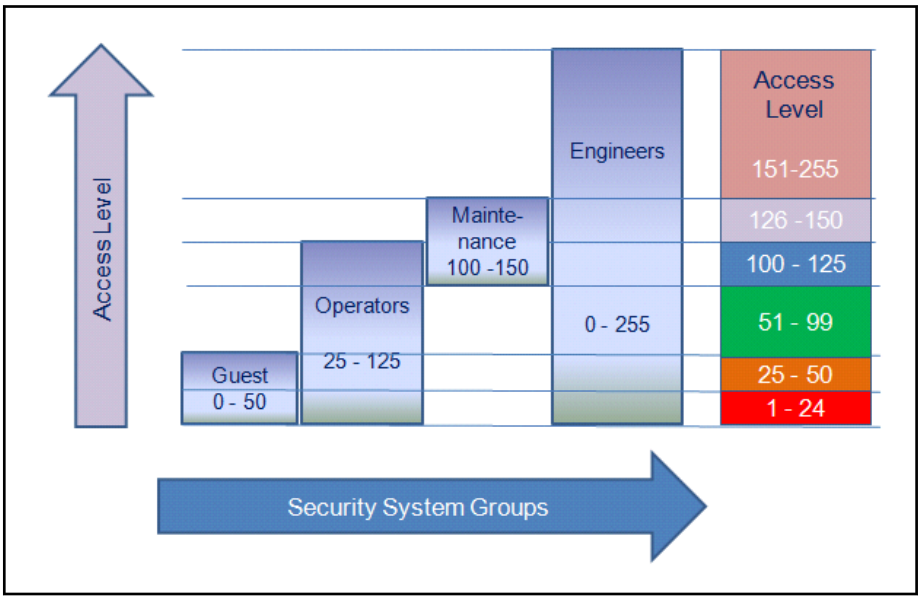


Fig. 12-1: Example of security access levels

This means that for a user to be able to edit and/or use an item, the item's access level has to fall within the range specified for that user's group.

Example: **UserA** of **GroupA** has a security access level range of "1-10" and **UserB** of **GroupB** has a security access level range of "5-15". Furthermore, the project contains the following elements:

- Item #1 has access level = 1
- Item # 2 has access level = 7
- Item # 3 has access level = 12
- Item # 4 has access level = 20

This results in the following access levels for the items:

- Only **UserA** can access Item #1
- Both users can access Item #2

- Only **UserB** can access Item #3
- Neither user can access Item #4



The default access level for all items is 0 and all users can use all items at that level.

Setting access levels for different types of items

The access levels for WinStudio screens can be set in the Screen attributes dialog on page 170. Security (access level) settings in the "Screen attributes" dialog

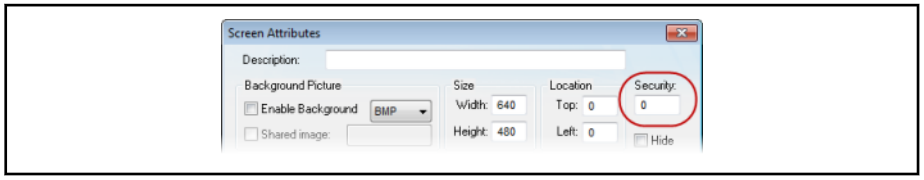


Fig. 12-2: Security (access level) settings in the “Screen attributes” dialog

For screen objects and object animations, the access level can be set in the individual Object properties dialog on page 178. (If the option is not available in the main dialog, it is generally available in one of the sub-dialogs.) Security (access level) settings in the "Object properties" dialog



Fig. 12-3: Security (access level) settings in the “Object properties” dialog

For task worksheets, do the following:

1. Open the worksheet for editing.
2. Click anywhere in the body of the worksheet. The item **WinStudio <Worksheet> ▸ Change access level** is enabled in the IndraWorks main menu.
3. Click on **Change security level**. The "Security" dialog is displayed.

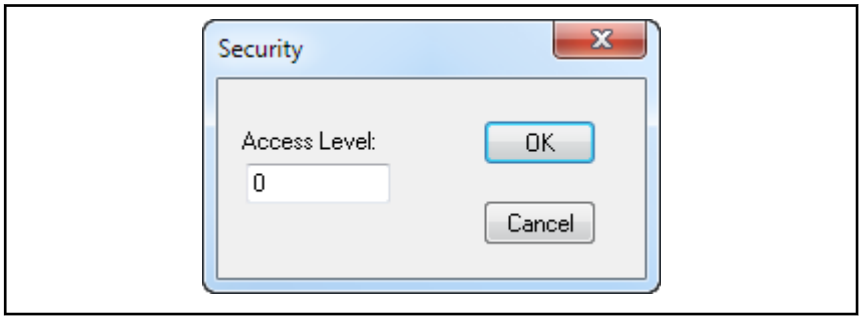


Fig. 12-4: Security dialog

4. In the **Access level** field, enter an access level for editing the worksheet.
5. Click on **OK**.

12.3 Using the security system configuration wizard

The security system configuration wizard guides through the steps of configuring the project security system. To start the configuration wizard, select **Tools ▸ Security settings** in the context menu of the **WinStudio** folder.

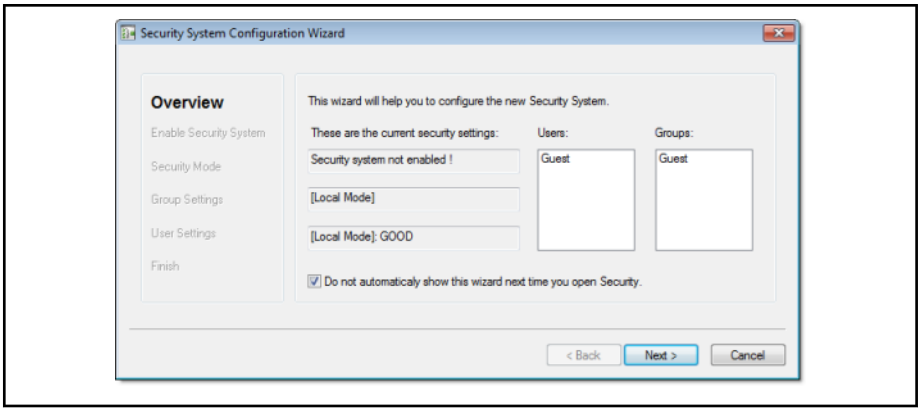


Fig. 12-5: Configuration wizard for the security system

1. The first page of the wizard is displayed. This page always shows how the security system is currently configured.
2. Click on **Next**. The second page of the wizard is displayed.

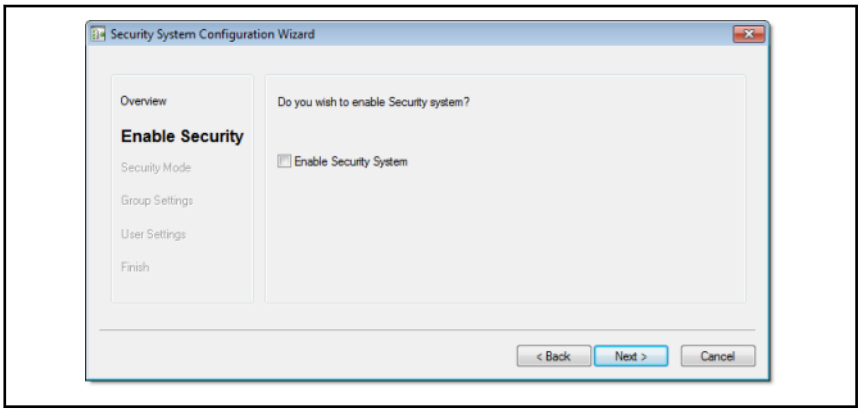


Fig. 12-6: Configuration wizard for the security system (2)

Select **Enable security system**.

3. Click on **Next**. The third page of the wizard is displayed.

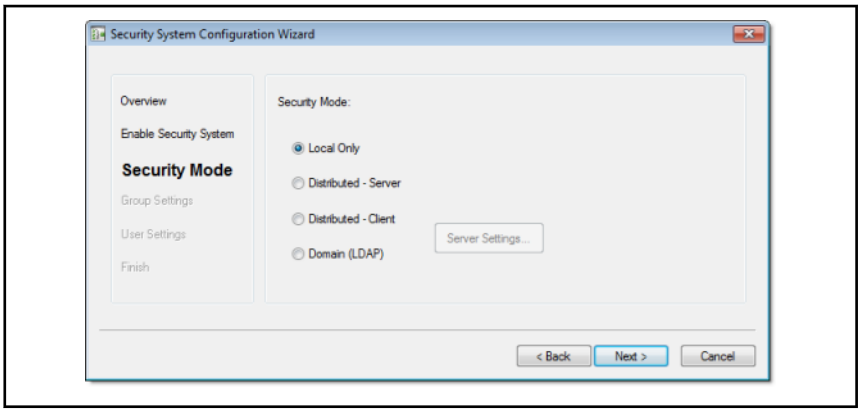


Fig. 12-7: Configuration wizard for the security system (3)

Select the Security mode on page 461.

For **Distributed – Client** and **Domain (LDAP)**, click **Server settings** and configure the Required settings on page 466.

4. Click on **Next**. The fourth page of the wizard is displayed.

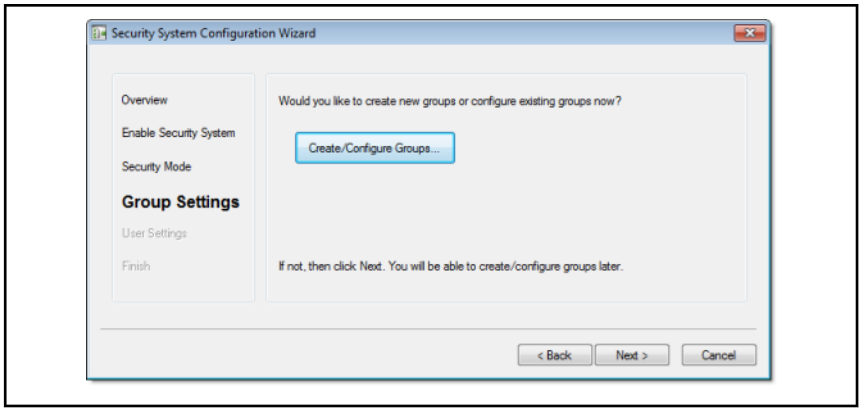


Fig. 12-8: Configuration wizard for the security system (4)

If groups have to be created or configured, click on **Create/Configure groups**. The Group account dialog on page 474 is displayed. If this dialog is closed, the system automatically returns to the wizard.

5. Click on **Next**. The fifth page of the wizard is displayed.

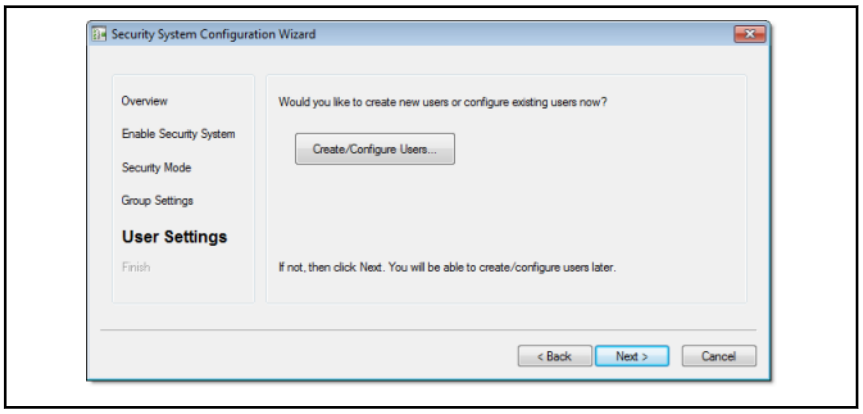


Fig. 12-9: Configuration wizard for the security system (5)

If users have to be created or configured, click on **Create/Configure Users**. The User account dialog on page 478 is displayed. If this dialog is closed, the system automatically returns to the wizard.

6. Click on **Next**. The sixth page of the wizard is displayed.

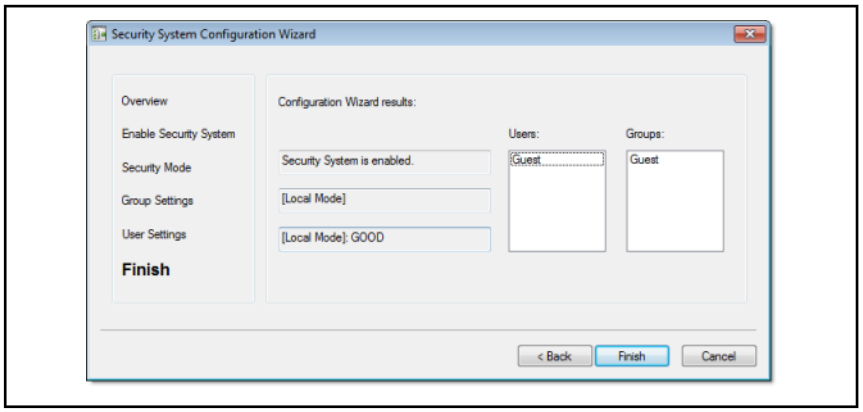


Fig. 12-10: Configuration wizard for the security system (6)

7. Review the configuration and click **Finish** to close the wizard.

12.4 Configuring server settings for security modes

12.4.1 Configuring server settings for security modes, general information

If the security mode of the project is either set to Distributed – Client or Domain (LDAP), the server settings have to be configured as well.

Click **Server settings** in the security system configuration wizard on page 463 or the security system main dialog on page 480 to open the respective "Server settings" dialog.

Configuring the server settings for the security mode: Distributed – Client

To configure the server settings in the “Server settings” dialog for the security mode. proceed as follows:

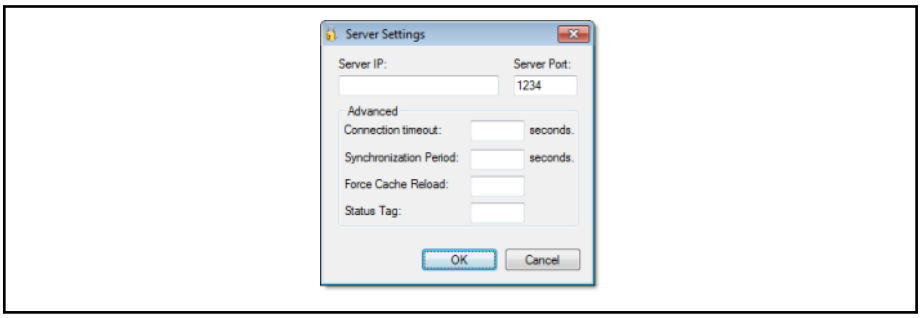


Fig. 12-11: Distributed – Client

1. In the **Server IP** and **Server Port** fields, enter the IP address of a runtime project that is set to **Distributed – Server**.
2. In the **Connection timeout** field, enter the timeout (in seconds) after which the client is to attempt to reconnect to the server (a typical connection timeout is 3 seconds).
3. In the **Synchronization period** field, enter the frequency (in seconds) at which the client is to synchronize its security system configuration with the server configuration. (A typical synchronization period is 10 seconds.)
4. In the **Force cache reload** field, enter the name of an application variable (Integer or Boolean type). If the tag value is TRUE (not zero) and the specified server has a timestamp older than the client, the local security system is updated with outdated server information.
5. In the **Status tag** field, enter the name of an application tag (Integer type) that is to receive a value indicating the status of the server connection. While the project is running, the following values are applicable: For further information, refer to GetSecuritySystemStatus on page 681:

Value	Description
0	No cache
1	Updated cache
2	Outdated local cache
3	Outdated server cache
4	Disconnected from server

6. Click on **OK**.

Configuring the server settings for the security mode: Domain (LDAP)

To configure the server settings in the “Server settings” dialog for the security mode. proceed as follows:

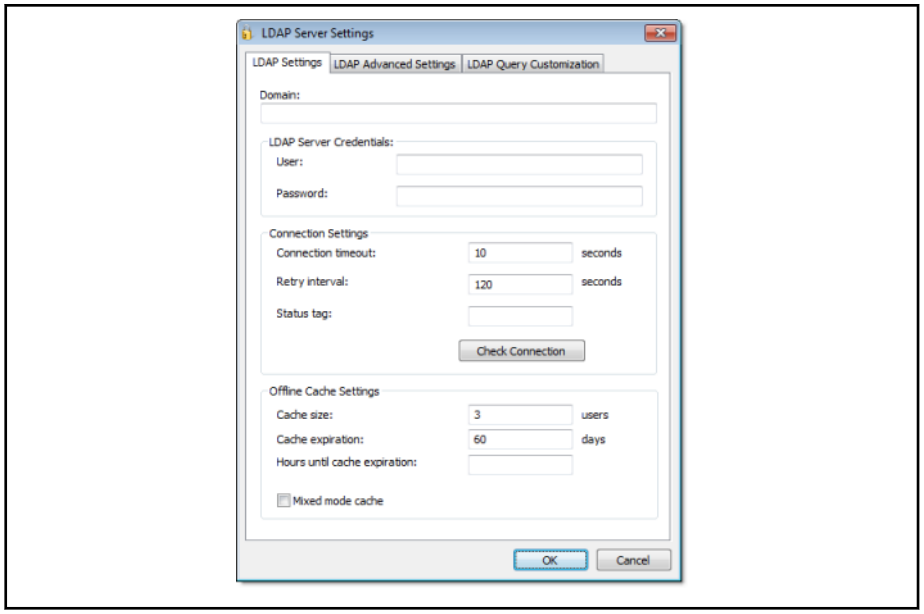


Fig. 12-12: Domain (LDAP)

1. In the **Domain** field, enter the domain name of the LDAP server.
2. In the **User** and **Password** fields, enter the access data for the LDAP server. To access group and user lists, sufficient user rights are required. Please contact the responsible LDAP administrator.
3. In the **Connection timeout** field, enter the timeout (in seconds) after which the client is to attempt to reconnect to the LDAP server (a typical connection timeout is 4-5 seconds).
4. In the **Retry interval** field, enter the frequency (in seconds) at which the client is to try to connect to the LDAP server if the connection could not be established.
5. In the **Status tag** field, enter the name of an application tag (Integer type) that is to receive a value indicating the status of the server connection. While the project is running, the following values are applicable:

Value	Description
0	Connection timeout
1	Binding timeout
2	Request timeout
3	Disconnected
4	Connected
5	Query returned no users or groups
6	Invalid user or group

For further information, refer to GetSecuritySystemStatus on page 681:

6. Click **Check connection** to confirm that the project can connect to the specified domain. If this is not possible, review and correct the settings.
7. Click on **OK**.

The project security system caches the most recent users in case the project loses its connection to the LDAP server. These users are still able to log onto the project. The cache size can be unlimited by setting **Cache size** to "0" and never expires by setting **Cache expiration** to "0".

The users are buffered in a FIFO list (first in, first out). Thus, if the latest user is included in the buffer, the oldest user is removed. The default user is an exception. It always remains in the buffer.

To monitor the cache at project runtime, enter the name of a project tag in the **Hours until cache expiration** field. This tag receives a value equal to the number of hours until the cache expires.

If users are to be enabled to log on with either local or server-defined accounts, select **Mixed mode cache**. Local users and groups can only be created if the **Security mode** is set to **Local only**. After creation, they can be kept in the cache if the **Security mode** is set to **Domain (LDAP)**.

If required, select that trace messages (in the output window) are reported each time the buffer is refreshed. Change your project file `projectname.APP`) manually or add the following setting:

```
[Options]
EnableSecuritySystemTraceLdapOfflineCache=1
```

In the "LDAP advanced settings" tab of the dialog, there are additional settings that are to be configured only by experienced LDAP administrators:

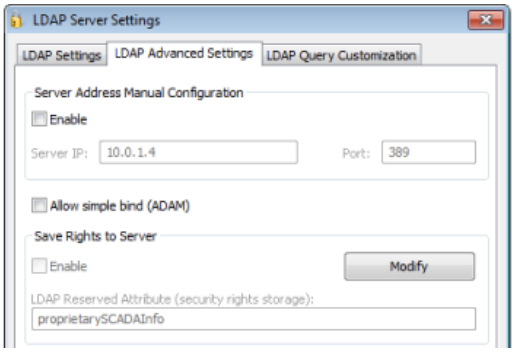


Fig. 12-13: LDAP advanced settings

Server address manual configuration — If for some reason the LDAP server cannot be accessed using its domain name, the server IP address can be manually configured: Select **Enable** and enter the IP address and port number of the LDAP server.

The default port for LDAP is 389. However, please verify the port number with the server administrator.

Allow simple bind (ADAM) — LDAP normally requires secure binding for authentication, but active directory application mode (ADAM) in Windows Server 2003 does not fully support secure binding. To allow simple binding with an ADAM server, select this option.

Simple binding means that user credentials are sent in plain text. Thus, secure the connection by other means such as VPN, TLS/SSL or proxies.

Save rights to server

By default, "WinStudio" security rights are saved entirely within the project. However, those rights can be saved back to the LDAP server, either to make them available to other projects that use the same LDAP server or for simple redundancy.

To make this option work, the server LDAP schema is to be extended by additional information on the project security system. Refer to Extending the LDAP schema to allow saving of security rights on page 469.

Afterwards, click on **Modify** to provide the LDAP server credentials and select **Enable**.

In the "LDAP query customization" tab of the dialog, the formatting of LDAP server queries can be further customized:

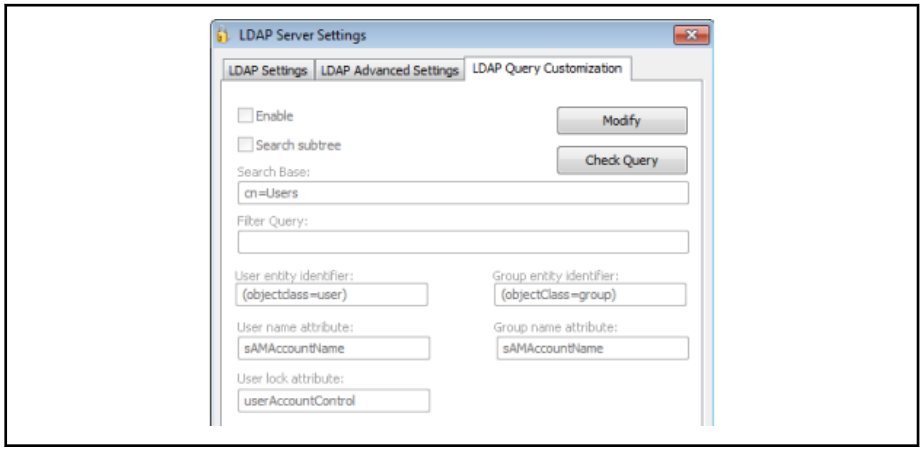


Fig. 12-14: LDAP query customization

By default, the LDAP server provides a list of "all" registered users and groups. In a large or complex network environment, this may lead to an impractically long list for project security system configuration. To restrict the list of users and groups, the LDAP query can be customized to eliminate any users and groups that never are to access the project: Click on **Modify** to enter the LDAP server credentials, select **Enable** and then configure the **Search base** and **Filter query** settings. For the proper syntax, consult the LDAP server documentation.

Also, some non-standard LDAP implementations – such as Linux-based LDAP servers and Active Directory Application Mode (ADAM) in Windows Server 2003 – use different entity identifiers and attributes. These can be customized in this dialog, however, only by experienced LDAP administrators.

Example of alternative attributes in ADAM

LDAP server	User name attribute	Group name attribute	User lock attribute
Active Directory	sAMAccountName	sAMAccountName	userAccountControl
Active Directory Application Mode (ADAM)	Name	Name	userAccountControl

12.4.2 Extending the LDAP Schema to Allow Saving of Security Rights

In order to save "WinStudio" project security rights back to a Domain (LDAP) server, the server LDAP scheme has to be extended for these additional information.

The server has to already be configured and running on the network, and sufficient access rights are required to change to the server configuration.

In this procedure, new attributes with the designation "proprietarySCADAInfo" are created to include the "WinStudio" project security rights. Afterwards, the attributes are assigned to the classes "user" and "group" in the server configuration. These classes correspond to users and groups in the project security system.

Please note that this procedure only illustrates how the scheme is to be extended in Microsoft Active Directory under Windows Server 2003. For different LDAP servers and operating systems, a different procedure applies, however, the basic steps are generally identical. Please refer to the LDAP server documentation.

NOTICE **Extending the LDAP scheme of a server cannot be undone.**

1. Register the schema management DLL.
   1.1 Click on **Start** ▸ **All Programs** ▸ **Accessories** ▸ **Command Prompt**.
   A command prompt window is displayed.
   1.2 In the prompt, enter `cd %SystemRoot%\System32` and press Return.
   The working directory is changed.
   1.3 Enter `regsvr32 schmmgmt.dll` and press Return.
   If the DLL is successfully registered, a confirmation message is displayed.

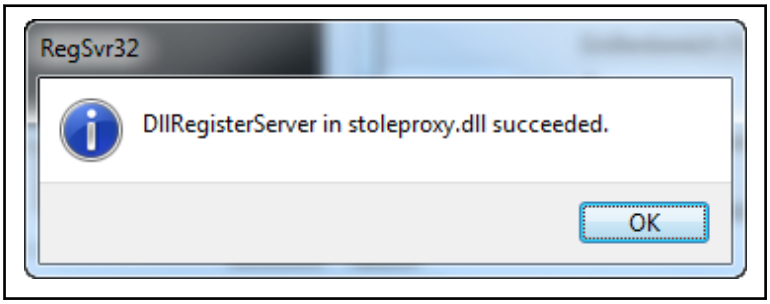


Fig. 12-15: Confirmation message

   1.4 Click OK to close the message.
   1.5 Close the command prompt window.
2. Add the Active Directory Schema snap-in to the console root.
   2.1 Click on **Start** ▸ **All Programs** ▸ **Accessories** ▸ **Run**. A Run dialog is displayed.
   2.2 In the **Open** field, enter "mmc" and click on **OK**.

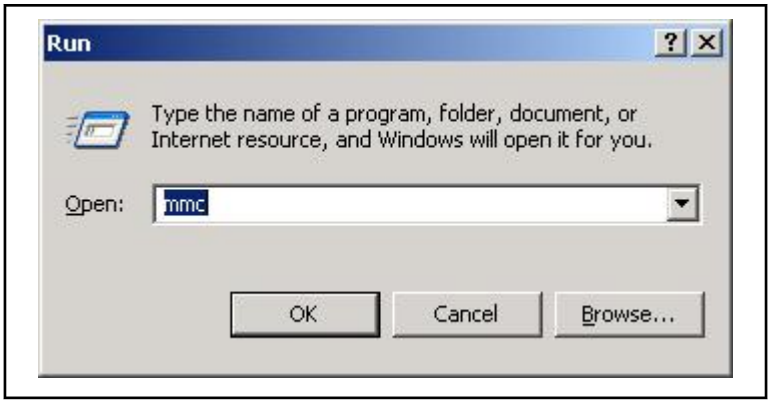


Fig. 12-16: Open field

(If User Access Control (UAC) is enabled, carrying out changes to the Microsoft Management Console is to be confirmed. Click on Yes.) A console window is displayed.

2.3 In the console window, click on **File ▸ Add/Remove Snap-in**.

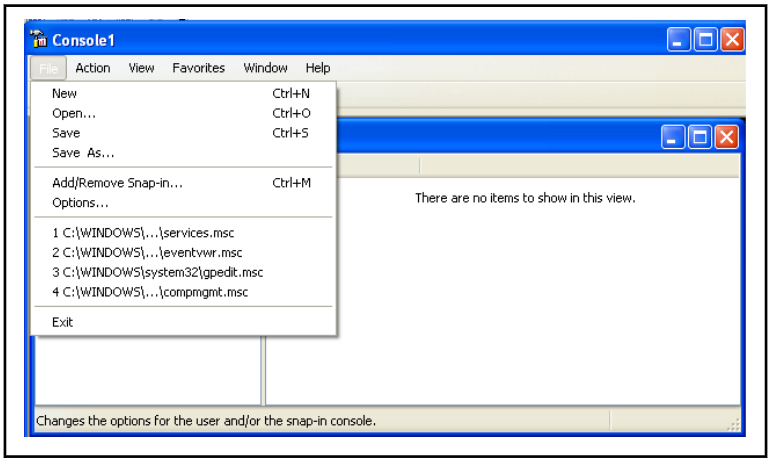


Fig. 12-17: Menu: Add/Remove Snap-in

The Add/Remove Snap-in dialog is displayed.

2.4 In the **Snap-ins added to** list, select **Console Root** and click on **Add**. The "Add Standalone Snap-in" dialog is displayed.

2.5 In the list of available snap-ins, select **Active Directory Schema** and click on **Add**.

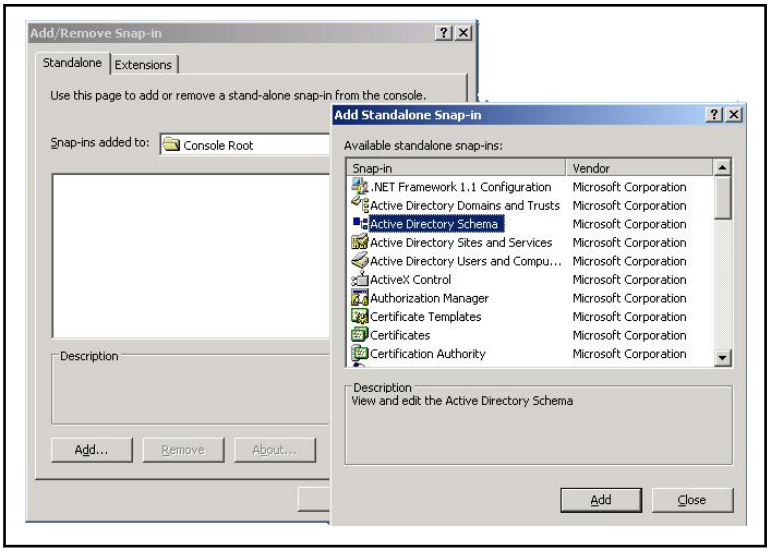


Fig. 12-18: Dialog Add Standalone Snap-in

The snap-in is added to the Console Root.

2.6 Click **OK** to close the Add/Remove Snap-in dialog.

3. Create the proprietary SCADAInfo attribute in the Active Directory Schema snap-in The proprietary SCADAInfo attribute is added to the list.

3.1 In the **Console Root** tree-view, expand **Active Directory Schema**.

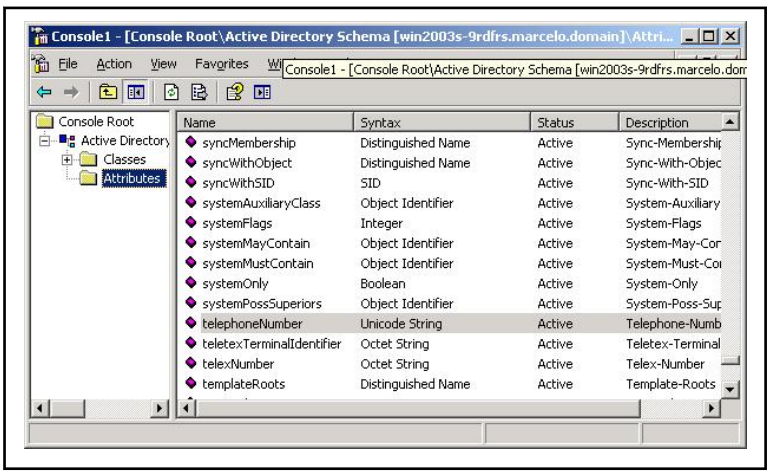


Fig. 12-19: Active Directory Schema

3.2 Right-click on Active Directory **Schema** ▸ **Attributes** and click on **New Attribute** in the context menu. A message is displayed explaining that the schema is permanently changed.

3.3 Click on Continue. A Create New Attribute dialog is displayed.

3.4 In the dialog, complete the fields as follows.

- **Full Name**: `proprietarySCADAInfo`
- **LDAP Display Name**: `proprietarySCADAInfo`
- **Unique X500 Object ID**: `0.7.7777.77777777.777.7.7`



A unique Object ID is to be used.

Description: `proprietarySCADAInfo`

Syntax: `Octet String`

Minimum: `0`

Maximum: `10240`

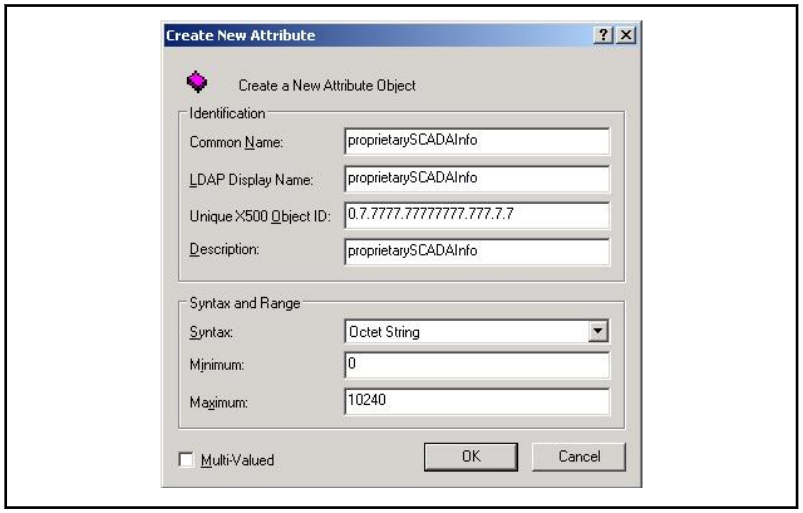


Fig. 12-20: Dialog: Create New Attribute

3.5 Click on **OK** to close the dialog.

The proprietarySCADAInfo attribute is added to the list.

4. Add the proprietarySCADAInfo attribute to the **person** and **group** classes.

4.1 In the **Console Root** tree-view, select Active Directory. **Schema ▸ Classes**.

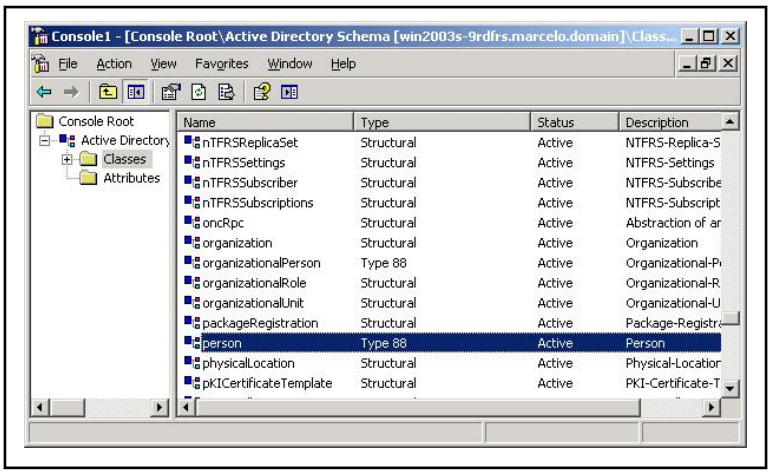


Fig. 12-21: Dialog Active Directory Schema > Classes

4.2 In the list of classes, right-click on **person** and click on **Properties** in the context menu.

The Properties dialog is displayed.

4.3 In the dialog, click on the **Attributes** tab.

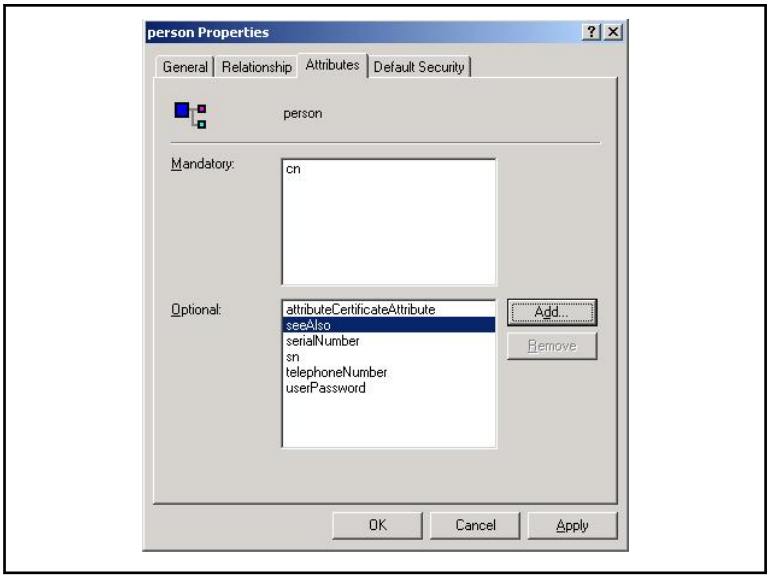


Fig. 12-22: Dialog person Properties, Attributes

4.4 Click on **Add**. The Select Schema Object dialog is displayed.

4.5 In the list of schema objects, select **proprietarySCADAInfo** and click on **OK**.

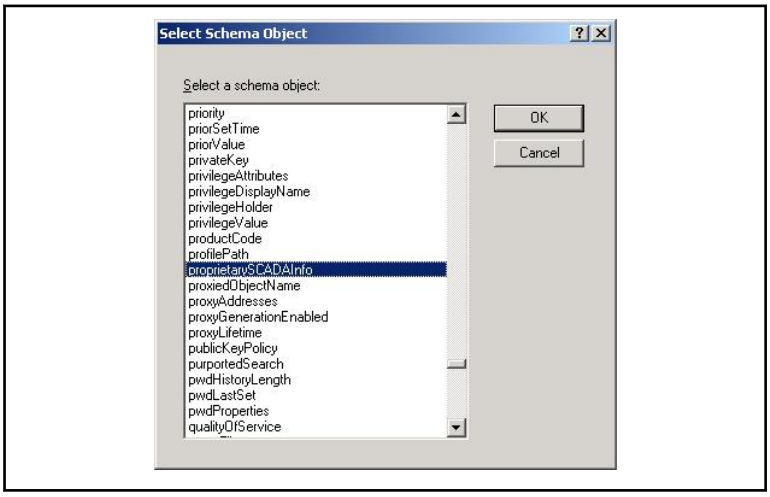


Fig. 12-23: Select Schema Object dialog

The attribute is added to the class properties.

4.6 Click on **OK** to close the Properties dialog.

4.7 Repeat steps b through f for the **group** class.

5. In the **Console Root** tree-view, right-click on **Active Directory Schema** and select **Refresh** in the context menu.
6. Click on **File** ▸ **Close**to close the console window.
7. Restart the server.

12.5 Creating and configuring groups

Use the "Group Account" dialog to create and delete user groups and to configure the access rights of a selected group.

Accessing the dialog

Assuming the project security system has already been enabled (i.e., the security system configuration wizard has been used at least once), this dialog can be accessed as follows:

- Open the Security System dialog on page 480 and click on **Groups**; or
- In the WinStudio context menu, select the entry **Tools** ▸ **Add group** and click on **Groups**

The dialog in detail

Note that if a user is assigned to more than one group (see Creating and configuring users page 478), the group settings may be in conflict with each other. How to undo the settings, depends on the respective setting:

- The settings in the following “Group account” are tolerant — that is, the most tolerant setting from all users groups applies to the user. For example, if any of the groups can create and modify tags, then the user can create and modify tags.
- The settings in the “Group account advanced” dialog (both tabs) are restrictive — that is, the most restrictive setting from all users groups applies to the user. For example, if one group has a minimum password size of 8 and another group has a minimum password size of 12, the minimum password size of the user is 12. (For auto log off in particular, **Counting from logon** overrides **Counting from user's last action**.)

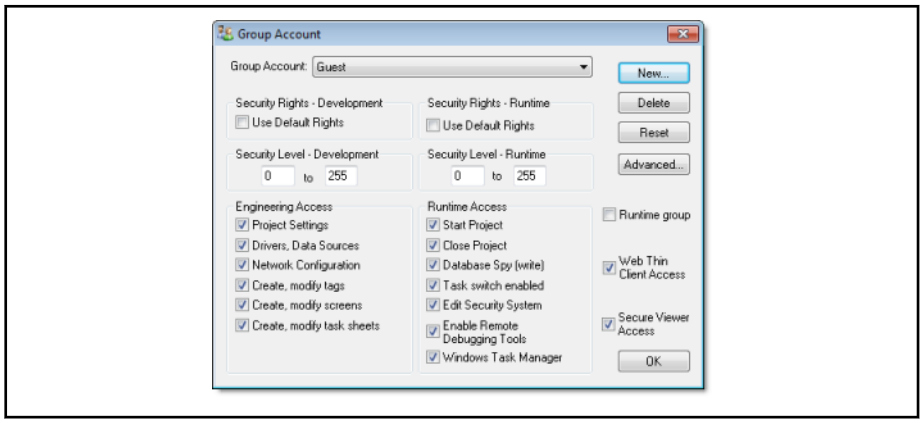


Fig. 12-24: Group account dialog

Area/element		Description
Group account		The user group that is currently configured. **Note**: There are two default groups for all projects: **Guest** and **(Default rights)**. **Note**: If the security mode is set to **Domain (LDAP)**, please note that the built-in groups in Microsoft Active Directory are not displayed in this list of groups and cannot be added to the project.
New		Creates a new group. In the "New group account" dialog, enter the name of the new group and then click **OK**.
Delete		Deletes the currently selected group.
Reset		Resets the privileges of the currently selected group to match the **(default rights)** group. This does not lock the group to the default; further changes can be carried out. To lock the group, see **Use Default Rights** below.
Advanced settings		Opens the "Group account advanced" dialog (see below).
Security rights – Development	**Use default rights**	Locks the development privileges of the currently selected group to those configured for the **(default rights)** group. If changes are made to the **(default rights)** group, then they also apply to this group.

Area/element		Description
Security level – Development	**Range**	The range of access levels on page 462 that this group can access in the development application.
Engineering access	**Project settings**	Members of the group may modify the Project settings on page 53.
	Drivers, data sources	Members of the group may create and modify device drivers and external data sources on page 413.
	Network configuration	Members of the group may create and modify TCP/IP client worksheets on page 453.
	Create, modify tags	Members of the group may create and modify project tags on page 104.
	Create, modify screens	Members of the group may create and modify project screens on page 169.
	Create, modify task sheets	Members of the group may create and modify task worksheets on page 391.
Security rights – Runtime	**Use default rights**	Locks the runtime privileges of the currently selected group to those configured for the **(Default Rights)** group. If changes are made to the **(default rights)** group, then they also apply to this group.
Security level – Runtime	**Range**	The range of Access levels on page 462 that this group can access in the runtime project.
Runtime access	**Start project**	Members of the group may run the project.
	Close project	Members of the group may stop the project.
	Database spy (write)	Members of the group may write values to the project database using the Database spy window on page 78. **Note:** This only applies to projects running locally. For projects running remotely, see **Enable remote debugging tools** below.
	Task switch enabled	Members of the group may switch away from the runtime project to another Windows task. (This option does not apply to projects running on Windows Embedded target systems; the user may always switch away from the runtime project.)
	Edit Security system	Members of the group may edit the project security system at runtime. **Note**: This option may not be deactivated for the own group as, in this case, own changes cannot be undone.
	Enable remote debugging tools	Members of the group may use the remote database spy on page 502 and the remote LogWin on page 503.
	Windows Task Manager	Members of the group may open the Windows Task Manager. (This option does not apply to projects running on Windows Embedded target systems; the user may always open the Windows Task Manager.) **Note**: Clearing this option means disabling the task manager at runtime, which requires administrator rights. The project has to be executed with higher rights.
Runtime group		A user created at runtime by the function CreateUser on page 678 can be assigned to this group.
Web thin client access		Members of the group may access the runtime project by using a web thin client.
Secure viewer access		Members of the group may access the runtime project by using a secure viewer.

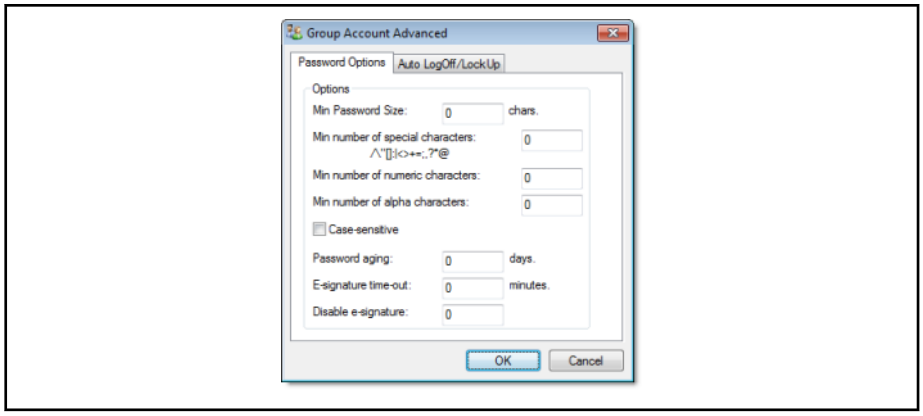


Fig. 12-25: Advanced dialog – Password options

Area/element	Description
Min password size	To make user passwords more complex and therefore more secure, passwords can be required to contain a certain number of alpha (A-Z, a-z), numeric (0-9) and special (punctuation) characters. If users are prompted to change the password, for example, if the old password expires (refer to **Password aging** below), the new password is not accepted unless it meets these requirements.
Min number of special characters	
Min number of numeric characters	
Min number of alpha characters	
Case-sensitive	If this option is enabled, passwords are case sensitive, i.e., passwords created with both upper and lowercase characters "have to" be entered the same way by the user. **Note**: In projects created with "Rexroth WinStudio v6.5" to "v7.2+SP1", all passwords are case sensitive.

Password aging	Validity (in days) of the password for all users in this group. After the specified number of days, the project forces users to change their password: If users try to log in, the "Change password" dialog is automatically displayed and the logon process cannot be completed before a new password is specified. By default, users have to choose a new password that is different from the old password. To change this and to enable users to re-use the same password, manually edit the project file (**project_name.app**) to include the following setting:
	`[Security]`
	`ChangePasswordMode=1`
	To make passwords never expire, set **Password aging** to "0".

E-signature time-out	Timeout period (in minutes) of the E-sign prompt for all users in this group. Users have to enter their user name and password before the specified timeout to use project functions that require an e-signature.
Disablee-signature	If the value in this box is TRUE (not zero), users in this group cannot use any project functions that require an e-signature. In this field, project tags can be configured to dynamically enable/disable the e-signature at runtime.

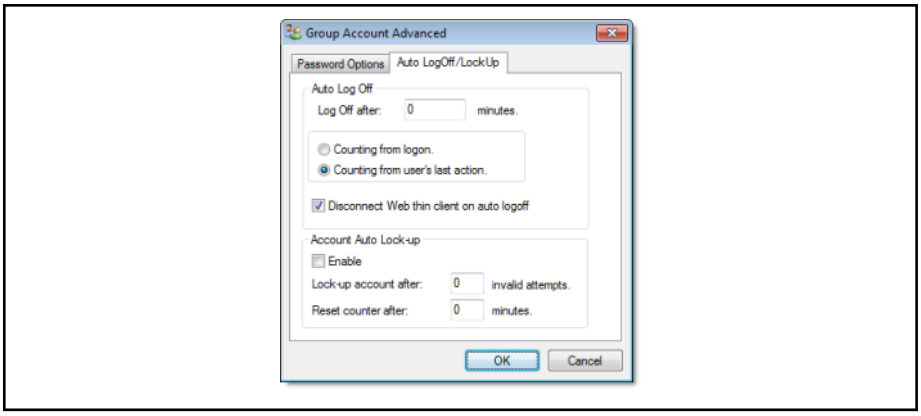


Fig. 12-26: Advanced dialog – Auto LogOff/LockUp

Area/element		Description
Auto log off	Log off after	Number of minutes after which the current user has to be logged off automatically. If this field is left in blank (or with the value 0), the current user is never logged off automatically.
	Counting from logon	If this option is selected, the current user is automatically logged off after the period of time configured in the **Log Off after** field elapsed since the current user logged onto the system.
	Counting from user's last action	If this option is enabled, the current user is automatically logged off after the period of time configured in the **Log Off after** field elapsed since the last action (mouse, touch screen or keyboard action) was performed by the current user.
	Disconnect web thin client on auto log off	If users logged on via a web thin client are logged off, the client is automatically disconnected from the data server.
Account auto lock-up	Enable	Enables the auto lock-up functions described below.
	Lock up account after	Maximum number of times users can try to log on to an account. If a user exceeds the specified maximum number of attempts (provides an invalid password) within the period of time specified in the **Reset counter after** field, the project automatically blocks the respective user.
	Reset counter after	Defines how long the project waits after an invalid log-on attempt (in minutes) before resetting the log-on attempts counter.

☞ If the VBScript troubleshooting is enabled, the “Automatic logoff” function cannot be used. The normal execution cycle is paused while searching for the error. Thus, it is not possible to measure the time passed without user input. For further information, refer to Troubleshooting VBScript, page 966

12.6 Creating and configuring users

To create and maintain accounts for project users, click the **Users** button on the "Security system" dialog.

The "User account" dialog is displayed.

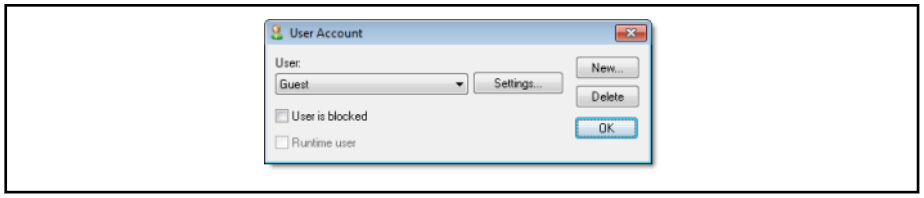


Fig. 12-27: User account dialog

If no users log on after project initialization (or if the current user logs off), the project automatically logs on the default user (**Guest**). In addition to the default **Guest** user, there is a **Guest** group, which has default privileges that enable all tasks. It is recommended to evaluate and edit the **Guest** group access rights to specify minimum rights for the start up procedure.

To create a new user, click **New** to open the "New user account" dialog.

To delete a user, click the **User** combo box button, select the user name from the list and click **Delete**.

To configure a user, proceed as follows:

1. Click on the **User** combo box button and select a user from the list.
2. If necessary, click on the **User is blocked** checkbox to unblock the selected user.

☞ The **Runtime user** option indicates that a user was created at using the function runtime using the function CreateUser on page 678.

This option cannot be selected for users created using the procedure described here.

---

3. Click on the **Settings** button to open the "User settings" dialog:

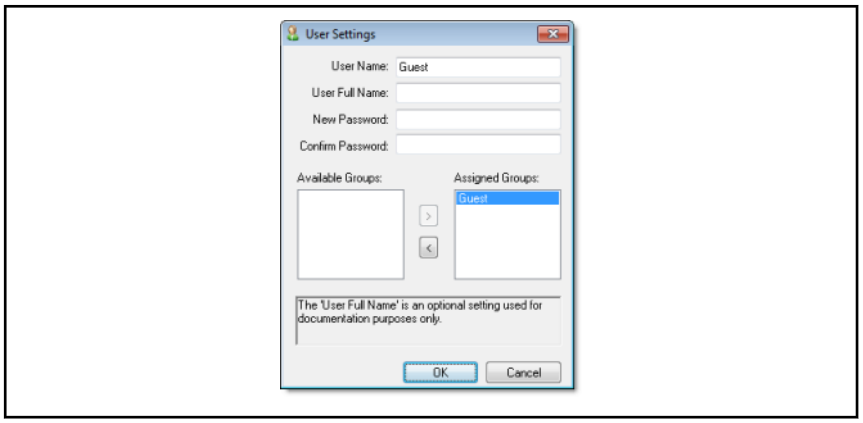


Fig. 12-28: "User settings" dialog

4. To configure the parameters on this dialog, proceed as follows:
   - **User full name** text field ("optional"): Enter the full user name.
   - **New password** text field: Enter the user password.
   - **Confirm password** text field: Re-enter the user password.
5. In the **Available groups** list, select the group(s) to which the user is to be assigned and click **>** to move these group(s) to the **Assigned groups** list.
6. To finish, click on **OK** to apply the changes and close the "Settings" dialog.

12.7 Security system dialog

The main "Security system" dialog is used to manage the project security system after it has been initially configured.

IT security

Within the framework of IT security, we strongly recommend to comply with the following principles:

1. Create a safe main password.
2. Create a safe password for the initial default account.

   By default, the initial account is named "Guest".
3. Create safe passwords for all other accounts.

Accessing the dialog

Assuming the project security system has already been enabled (i.e., the security system configuration wizard was used at least once), this dialog can be accessed as follows:

In the WinStudio context menu, select **Tools ▸ Security settings**.

If one of these actions is carried out "before" the security system has been enabled, the Security system configuration wizard on page 463 opens automatically.

If the security system has already been configured and a main password has been set, the password is requested.

The dialog in detail

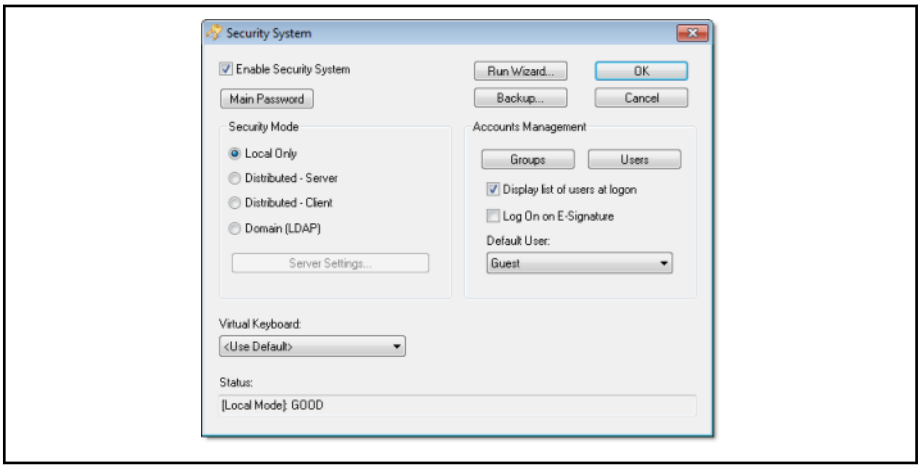


Fig. 12-29: "Security system" dialog

Area/element		Description
Enable security system		Indicates whether the project security system is currently enabled. If it is, the specific access rights of users and groups are enforced.
Main password		Opens a dialog to specify a main administrative password for the entire project.
Security mode	Mode	The current security mode on page 461 of the project.
	Server settings	Opens the Server Settings dialog on page 466 to configure the server settings for **Distributed – Client** or **Domain (LDAP)**.
Run wizard		Opens the security system configuration wizard on page 463.
Backup		Opens the Import/Export dialog on page 481 to export or import the security system configuration.

Area/element		Description
Account management	Groups	Opens the Group account dialog on page 474 to create and configure groups.
	User	Opens the User account dialog on page 478 to create and configure users.
	Display list of users at logon	Displays a list of available users (in the Log on dialog on page 482) if a user is prompted to log on. Users can select from this list instead of entering the user name.
	Log on E-signature	Forces users to log on with their own user account when prompted to e-sign an event. If this option is not enabled, the current user account remains logged on regardless of who e-signs the event.
	Default user	This user is automatically logged on when no other user is logged on, e.g., if the previous user times out or logs off manually. **Note**: This user's privileges should be heavily restricted, to prevent your project from being left vulnerable.
Virtual keyboard		The type of virtual keyboard displayed on the client if the user is prompted to log on.

12.8 Backing up the security system configuration

The security system configuration of the project can be backed up by exporting it to a file. Configurations can also be imported from a file or from another runtime project.

Exporting the configuration to a file

To export the security system configuration, proceed as follows:

1. In the main dialog Security System dialog on page 480, click on **Backup**. The "Import/Export" dialog is displayed.
2. Click on **Export to file**. A standard "Save as" dialog is displayed.
3. Specify a file name and location for the file and click on **OK**.

The exported file is encrypted with the main password configured in the "Security system" dialog.



Use the function ExportSecuritySystem on page 680 to also export the configuration at runtime.

Importing the configuration from a file

If the security mode on page 461 is set to **Local only**, a configuration can be imported from a previously exported file.

To import the security system configuration, proceed as follows:

1. In the main dialog Security system dialog on page 480, click on **Backup**. The "Import/Export" dialog is displayed.
2. Click **Import from file**. A standard "Open" dialog is displayed.
3. Locate the configuration file (***.dat**) to be imported and click on **OK**. The main password is requested for configuration.
4. Enter the password and click on **OK**. The "Import from File" dialog is displayed.
5. Select an import method:
   - **Import only settings that do not conflict**: Merge the imported settings with the current project settings. In the event of a conflict, the current settings are kept.

- **Import all settings and replace conflicts**: Merge the imported settings with the project settings. In the case of conflicts, use the imported settings.
- **Replace the current settings**: Completely replace the current project settings with the settings imported from the file.

6. Click on **OK**.



Use the function ImportSecuritySystem on page 685 to also import the configuration at runtime.

Importing the configuration from another project

If the security mode on page 461 of the project is set to **Distributed – Server**, a configuration can be imported from a different project if:

- The security mode of the other project is set to **Distributed – Client** and its server settings are configured to use the project as server; and
- The other project is currently running on the same network.

To import the security system configuration, proceed as follows:

1. In the main dialog Security system dialog on page 480, click on **Backup**. The "Import/Export" dialog is displayed.
2. Click on **Import from client station**. The **Import security from client station** dialog is displayed. The dialog shows a list of runtime projects that are using the project as security system server. Each project/client list includes a time stamp that shows when it last cached the security system configuration.
3. Select a client station and click on **Import from client**. The main password is requested for configuration.
4. Enter the password and click on **OK**.

12.9 Logging On/Off

If the project security system has been enabled and the default "Guest" user rights have been restricted, logging on is required to fully use the development application and/or the runtime project.



The project security system has to be enabled before this function is available.

To log on to the development application, in the WinStudio context menu, select **Tools ▸ User log on/log off...** .

To prompt a user to log on to the runtime project, proceed as follows:

- Open LogOn on page 702 where an expression can be configured – for example, draw a button object on page 195 in a screen and apply the command animation on page 246, so that pressing the button shows a logon prompt; or
- select the **Log on on E-signature** option (in the main dialog Security system dialog on page 480) to force the user to log on upon each action that requires an e-signature.

In either the development application or the runtime project, the “Log on” dialog is displayed:

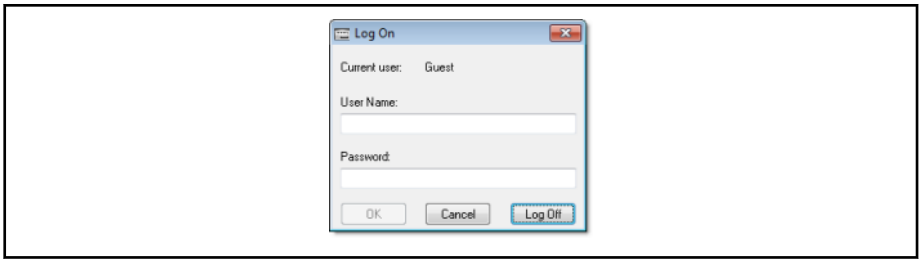


Fig. 12-30: "Log on" dialog

Use this dialog as follows:

- To log on with an own account, enter the user name and password in the respective fields and click on **OK**.
- To log on as the default "Guest" user, enter "Guest" in the **User name** field and click **OK**.

☞ By default, "Guest" has no password and the **Password** field can be left empty. However, if the password was changed or the security settings are received from a server (either Distributed or Domain), the "Guest" password has to be reentered.

- To log off, simply click on **Log off**.

  The default user (typically "Guest", however, this may be changed in the main Security System dialog on page 480) is automatically logged on to replace the user.

NOTICE

If the security mode on page 461 is set to Domain (LDAP) and a user created on the LDAP server is required to change the password when logging on to the domain for the first time, this has to be done before the user is able to log on to the "WinStudio" project.

12.10 Blocking or unblocking users

Individual users in the project security system may be completely blocked from accessing the project and respectively, blocked users can be unblocked.

To block users, proceed as follows:

- Manually select the checkbox **User is blocked** in the User account dialog on page 478;
- Call the function BlockUser on page 675 at runtime or
- Select automatically if the user enters the incorrect password too often. (The number of attempts allowed is configured in the Group account dialog on page 474.)

To check whether a user is blocked, use the function GetUserState on page 685.

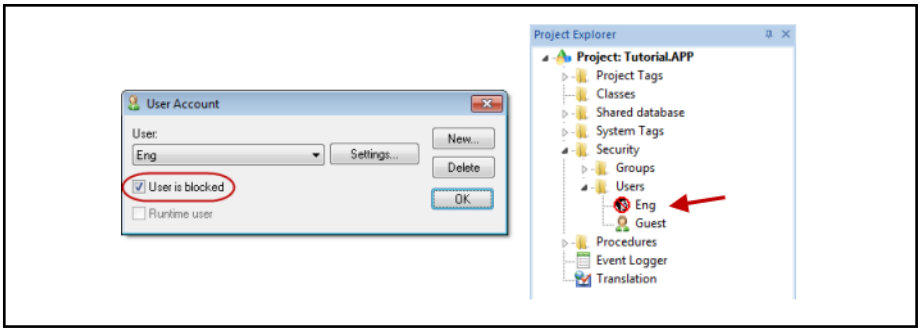


Fig. 12-31: Unblock users

To unblock a blocked user, proceed as follows:

- Disable the **User is blocked** option in the "User account dialog"; or
- use the function UnblockUser on page 689 at runtime.

12.11 Password protection for screens, symbols and worksheets

In the Project Explorer, screens, symbols and worksheets can be password-protected. Individual passwords can be assigned to each file or a single password to all files in the project.

Almost all project files are automatically encoded to prevent unauthorized analysis by third-party tools. (Screen files are not encoded as decoding them at runtime would decrease performance.)

However, files can be additionally password protected to prevent unauthorized changes or re-use by other "WinStudio" project developers.



These passwords are always case sensitive.

Assigning a password to a single file

To assign a password to a single project file, proceed as follows:

1. In the Project Explorer, right-click on the respective file. In the context menu, click on **Password protection**. The "Edit password protection" dialog is displayed.
2. Specify the new password and reenter it for confirmation.
3. Click on **OK** to close the dialog.

The file is now protected. The next time the file is opened, the password is requested.

Clearing the password from a single file

To clear a password from a single project file, proceed as follows:

1. In the Project Explorer, double-click on the respective file to open it. The password is requested.
2. With the file open for editing, right-click on the file in the Project Explorer and select **Password protection** in the context menu. The "Edit password protection" dialog is displayed.
3. Leave the **New password** and **Confirm password** boxes empty.
4. Click on **OK** to close the dialog.

The file is no longer protected. The file can be opened without entering the password.

Changing user password for all protected files

To change a single password to all project files, go to:

1. **WinStudio ▸ Tools ▸ Change password for all documents...**
2. Enter the current password for the project.

3. Specify the new password and reenter it for confirmation.
4. Click on **OK**. The verification process is continued.

All files in your project are now protected. When opening a file, the password has to be entered.

13 Project localization

13.1 Project localization, general information

The user interface of your project can be translated into multiple languages. Then the project language can be changed during runtime with a simple function call.

The translation table is a worksheet which can be used to create a multilingual user interface (MUI) for your project.

(This is different from changing the language of the development environment itself: **Extras** ▸ **Options** ▸ **General** ▸ **Language of the user interface**).

The worksheet is separated into a source column with the original text pieces of your project screens and a target column with the translated equivalents of the elements in the source column.

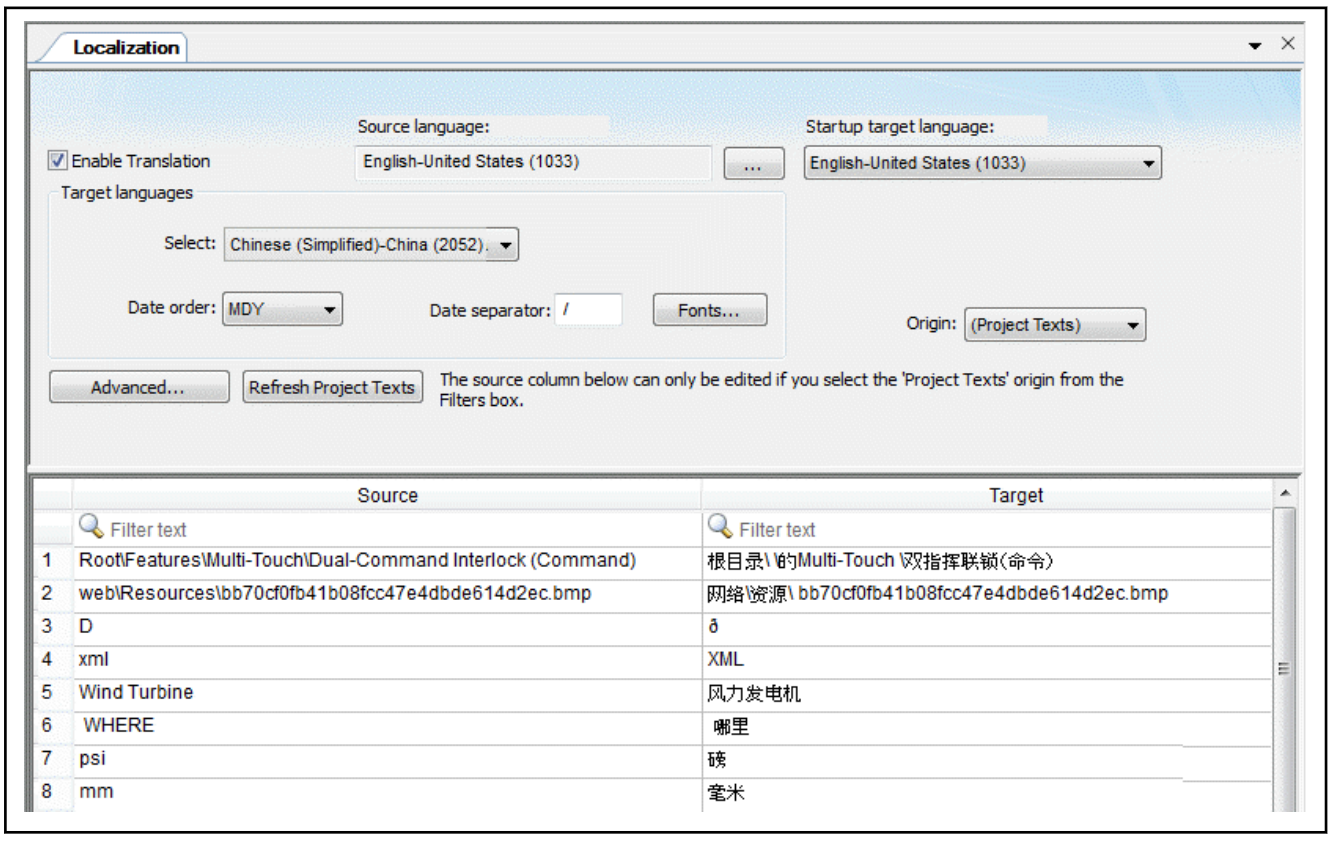


Fig. 13-1: Translation table worksheet

13.2 Adding a target language to the translation table

The translation table is used to manage the languages to translate the project to. The list of available languages corresponds to the languages available in the IndraWorks project.

To add a language, proceed as follows:

1. In the IndraWorks main menu, select the **Project** ▸ **Language** ▸ **Manage project languages** entry

   A dialog is displayed to add the required languages by clicking on "Add".
2. Open the "Localization" worksheet in the **WinStudio** ▸ **Tasks** folder of the IndraWorks project explorer. In the "Localization" worksheet, all

character strings that can be translated are displayed in the “Source” column of the table.

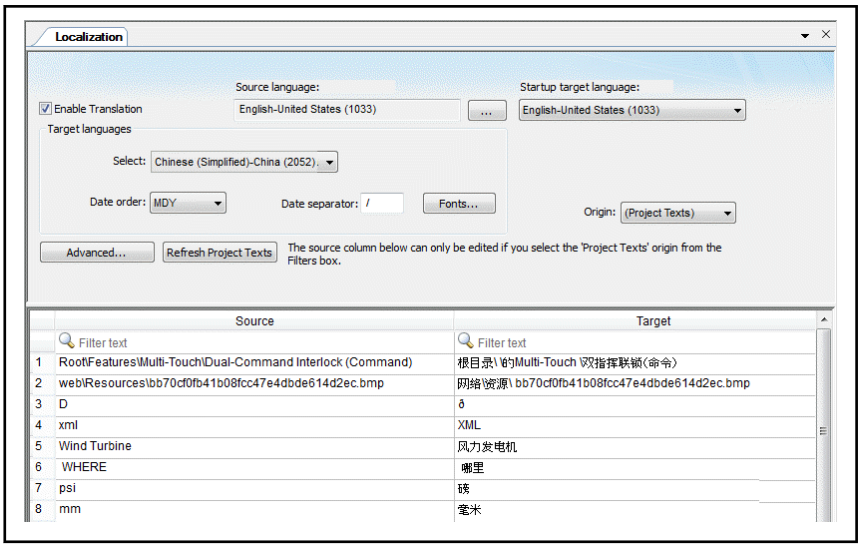


Fig. 13-2: Translation Table worksheet

3. In the "Target languages" pane, select the languages to which the application texts are to be translated. The list of available languages corresponds to the IndraWorks project languages.
4. Ensure that the **Enable translation** option is selected. When this option is deactivated, the entire translation table worksheet is deactivated and the language cannot be changed during runtime.
5. *Setting the target language:*
   - Click Add in the Target languages group. The Language dialog box is displayed.
   - In the Languages dialog box, select the language to which you want to translate your project and then click **OK**.

   The language is added to the **Select list** and a new worksheet is created for the language.

   The **Source** column of the worksheet is automatically filled with all translatable strings from your project. The **Target** column remains free.
6. Configure the **Date order** and **Date separator** as required for the target language. For example, the **Date order** for `English-United States` (i.e. American English) is typically MDY and the **Date separator** is typically /, resulting in a date format of MM/DD/YYYY.

   In many European languages, the **Date order** is, however, normally YMD and the **Date separator** is usually . which results in the date format YYYY.MM.DD.

   The language is added to the **Select list** and a new worksheet is created for the language. The **Source** column of the worksheet is automatically filled with all translatable strings from your project. The **Target** column remains free.
7. Use the **Filters** to search the worksheet for specific text elements. When entering a few characters, the list is filtered dynamically and only displays suitable elements.

The ampersand (&) is ignored when the lines are filtered. In this way, the handling of text in program dialogs which use ampersands to display keyboard shortcuts is improved.

8. Enter for each text element in the **Source** column the translation in the target column. The elements can be translated manually one after the other or several elements can be translated automatically using Google Translate.
   - Highlight the cells in the **Source** column to select them and then press <Ctrl>+<C> to copy these elements to the clipboard. It is also possible to click on the column header to select the entire column.
   - In your web browser, go to translate.google.com.
   - Select the source and target language.
   - Click in the text field on the left and press <Ctrl>+<V> to insert the elements from the clipboard. The elements are inserted as separate lines.
   - Click on **Translate**. The translation appears on the right. The elements are translated as separate lines as well.
   - Highlight the translated elements to select them and then press <Ctrl>+<C> to copy these elements to the clipboard.
   - In Rexroth WinStudio, highlight the empty cells in the **Target** column to select them. It is also possible to click on the column header to select the entire column.
   - Press <Ctrl>+<V> to insert the translated elements from the clipboard in the **Target** column. Since the translated elements were copied as separate lines, they should be in the correct lines of the worksheet.
9. Check the translated elements in the **Target** column to confirm the translations and to ensure that they correspond to the elements in the **Source** column.

To keep the original, untranslated text for a specific element, leave the Source column empty.

10. Save and close the worksheet.

Changes made to the translation table will not take effect until either the Set-Language function, page 494 is called or the project runtime is restarted.

The translation table is saved as a tab-delimited text file in the project folder, at

`[…]\Documents\WinStudio v7.4 Projects\project_name\Web\Translation.trn`

It is possible to open this file and edit it directly using Microsoft Excel if desired.

13.3 Configuring fonts for a target language

13.3.1 Configuring fonts for a target language, Basics

Configure fonts for a target language so that translated text elements are displayed in a different font, size or style which is more suitable for the respective language.

Before proceeding with this task, first add a target language to the translation table of your project.

If the layout of the runtime environment of your project is changed to one of the target languages in the translation table, the translated text elements are displayed in their original font, size and style by default. This may cause many different problems depending on the target language. If, for example, a target language does not use a Latin character set or typically consists of longer or shorter words or is read from right to left instead of left to right, screen objects with a translated text element may not be displayed correctly.

You can use the Font configuration tool on the translation table worksheet to configure new fonts, sizes and styles for the translated text elements. If you set your project interface to the target language, the translated text elements are changed according to the new configuration. The configuration can either be very general or very specific - just add more lines for more specific changes.

If there are two or more conflicting lines - for example, if one line applies to all objects and another line only to button objects - the line which most closely corresponds to a given text element is applied. See also Examples, page 491.

Configuring fonts:

1. Open the translation table by double-clicking on the "Localization" element in the **WinStudio** ▸ **Tasks** folder.

   The translation table worksheet is opened for editing.

2. In the **Target languages** pane, select the language for which fonts are to be configured and then click on Fonts. This font configuration only applies to the selected target language. The Font configuration dialog box is displayed.

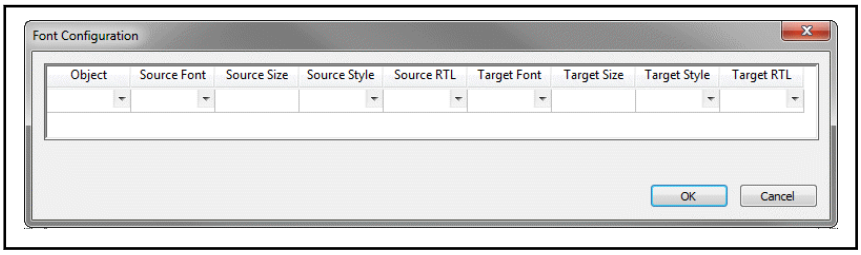


Fig. 13-3: Font configuration dialog box

3. In the **Object** column, select the screen object type to which the configuration is to apply. Select (`All`) to make it apply to all types.
4. In the **Source font** column, select the font to which the configuration is to apply. Select (`All`) to make it apply to all fonts. The list of selectable fonts includes all fonts which are installed on your computer, not only the fonts which are used in your project.
5. In the **Source size** column, select the font (in points) to which the configuration is to apply. Select (`All`) to make it apply to all font sizes.
6. In the **Source style** column, select the font style to which the configuration is to apply. Select (`All`) to make it apply to all font styles.

7. In the **Source RTL** column, select the reading direction (left to right or right to left), to which the configuration is to apply. Select (`All`) to make it apply to all both directions.
8. In the **Target font** column, select the font for which the **Source font** is to be changed. If the source font is to remain unchanged, select *. The list of selectable fonts includes all fonts which are installed on your computer.



Please ensure that the selected font is also installed on all computers or devices which are used as project runtime clients. If this is not possible - i.e. if you do not control all potential clients - select a “standard” font which is usually installed on all platforms.

9. In the **Target size** column, enter the font (in points or percent) for which the **Source size** is to be changed. If the **Source size** is to remain unchanged, select *.
10. In the **Target style** column, select the font style for which the **Source style** is to be changed. If the **Source style** is to remain unchanged, select *.
11. In the **Target RTL** column, select the reading direction (left to right or right to left), for which the **Source RTL** is to be changed. If the **Source RTL** is to remain unchanged, select *.
12. Repeat this process for additional lines, if required.
13. Click on **OK** to save the configuration and close the dialog box.

If you set your project interface to the target language, the translated text elements are changed according to the new configuration.

13.3.2 Configuring fonts for a target language, Examples

The font configuration tool on the translation table worksheet

Example 1 Ten different fonts are used in your project and all of them have different font sizes. Chinese was added as a target language in your translation table, which means that all text elements are to be transformed to a font which supports the Chinese character set.

To achieve this, the following configuration would have to be created using the font configuration tool:

Object	Source font	Target size	Source style	Source RTL	Target font	Target size	Target style	Target RTL
(All)	(All)	(All)	(All)	(All)	Arial Unicode MS	*	*	*

The * which is configured in the **Target** columns means that the original settings are to be maintained. In other words: If there are two texts with different sizes, the font is changed to Arial Unicode MS (a universal font which contains all Unicode character sets including Chinese) but the font sizes remain the same.

Example 2 A single font is used in your project but there are different font sizes depending on the place and type of use. German was added as a target language in your translation table. In German, words are often much longer. This means that if German is set as language for your project interface, the font sizes need to be decreased to ensure that the translated text elements fit in your screen objects.

To achieve this, the following configuration would have to be created using the font configuration tool:

Object	Source font	Target size	Source style	Source RTL	Target font	Target size	Target style	Target RTL
(All)	(All)	(All)	(All)	(All)		80 %	*	*

In this configuration, it does not matter how many fonts are used or what the relevant sizes are. They are all reduced proportionally to 80% of the original size.

Example 3 Arabic was added as a target language in your translation table but you also decided not to translate the names of the buttons. This means that all objects except for buttons need to be changed to a font which supports the Arabic character set. In addition, the reading direction needs to be changed from left to right (default) to right to left.

To achieve this, the following configuration would have to be created using the font configuration tool:

Object	Source font	Target size	Source style	Source RTL	Target font	Target size	Target style	Target RTL
(All)	(All)	(All)	(All)	(All)	Arial Unicode MS	*	*	Enabled
Button	(All)	(All)	(All)	(All)	*	*	*	*

Even if it seems as if both lines of the configuration apply to button objects, only the second line is actually applied since it fits best.

13.4 Translating application texts to additional languages

After adding the required project languages to the project, these languages can be used in the WinStudio application to translate the application texts respectively.

To translate application texts, proceed as follows:

1. Open the "Localization" worksheet in the "WinStudio\Tasks" folder. In the "Localization" worksheet, all character strings that can be translated are displayed in the “Source” column of the table.
2. In the "Target languages" pane, select the languages to which the application texts are to be translated. The list of available languages corresponds to the IndraWorks project languages.
3. Enter the translation of the original texts for the respective target language into the "Target" column. In order to only edit texts from a certain context of the WinStudio application, select the respective type, e.g. Alarm Control, in the "Origin" combo box.
4. Use the **Filters** fields to search the worksheet for a specific character string. When entering a few characters, the list is filtered dynamically and only displays suitable strings.
5. Save and close the worksheet.

Changes made to the translation table do not apply before the Setting the project language at runtime on page 494 function is executed or the runtime project is restarted.



The translation table is saved as a tab-delimited text file in the project folder, at

`[…]\project_name\Web\Translation.trn`.

If required, this file can be opened and directly edited with Microsoft Excel.

13.5 Activating a language at runtime

13.5.1 Activating a Language at Runtime, General Information

When activating a project language during runtime, the application type has to be generally considered:

- **HMI or MTX application**

  If an HMI or MTX application is projected, i.e., the application is executed within Indraworks Operation, no additional configuration is required to activate a language. The application is already set for language selection and the active language can be either selected in IndraWorks Operation via the main menu under **Tools** ▸ **Options** or by pressing an F/M button.

  The selected language applies for all WinStudio screens.

- **WinStudio application**

  For WinStudio applications, the following steps are required to activate a language at runtime:

  1. Select a starting language to be activated if the application is started.
  2. Add control elements at suitable locations in the application to change the active language during runtime.

  The two steps are described in the following chapters.

13.5.2 Setting the project language at program start

If multiple languages are configured for the project, the project starting language at runtime has to be specified.

Before, at least one target language has to be added to the translation table.

For further information, refer to Adding a language on page 487.

1. Open the translation table by double-clicking on the "Localization" element in the **WinStudio** ▸ **Tasks** folder.

   The translation table worksheet is opened for editing.

2. Select the project starting language in the **Startup target language** list. The list of available languages includes the source language in which the project has been developed as well as all added languages.
3. Save and close the worksheet.

Setting your project's language will also automatically set the language of the Virtual Keyboard, as long as there is a keyboard configuration file for the selected language. Keyboard configuration files are stored at `C:\Program Files\Rexroth\WinStudio\Bin\VK**.ini`, where `**` is the two-letter language code.

You can directly set the language of the Virtual Keyboard, separate from the project's language, by calling the function SetKeyboardLanguage, page 802, during run time.

13.5.3 Setting the project language at runtime

To activate a project language, WinStudio provides the command SetLanguage on page 770: By means of this function, the project language can be set at runtime where expressions can be configured.

To use this function, at least one target language has to be added to the Translation Table. For further information, refer to Adding a language on page 487.

The "SetLanguage" function requires a numerical code for the target language parameter. The codes are shown in parentheses in the list of languages – for example, "English-United States" is "1033".

Local IDs are the unique language/regional codes used in Microsoft Windows. For further information, refer to msdn.microsoft.com/library/ms912047

The following very basic examples illustrate how to draw two button objects that switch the project language between English and French.

1. In the "WinStudio" folder, click on a WinStudio screen to open it for editing.
2. In the "WinStudio screen tools" tool window, click on **Button**.
3. Draw a button object into the WinStudio screen.
4. Double-click on the button object. The "Object properties" dialog is displayed.
5. In the **Caption** box, type "English".
6. Click on **Command**. The command animation properties are displayed in the dialog.
7. In the first row of the **On Down** tab, in the **Expression** field, enter `SetLanguage(1033)`.

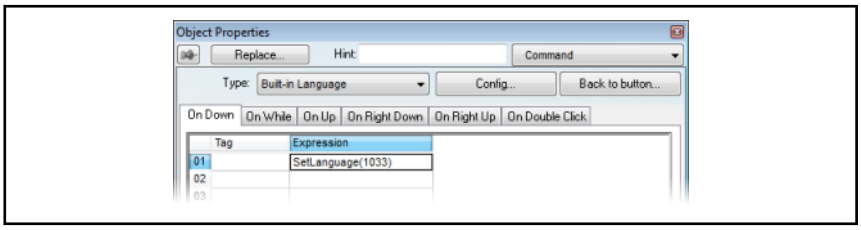


Fig. 13-4: Setting the project language (1)

8. Close the "Object properties" dialog
9. Duplicate the “Button” object, either by copy-and-paste or by Ctrl+click.
10. Repeat steps 4 through 8, replacing the caption with "French" and the expression with `SetLanguage(1036)`.

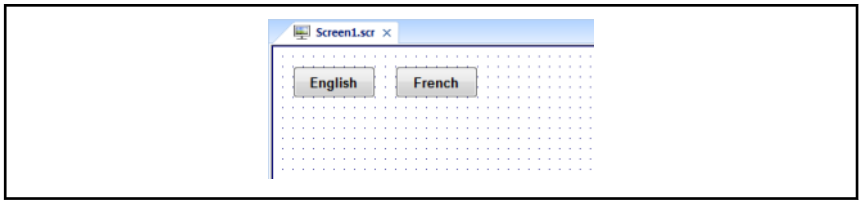


Fig. 13-5: Setting the project language (2)

11. Save and close the WinStudio screen.

During project runtime, clicking each button sets the language of the entire project, using the translated text from the translation table.

Setting your project's language will also automatically set the language of the Virtual Keyboard, as long as there is a keyboard configuration file for the selected language. Keyboard configuration files are stored at `C:\Program Files\Rexroth\WinStudio\Bin\VK**.ini`, where `**` is the two-letter language code.

You can directly set the language of the Virtual Keyboard, separate from the project's language, by calling the function SetKeyboardLanguage, page 802, during run time.

13.6 Disabling Translation of Selected Screen Objects

By default, translation is enabled for all screen objects that have text to be translated. However, translation of selected objects can be disabled if the original text has to be kept.

1. Double-click on a screen object to open its "Object Properties" dialog.
2. In the dialog, look for the **Enable Translation** option. If the option is not available in the basic properties of the object, click on **Advanced** to open the advanced settings.
3. Disable the **Enable Translation** option.
4. Close the "Object Properties" dialog

Once the option is disabled on an object, its text is no longer translated during project runtime.

The text is still added to the Translation Table and can be processed though the automatic translation engine along with all other project texts, however, the resulting translation is not applied to the object during runtime.

13.7 Configuring the advanced translation settings

Before proceeding with this task, first enable the translation and add at least one target language to the translation table of your project.

Configuring the advanced translation settings:

1. Open the "Localization" worksheet in the **WinStudio** ▸ **Tasks** folder.
2. Click on **Advanced**. The Advanced dialog is displayed.

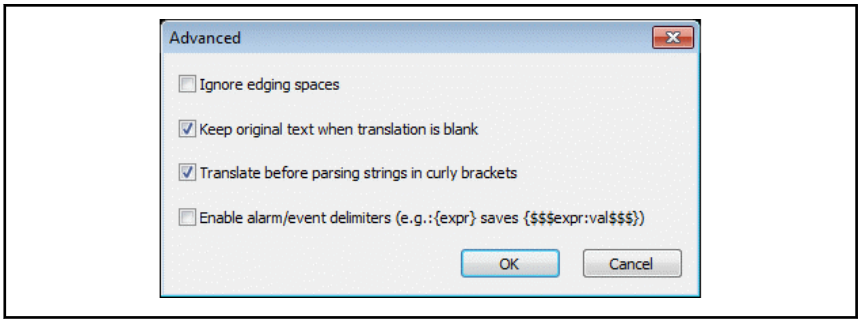


Fig. 13-6: Advanced translation settings

3. Select or remove options as required.

Option	Description
Ignore edging spaces	Spaces at the beginning or end of character sets which are to be translated are ignored. If this option is not selected, the character strings must match exactly.
Keep original text when translation is blank	Displays the character string in der **Source** column, if the **Target** column is empty. If this option is not selected, the original text may be replaced by a blank field.
Translate before parsing strings in curly brackets	Translates the characters before tags/expressions which are configured in curly brackets are assessed. If the original text is, for example, "The current user is {UserName}" and the Portuguese translation is "O usuário atual é {UserName}", the character string is translated correctly, before the value of `UserName` is determined. If this option is not selected, the tag/expression is assessed, before the character string is translated. This means that if no translation was configured which contains the exact value of the tag/expression, the character string is not translated.
Enable alarm/event delimiters	Enables the use of delimiters in alarm/event messages. In this way, when messages are saved, they are formatted in a way that ensures that they are translated correctly when they are called. If this option is not selected, all tags/expressions in curly brackets are assessed prior to translation when the message is generated and saved. In other words: The message is saved as a word for word character string and if no translation was configured which contains the exact value of the tag/expression, the character string is not translated. Enabling this option, however, ensures that the message is saved with both, the tag/expression and the corresponding value at the time of generation. If, for example, an event message is configured which says “The current user is {UserName}”, it is saved as “The current user is {$$$UserName:Michael$$$}“. This means that the value of `UserName` at the time the message was generated was `Michael`. When the message is called, it is translated as described above under **Translate before parsing strings in curly brackets**. The saved value is inserted if required.

4. Click on OK to close and save the dialog and then close the translation table.

14 Testing and debugging projects

14.1 Output (LogWin)

Use the "WinStudio log" window to display error analysis messages generated during project runtime. The window displays OPC, DDE, and TCP/IP transactions, module activations, trace messages, and so on.

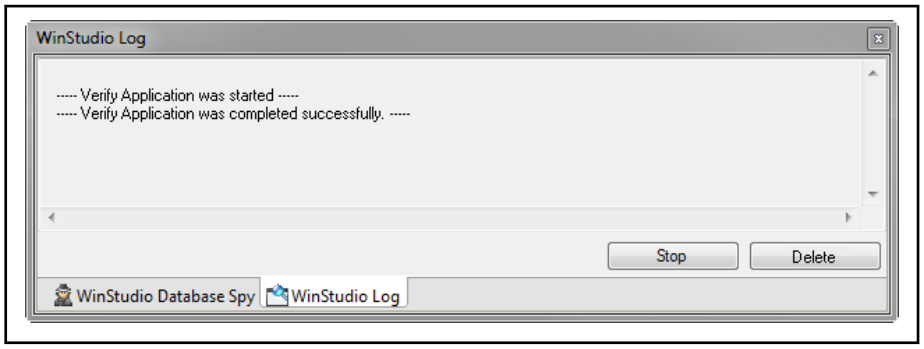


Fig. 14-1: Example of output window

The "WinStudio log" window is dockable, i.e. it can be moved to another location in the development environment. To do so, click on the title bar and drag the window to a new location. Release the mouse button to attach or "dock" the window to its new location.

Configuring the log to show additional information

By default, the log shows only error analysis and error messages — that is, messages indicating that the project is not running properly. If the log displays **all** messages created by "WinStudio", the output quickly becomes confusing.

To configure the messages displayed in the log, proceed as follows:

1. Right-click anywhere in the "WinStudio log" window and then click **Settings** in the context menu. The "Log settings" dialog is displayed:
2. On the "Log options" tab of the dialog, select the specific types of messages that are to be displayed in the log.

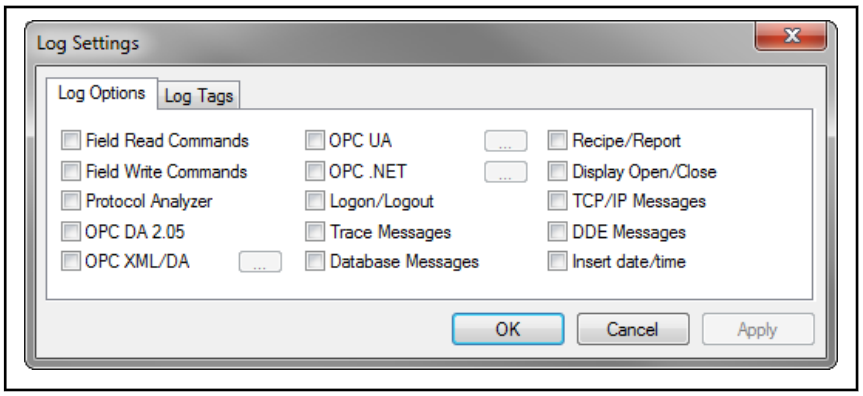


Fig. 14-2: Log settings - Options tab

Options	Description
Field read commands	Displays all field read commands that are sent to connected devices
Field write commands	Displays all field write commands that are sent to connected devices.
Protocol analyzer	Displays messages generated by configured device drivers on page 413.

Options	Description
OPC DA 2.05	Displays messages generated by the OPC DA 2.05 Client runtime task, page 427.
OPC XML/DA	Displays messages generated by the OPC XML/DA Client runtime task.
OPC UA	Displays messages generated by the OPC UA Client runtime task, page 433.
OPC .NET	Displays messages generated by the OPC.Net Client runtime task.
Logon/logout	Displays a message whenever a user logs on or logs out (For further information, refer to Security on page 461.)
Trace messages	Shows messages generated by the **Trace()** function. This function is used to generate customized messages from the project to trace execution of the application
Database messages	Displays messages generated by ODBC on page 397 and ADO.NET on page 404 database interfaces
Recipe/report	Displays messages of the tasks for recipe on page 395 and reports on page 396.
Displaying Open/close	Displays detailed information whenever a screen on page 170 is opened or closed: **Disk load time:** Time to load the screen file from the disk into memory **Open time:** Time to load the WinStudio screen, including initialization tags used in the screen and running any "OnOpen" scripts or functions. **Total load time:** Total time to load the screen (includes disk load time and open time above). **"First draw" time:** Time for the first drawing of screen objects **"First OnWhile" time:** Time for running any "OnWhile" scripts or functions for the first time **Total open time:** Total time to open the screen (includes first draw time and first OnWhile time as described above). **Close time:** Time to close the screen, including finalizing tags used in the screen and running any "OnClose" scripts or functions. **Total close time:** Total time to close the screen, including the time to write the screen file on the disk This information can be used to analyze runtime performance on low-end end systems. If a particular step of opening or closing takes an unusually long time, it can be identified and redesigned
TCP/IP Messages	Displays messages generated by TCP/IP communications on page 453.
DDE messages	Displays messages generated by DDE communications on page 455.
Insert date/time	Add a timestamp to each message

Tab. 14-1: Log settings - Options tab

3. If **OPC XML/DA**, **OPC UA**, **OPC .NET** was selected, specify precisely which messages are to be displayed in the output window.

   Required steps:

   Click the Browse button to the right of the option. An additional Log settings dialog is displayed.

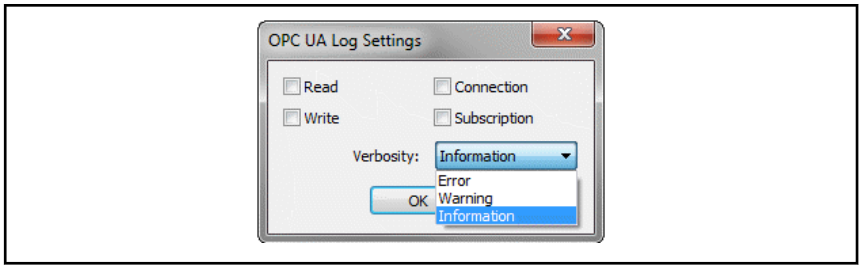


Fig. 14-3: OPC UA log settings dialog

Each task has its own settings but the descriptions of the settings are the same for all three tasks.

Select the type(s) of messages to be displayed

Option	Description

Connection	Activate trace messages for changes of the connection status between the OPC client (i.e. your project) and the OPC server.
	Example of messages:
	`OPC UA: Connection established with server "Connection1"`
	`OPC UA: Error to connect to server "Connection2" - Time out waiting for server response`

Subscription	Activate trace messages for subscriptions to server elements such as the creation of new subscriptions and changes of the data type. Example of messages: `OPC UA: Group 1 => Subscription created`

Read	Activate trace messages for reading processes.
	Example of messages:
	`OPC UA: Read Group 1 Started - OK`
	`OPC UA: Read Group 1 Completed - OK`
	`OPC UA: Read Group 25 Started - Error, asynchronous reading pending for the current group`

Write	Activate trace messages for writing processes.
	Example of messages:
	`OPC UA: Write Group 1 Started [Line 1 = 10.25, Line 42 = 20.45] - Status OK`
	`OPC UA: Write Group 1 Completed [All Items] - Status OK`

Select the verbosity level from the Verbosity list.

Option	Description
Error	Critical problems which have lead to the breakdown of the communication. These problems have to be remedied before the communication can be resumed.
Warning	Non-critical problems which may lead to the breakdown of the communication under different conditions. These problems are to be remedied as soon as possible. Includes the above-mentioned **errors**.
Information	All trace messages for communication. This is the default setting. Please note that these messages may be extremely detailed depending on which type(s) of messages were selected for display and how many member subscriptions were created. Includes the above mentioned points **Error** and **Warning**.

Click on **OK** to close the dialog.

4. In the "Log Tags" tab of the dialog, click **Add** to browse for project tags on page 104.

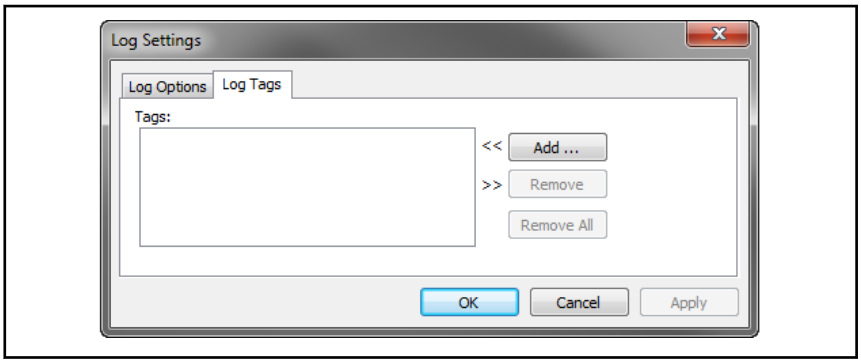


Fig. 14-4: Log settings — Tags tab

The log pane of the LogWin window displays a message if the value of a selected tag changes.

5. Click **OK** to save the settings and close the "Log settings" dialog.

14.2 Database spy

The "Database spy" window is a tool for error analysis to display and change tag values and execute and test functions and mathematical expressions.

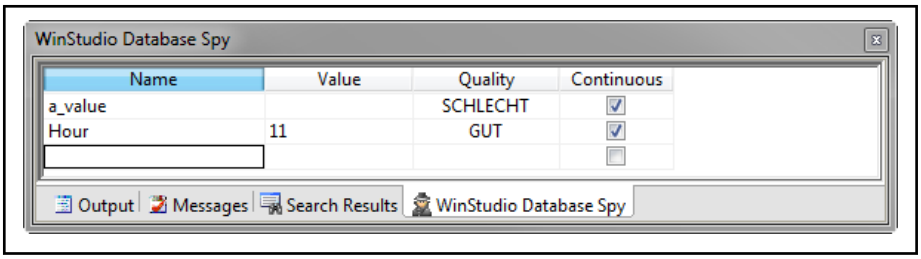


Fig. 14-5: Example of database spy window

The window displays the following information:

- For each desired date, the following is displayed at runtime:

- **Name**: Indicate an application variable, an internal variable or an expression to be monitored
- **Value**: Displays the value returned by the tag/expression
- **Quality**: Displays the quality (GOOD or BAD) of the value returned by the tag/expression
- **Continuous**: Select this option to have the project continuously evaluate the tag/expression

In order to monitor a variable, enter its name into the column or click on an empty field in the "Name" column and open the context menu. By means of the "Insert variable or expression" function, the browser dialog can be opened alternatively and a variable can be selected.



The database spy is dockable, i.e. it can be moved to another location in the development environment.

Click on the title bar and drag the window to a new location.

Release the mouse button to attach or dock the window to its new location.

14.3 Using the LogWin module

This module provides a continuous record of activities and tags for debugging over long periods of time. It creates a file in which the results of the data collection can be stored. The size of this file increases until the logging process (data collection) is stopped.

Use the LogWin module (local and remote) to record DDE on page 455, OPC on page 427 and TCP/IP on page 453 transactions, activate modules, trace tags, etc.

To enable the LogWin module as a part of the project, proceed as follows:

1. In the context menu of the "WinStudio" folder, select **Tools** ▸ **Status**. The "Execution Tasks" dialog is displayed:

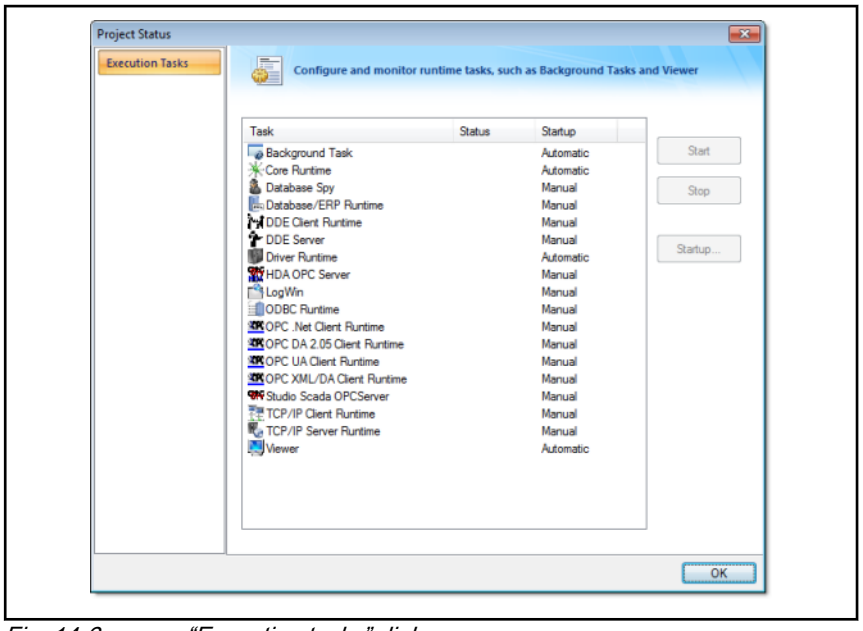


Fig. 14-6: "Execution tasks" dialog

2. Select LogWin from the **Task** list.

3. Select the LogWin task and click on **Start**. The "Start" dialog is displayed.

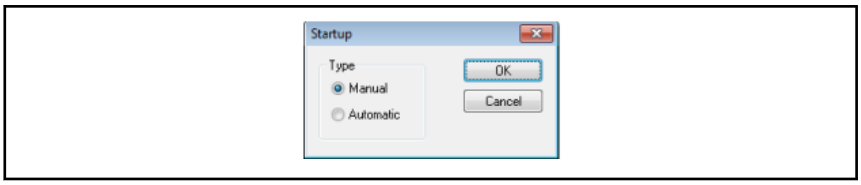


Fig. 14-7: Startup dialog

4. Enable the **Automatic** radio button.
5. Click on **OK** to apply the selection and close the "Startup" dialog.
6. Click on **OK** to close the "Execution tasks"dialog.

- To start the LogWin module directly on a Windows Embedded device, in the CEView menu bar, select **Tools** ▸ **LogWin**.
- To activate logging on a thin client on page 505 select the **Log** function in the "Web" project settings. The “Web settings” dialog can be found in the WinStudio context menu under **Tools** ▸ **Web settings**.

14.4 Using remote tools

14.4.1 Remote database spy

The **Remote database spy** tool can be used to monitor the database spy on page 78 of a "WinStudio" project executed on a remote computer. The execution task "Database spy" has to be enabled for the project and the remote computer has to be at runtime on page 51.

To use the "Remote database spy" tool, proceed as follows:

1. In the WinStudio context menu, click on **Tools** ▸ **Remote database spy**. The "Remote computer" dialog is displayed.
2. Enter the IP address of the remote computer, as illustrated below.

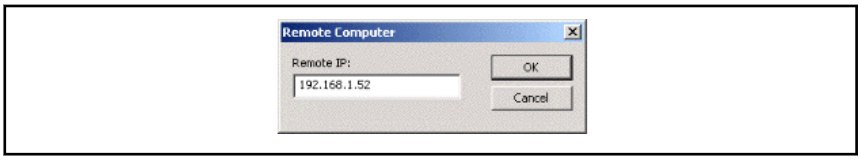


Fig. 14-8: Entering a remote IP

Verify the IP address **192.168.1.52** of the computer to which the connection is to be established. Verify the IP address of the computer to which the connection is to be established.

3. Click on **OK** to connect to the specified address. If the connection is successful, the "Remote database spy" window is displayed.

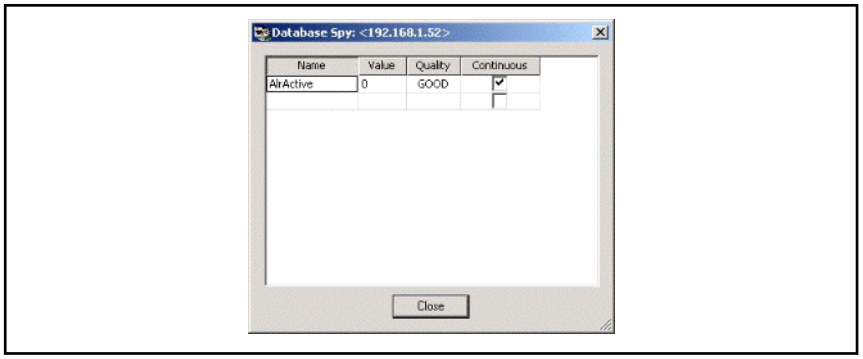


Fig. 14-9: Remote database spy

Tags cannot be added or removed remotely. The tag list of the database spy has to be configured on the remote computer itself.

To finish, click on **Close** to disconnect from the remote computer.

14.4.2 Remote LogWin

The **Remote LogWin** tool can be used to monitor the Output log (LogWin), page 79 of a "WinStudio" project executed on a remote computer. The execution task "LogWin" has to be enabled for the project and the remote computer has to be at runtime, page 51.

Using the Remote LogWin tool:

1. In the WinStudio context menu, click on **Tools** ▸ **Remote LogWin**. The "Remote computer" dialog is displayed.
2. Enter the IP address of the remote computer, as illustrated below.

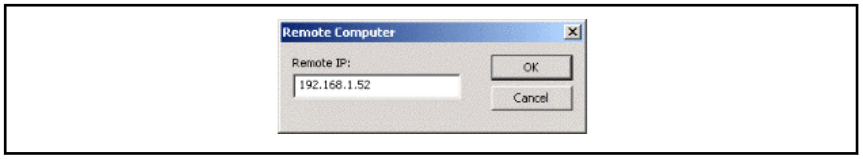


Fig. 14-10: Entering a remote IP

Verify the IP address **192.168.1.52** of the computer to which the connection is to be established. Verify the IP address of the computer to which the connection is to be established.

3. Click on **OK** to connect to the specified address. If the connection is successful, the "Remote LogWin" window is displayed.

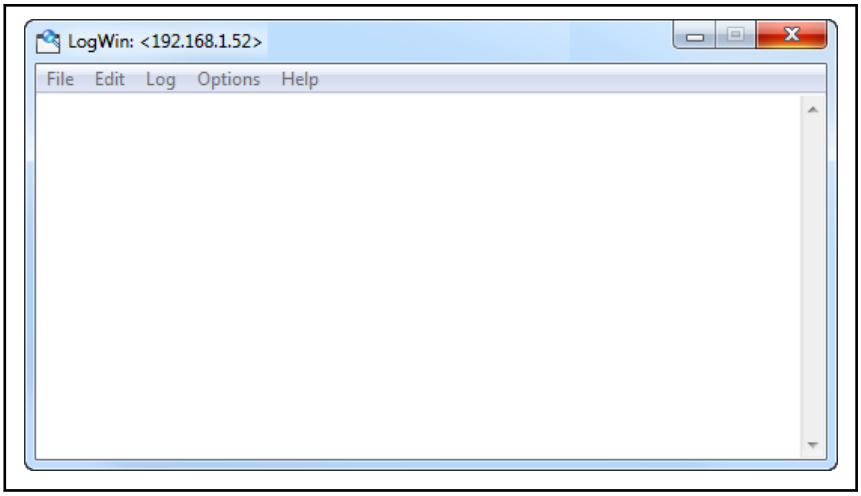


Fig. 14-11: Remote LogWin

After finishing, just close the window to disconnect.

15 Web thin client and secure viewer

15.1 Introduction into thin clients

"WinStudio" is based on a client/server architecture and supports both thin clients and thick clients. The selected type of client architecture depends on the system requirements.

Thick client

A thick (fat) client is a computer that performs most, if not all, processing activities.

Thick clients have sufficient processing power, memory, graphics, etc. to run the project. Thick clients can exchange data with the server as required (e.g., data logging, program synchronization). Servers can also be thick clients for other servers.

In WinStudio, thick clients are handled via Remote management, page 545.

Thin client

Thin (lean) clients are computers that depend primarily on a central server for processing activities. Thin clients require a processor with minimal memory, a browser (Internet Explorer) and a network connection (e.g., to the Internet or Intranet). The term "Web client" refers to a thin client solution. "WinStudio" enables selecting the type of browser that suits best the runtime environment.

- Microsoft Internet Explorer. This type of browser is recommended for Desktop or remote access environments.
- "Bosch Rexroth" Secure Viewer, a dedicated viewer for "WinStudio" projects. The secure viewer cannot be used to browse on other websites. I can only connect to a defined "WinStudio" server. The secure viewer is intended for plant floor environments were browsing to other websites is not required or desired.

This chapter concerns the application of thin clients with "WinStudio". Thick clients are covered in a subsequent chapter.

Thin clients in "WinStudio"

"WinStudio" enables creating screens that can be visualized on remote stations with a regular web browser (e.g., Internet Explorer). The station where the graphical interface (screens) can be visualized on the web browser is called “Thin client”. The thin client (browser) is the host.

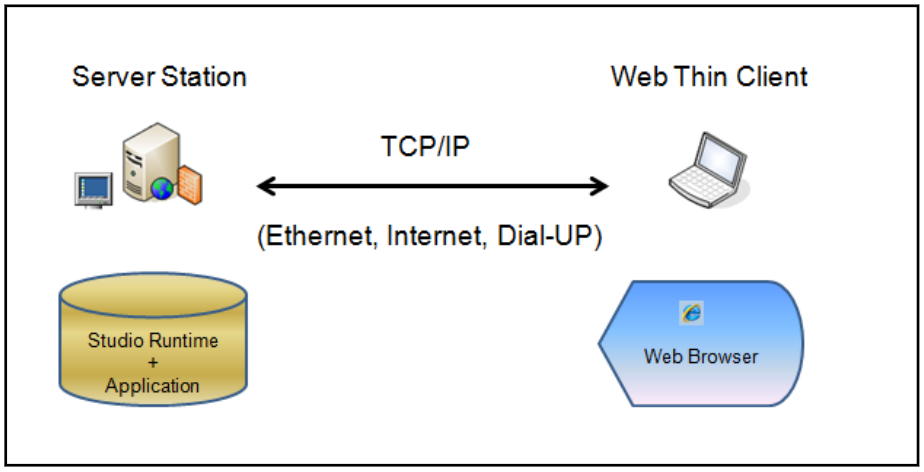


Fig. 15-1: Typical thin client architecture

"WinStudio" is only installed on the server station. Additionally, the project (screen files, tags database, configuration worksheets, etc.) is only stored on the server. The development application or the runtime project do not need to

be installed on the thin client. This solution provides a high level of flexibility as the screen and online/history data of the server can be accessed from any computer physically linked to the server station (via a TCP/IP link) without installing the development application or the runtime project on the thin client.

All background tasks (math, scheduler, etc.) and communication tasks (e.g., driver, OPC, DDE TCP/IP) are executed only on the server station. Thin clients can load graphical interfaces configured on the server (screens with objects and animations) and display the online values of the tags configured on the server as well as history data (alarm, event and trend history data).



Since VBScript and the "WinStudio" scripting language can be associated with a screen as well as a command animation for an object, these scripts can be executed on thin clients.

Any computer or device (e.g., a PDA with Windows Mobile 5.0) running Microsoft Internet Explorer 6.0 (or later) or the Bosch Rexroth Secure Viewer program can be used as thin client for projects. Additionally, "WinStudio" provides a sophisticated **security system** to prohibit unauthorized access to the project.

From thin clients, it is possible to visualize data from the server and to change set points, acknowledging alarms and/or sending commands to the server. When configuring the project, all commands can be optionally enabled/disabled from the thin client to the server. Even if the server has disabled all commands from the thin client, the thin client can still read data from the server, however, not send any data to the server.

Thin client – Competitive advantages

"WinStudio" is built on a client/server architecture that supports **true** thin clients. This capability is integrated into "WinStudio" and is not an add-on. This means that:

- The "WinStudio" server supports a large number of concurrent thin clients (up to 128 IE-based and 128 Secure Viewer-based). Each thin client can access the same or different screens as other thin clients.
- The "WinStudio" server knows which display is accessed by each thin client and automatically "pushes" all updated tag values to the thin client. This way, the display of the thin client is continuously updated and the screen does not need to be updated manually.
- The "WinStudio" server can support runtime language switching for each thin client. This means that one thin client can display a screen in English while a different thin client can display the same screen in Spanish.

Many competitive products offer either a static display on a thin client (i.e., it has to be "refreshed" manually to get the latest data), a terminal server solution (requires the server to build multiple instances of the project to support each thin client) or a thin client solution similar to "WinStudio" but with expensive "add-on" software products.

Other thin client advantages include:

- Optional Secure Viewer that does not allow navigation outside the "WinStudio" project. Secure Viewer is not based on Microsoft's Internet Explorer.
- Option of running VBScript and the built-in scripting language on the thin client
- Option of building thin client projects using Windows Embedded devices as server.

- Ability to support redundant web servers and data servers with automatic switchover.

Thin client licensing

The maximum number of thin client stations connected simultaneously to the server depends on the settings of the license installed on the server. No licenses need to be installed on the thin client. For further information, refer to License settings on page 33.

15.2 Creating a simple thin client program

As described in the following, there are different ways to build thin client programs. The following procedure enables developing a simple, unsecured thin client program.

Procedure A: thin client using NTWebServer, local loop back

An easy way of developing and testing a thin client program is using the lightweight web server **NTWebServer** on the development PC, initiating the runtime project and using Microsoft Internet Explorer on the same PC to display the websites.

Step 1: Configure the "WinStudio" settings

1. To open the configuration dialog for the web settings, go to the WinStudio context menu and click on **Tools ▸ Web settings**.

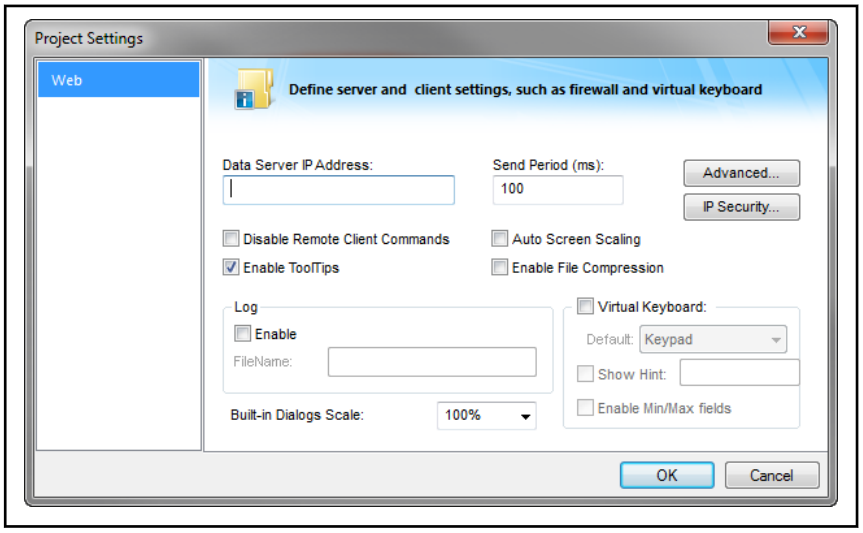


Fig. 15-2: WinStudio settings, thin client

- In the “Data server IP address” field, enter **127.0.0.1**
- Ensure that the **Auto screen scaling**, **Enable file compression** and **Enable tooltips** checkboxes are enabled.
- Click on the **IP security button**. Ensure that the **Enable** checkbox is disabled. Click on **OK**.
- Click on **Advanced**. Ensure that the **Web tunneling gateway enabled** checkbox is disabled.

  Click on **OK**.

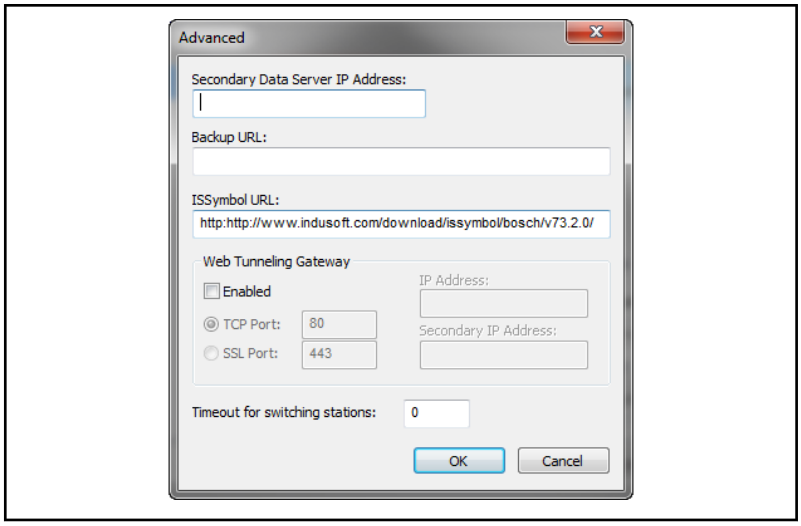


Fig. 15-3: WinStudio settings, thin client, advanced

2. Open the task configuration (via **Tools** ▸ **Status** in the WinStudio context menu) and ensure that the **TCP/IP server** task is set to Automatic.

Step 2: Configure database

1. In the "Variables" folder, open the **Application tags** editor.
2. Ensure that the application tags are properly set to either server or local. If the tags are to be provided to the web client, set them to **Server**, otherwise to **Local**.

Step 3: Develop screens and create HTML screens

1. Develop a WinStudio screen. Depending on the thin client screen size, it may be necessary to develop a separate set of screens with a different resolution. “Auto screen scaling” is supported (enabled in a prior step), however, has natural limitations.
2. Save and close all project screens.
3. Ensure that a startup screen has been defined (on the **Viewer** tab of the WinStudio Properties dialog).
4. Individual screens can be saved as HTML by selecting the **Save as HTML** or **Save screen group as HTML** option in the application menu or saving all screens and screen groups as HTML by selecting **Save all as HTML**.
5. If only one screen is updated and, particularly, if the configuration of the web settings is changed, the function "Verify application" is to be applied (in the "WinStudio" folder context menu).
6. The HTML pages (websites) are stored in the "Web" subfolder of the project folder.

Step 4: Install the NTWebServer

1. NTWebServer is "Bosch Rexroth"'s lightweight web server for Windows environments. `NTWebServer.exe` is located in `[…]\WinStudio\Bin`
2. Copy `NTWebServer.exe` from the `\Bin` folder and paste it into the `Web` main folder of the project folder.
3. In the `Web` subfolder, double-click on NTWebServer to start it.
4. A new window opens. A message is to be displayed at the bottom edge indicating that NTWebServer is ready-to-receive. If a message is displayed indicating that NTWebServer failed to open a socket, it is most

likely caused by Microsoft IIS (web server) running in a background mode. If this is the case, IIS has to be stopped and NTWebServer re-started.



Web servers typically work or "listen" on the TCP/IP port 80 of a computer. Only one running process can receive a defined port. Respectively, if other processes, e.g., SCADA third-party software, is already receiving on port 80, this process might collide with the server process. Either, one of the processes has to be configured to receive from a different port or the Task Manager has to be used to end the conflicting process. If the conflicting process cannot be identified, open the Windows command prompt and enter the following command to get a list of all networking processes:

```
netstat -a -o
```

Step 5: Start the runtime project

1. In the WinStudio context menu, click on **Start application**.

Step 6: Launch Microsoft Internet Explorer and connect to the Web Server

1. Click on the **Start** button (or Alt tab) to access the Microsoft Internet Explorer program.
2. Start the Internet Explorer and type the initial address (homepage).

   E.g. `http://127.0.0.1/startup.html`.
3. Note that the `startup.sg` (or whatever the startup display or screen group name is) has an HTML file extension when accessed from Internet Explorer.
4. Sign on as **Guest** with no password, provided security has not been enabled.

Procedure B: Thin client using NTWebServer, network IP

Once procedure A has been completed, the next step is to enable network connected thin clients instead of using the local loopback. The IP address of the PC has to be known, as seen from the network. This information can be obtained from the Network Settings in the Control Panel or by entering the **GetComputerIP()** function in a Rectangle Object on a screen and running the project.

Step 1: Configure the "WinStudio" settings

1. Stop all active runtime projects.
2. Open the Web Configuration dialog and enter the IP address of the computer into the Data Server IP address field (as viewed from the network), e.g., 192.168.1.100.
3. Execute the **Verify Application** tool. This changes the data server IP address on the websites, so that the thin client automatically exchanges data with the correct data server.

Step 2: Start the runtime project

1. In the WinStudio context menu, click on **Start application**.
2. Ensure that NTWebServer is still running.

Step 3: Launch Microsoft Internet Explorer and connect to the Web Server

1. Click on the **Start** button (or Alt tab) to access the Microsoft Internet Explorer program.
2. Start the Internet Explorer and type the initial address (homepage).

E.g. `http://192.168.1.100/startup.html`.

3. Sign on as **Guest** with no password, provided security has not been enabled.

Procedure C: Thin client using IIS, network IP

Once procedure B is completed, the next step is to enable IIS as web server for the thin client instead of NTWebServer. In the following, the basic steps for IIS configuration are explained. However, depending on the network and the operating system environment, further settings may become necessary, e.g. user security.

Step 1: Configure the "WinStudio" settings

1. Ensure that NTWebServer is not running.
2. Click on the Start button and then on "**Control Panel** and **Administrative Tools**".
3. Select (click on) **Internet Information Services**.
4. Expand the **Websites** tree structure to view the **Default website**.

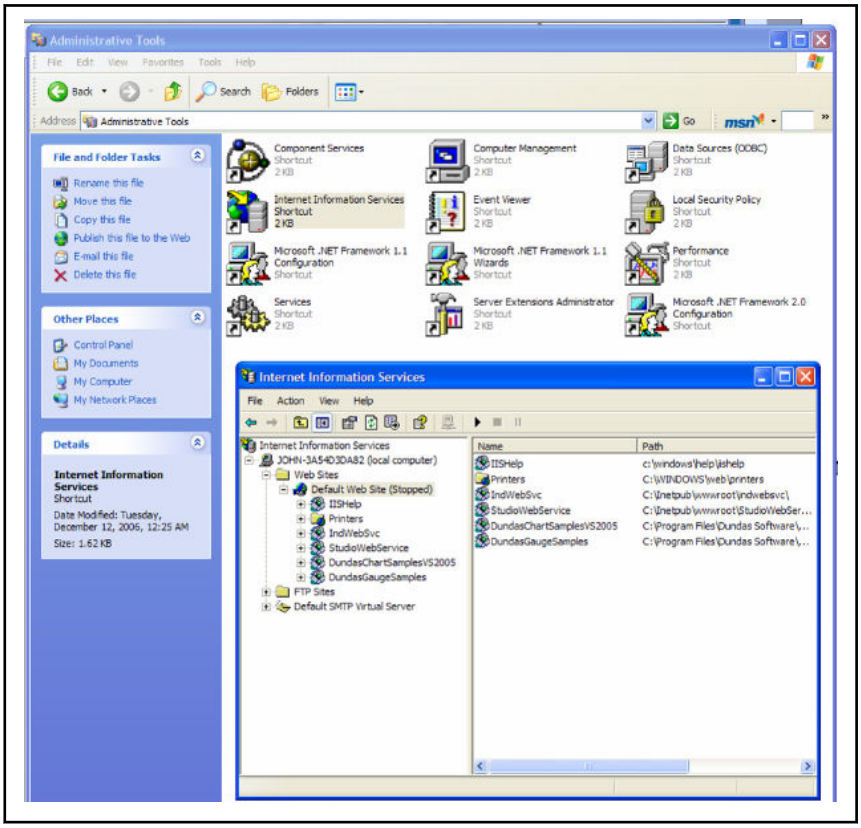


Fig. 15-4: Default website

5. Right-click on the default website and select **Properties** to open the Default Website Properties dialog.

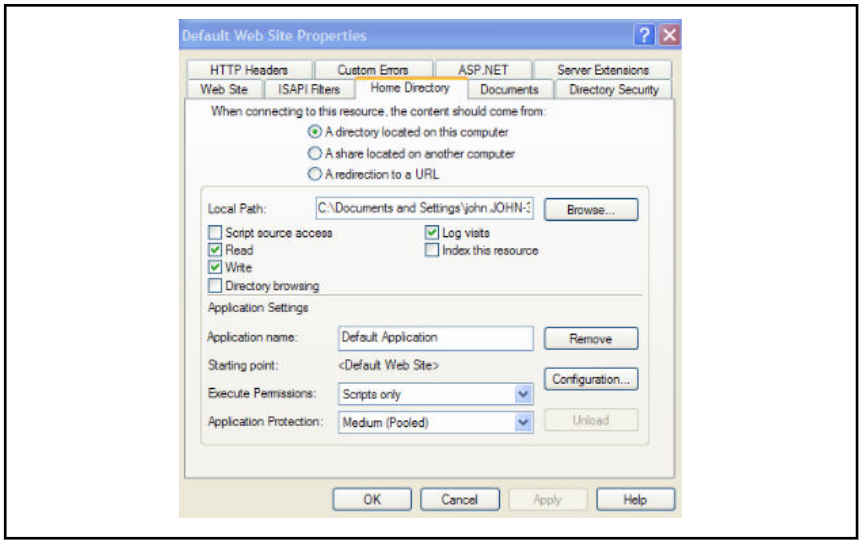


Fig. 15-5: Default website, Properties

6. Click on the **Root directory** tab. Click on the **Local path browse** button and point to the "Web" subfolder of the project.
7. Click on the **Website** tab. Ensure that the TCP port is set to 80. Click the **Advanced** button to activate the IIS web server to respond to specific IP addresses and IP port numbers.
8. If the web server is behind a proxy, make sure to enable the HTTP Keep-Alives checkbox has to be selected. It is recommended to keep this checkbox enabled at any time.

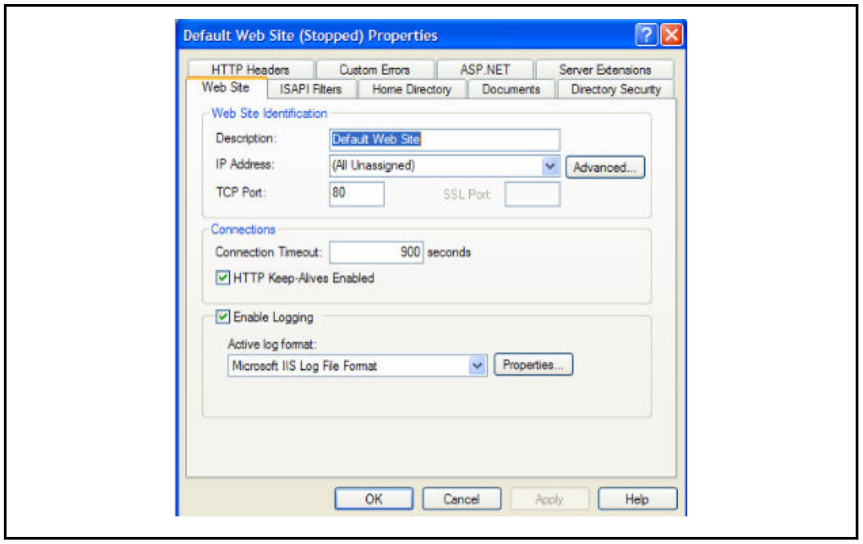


Fig. 15-6: Default website, Properties (2)

9. Click on the **HTTP headers** tab. Click on the **MIME map file types** button to open the **File types** dialog. Afterwards, click on the **New type** button to add a new MIME type. Enter the file extension in the **Associated extension** field. In the **Content type (MIME)** field, enter the MIME type followed by a **/** character, followed by the file extension name (application/studio), e.g.

   Associated extension: .scc

   Content type: application/studio

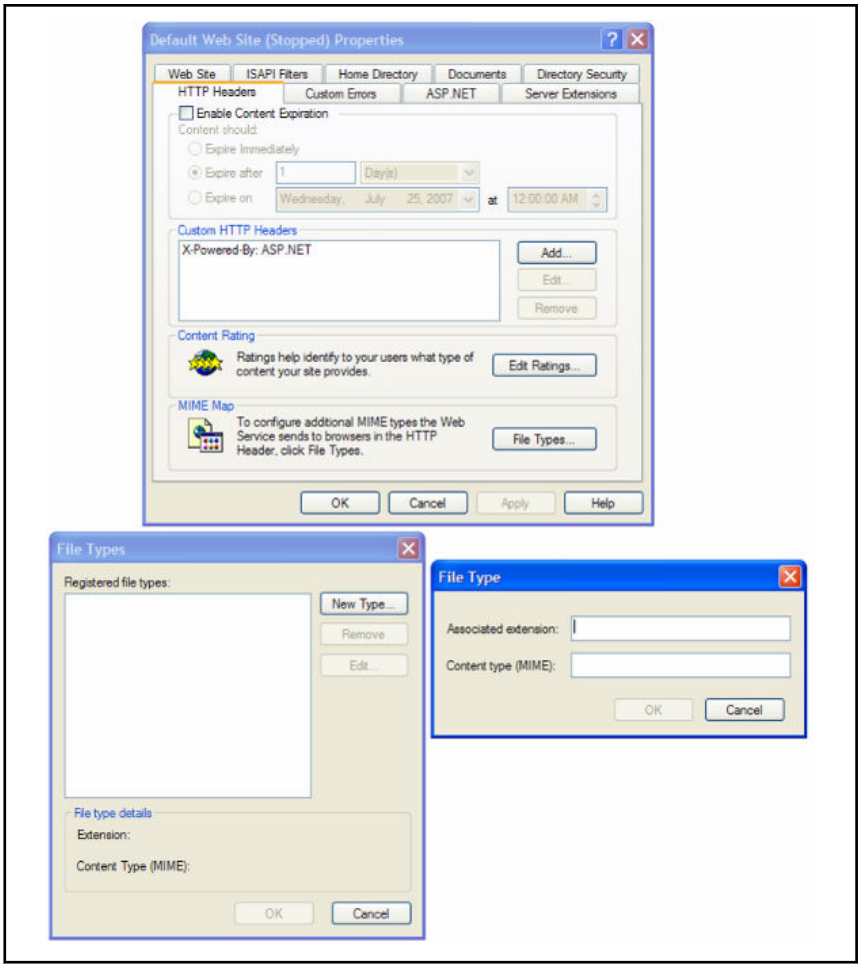


Fig. 15-7: Default website, Properties (3)

10. Click on the **Directory security** tab. On this tab, the settings for anonymous user access and authentication control as well as secure communications (i.e., using SSL 3.0) can be configured.

11. Click on the **Anonymous user access and authentication control edit** button to open the **Authentication methods** dialog. Normally, making settings in this dialog is not required. However, depending on the security system your network administrator installed, an additional configuration might be required.

12. The

    Secure communications server certificate

    button opens a wizard to define a certificate to support secure communication via SSL.

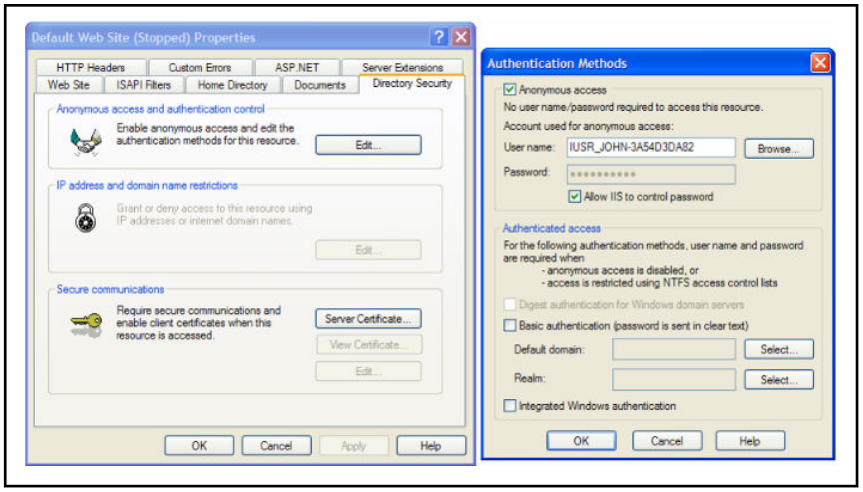


Fig. 15-8: Default website, Properties (4)

MIMEs

- MIME types (Multipurpose Internet Mail Extensions) instruct a web browser how to handle files received from a server. For example, if a web browser requests an item on a server, the MIME type of the object is requested as well. Some MIME types, like graphics, can be displayed inside the browser. Others, such as word processing documents, require an external helper program to be displayed.

  If IIS delivers a website to a client web browser, the MIME type of the data it is sending is also transmitted. If there is an attached or embedded file in a specific format, IIS also transmits the MIME type of the embedded or attached file to the client program. The client program receives information on how to process or display the data received from IIS.

  IIS serves only files with known extensions registered in the MIME types list or the operating system. IIS enables configuration of additional MIME types and changing or removing MIME types. Removing MIME types in IIS does not block the access to this MIME type by other programs, provided it is also registered in the operating system.

  IIs is preconfigured to recognize a default set of global MIME types. These MIME types are recognized by all websites created in IIS. MIME types can also be defined at the website and directory levels, independent of one another or of the types defined globally. When viewing MIME types at the website or directory level, only the types unique to that level are displayed, not all types inherited from the next level up.
- IIS returns a 404.3 error if a client request refers to a file name extension that is not defined in the MIME types.
- MIME configuration is usually only required for Windows Server 2003, not Windows XP or Vista due to Windows 2003 default settings.
- MIME types include all file extensions found in the "Web" directory. These include:
  - .app
  - .bin
  - .csv
  - .gis
  - .html

- .ico
- .ini
- .lst
- .rtgis
- .scc
- .scr
- .sg
- .stmp
- .tra
- .txt

Step 2: Start IIS and start the "WinStudio" project

1. Ensure that NTWebServer is not running.
2. Click on the IIS Start button.
3. Run the project.

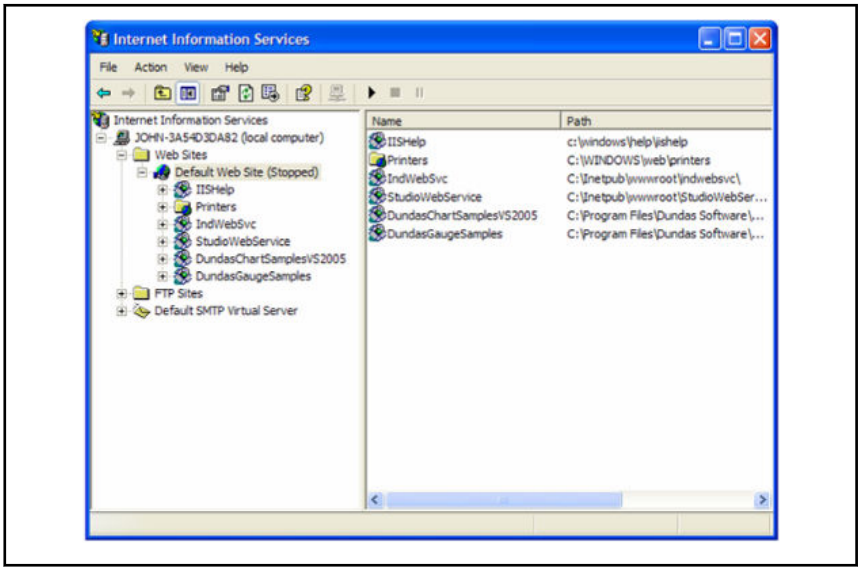


Fig. 15-9: Start IIS and execute the "WinStudio" project

Notes:

- If IIS is not installed on the PC, it can be added (to Windows XP Pro/ Windows 2000 or Windows Server 2003) by opening the Control Panel, selecting **Add/Remove Programs ▸ Add/Remove Windows Components** and checking the Internet Information Services (IIS) checkbox in the Windows components wizard. Click on the “Details” button to select various IIS components to install. Please note that the Windows installation disk might be required.
- For further information on IIS, open a browser and enter http://localhost/iishelp in the browser navigation address bar. Please note that IIS has to be running.
- NTWebServer is a Windows program, while IIS is a Windows service.
- It is **HIGHLY RECOMMENDED** to use IIS instead of NTWebServer in a runtime mode. It is more reliable and offers a higher performance than NTWebServer.

Debugging:

- If a **"Server not found"** error message is returned,
  - ping the server IP address (primary and/or backup) from the browser. To ping the server, the IP address or NetBios name can be used (e.g., enter a command prompt
  - Ping 152.57.100.25 or Ping `ServerName`.
  - Ensure that IIS is running
  - Ensure that the project is running
  - Ensure the TCP/IP server (in the project tasks) is running
  - Ensure IIS is set to the correct homepage (root directory).
  - Ensure the Port addresses are correct (HTTP – Port 80, HTTPS (SSL) – Port 443, Data – Port 1234)
  - Ensure these ports are opened at the firewall
  - If a Windows Embedded device is used, ensure that ISSymbol is properly loaded and registered
  - Ensure that the runtime license supports the web client configuration
- If a **"Cannot view page"** error message is returned,
  - Ping the server IP address from the browser
  - Stop and restart IIS
  - Ensure the MIME types are properly set
  - Ensure that the websites have been updated (i.e., Save as HTML) and use the **Verify project** tool if any web settings are changed.
  - Ensure that the Windows security settings are properly set
  - Ensure that the screen name (and website name) does not contain any spaces
- If the websites are incorrect:
  - Ensure that it is pointed to the correct primary URL
  - Ensure the backup URL (if any) refers to the correct (updated) websites

15.3 Underlying Technology

In "WinStudio" projects, there are several components used to implement the Thin Client capability.

These include:

Data server The data server is integrated into the "WinStudio" runtime. The data server has direct access to the "WinStudio" Project Tags Database (runtime) and is responsible for working with ISSymbol to make sure any tag data displayed on a web page at any Thin Client is updated with the latest value(s).

"WinStudio" can support a backup or secondary data server that is used if the primary data server becomes unavailable. The Thin Client automatically switches over to the secondary server without user intervention required.

Web server The primary web server is responsible for providing web pages on demand (i.e., when requested by the client) through navigation to various project screens by the Thin Client. The web server communicates with the Thin Client via HTTP pro-

tocol over TCP/IP. With SSL (Secure Socket Layer) encoded communication can be enabled. The web server does not need to be on the same PC as the "WinStudio" runtime project. In fact, the web server could be a non-Windows corporate web server. However, the web server requires access to the HTML files that are the project web pages.

"WinStudio" supports a secondary web server that is automatically switched to (by the Client) in case the primary web server becomes unavailable.

Web browser The web browser is located on the Thin Client PC and provides the graphical interface function with the user. Web pages (HTML) are passed to the browser via demand ("pull") and data is delivered ("pushed") to the browser by the data server whenever a tag or tags referenced on the screen displayed on the web client is updated in the tag database.

ISSymbol ISSymbol is a "Bosch Rexroth"-provided ActiveX V control element that facilitates the interaction between the browser on the web client and the web server as well as the data server. There are ISSymbol ActiveX control versions for Windows XP/Vista/7 and all Windows Embedded platforms.

The ISSymbol ActiveX control element is used for both the Internet Explorer-based and Secure Viewer-based browsers.

Web Tunneling Gateway The (Primary) Web Tunneling Gateway is a bridge between the web server and the data server that is used in one of two situations. The first is whenever data security is required (e.g., "WinStudio" data exchanged with the Thin Client needs to be encoded). In the second situation, the data server is "hidden" behind a corporate firewall and only the web server IP address (or URL) is displayed.

"WinStudio" supports a backup (secondary) Web Tunneling Gateway to be used if the primary Web Tunneling Gateway becomes unavailable. The Thin Client automatically switches over to the secondary Web Tunneling Gateway.

The Web Tunneling Gateway is automatically installed when "Rexroth WinStudio" is installed on the PC if the installation program detects that IIS is available.



- The Web Tunneling Gateway is automatically installed if IIS is detected during the installation process. Otherwise, it has to be installed manually.
- The main function of the Web Tunneling Gateway is to encapsulate data packages in HTTP or HTTPS for communication through a firewall.

15.4 ISSymbol control layer

ISSymbol is a component designed by "Bosch Rexroth" to display the screens created with "WinStudio" in the web browser and exchange data (tag values and history data) with the TCP/IP server of the project. On the thin client station, the web browser (e.g., Internet Explorer) is the container that hosts the ISSymbol control element.

ISSymbol works as a control layer between the project and the web browser, as the Java Virtual Machine for Java-based applications. This approach pro-

vides a high level of security because ISSymbol does not allow the project to access the operating system directly.

If the web browser downloads the HTML page specified by the user, it checks for ISSymbol control registration on the current computer. If it cannot be found, the browser tries to download the registry from the URL specified in the project web settings (in the advanced settings of the **Web settings** in the WinStudio context menu). The web browser is not able to display the screens from the project if the ISSymbol control is not properly registered in the thin client station.

NOTICE	**Ensure the web browser is enabled to download signed ActiveX control elements in order to automatically download ISSymbol. Otherwise, ISSymbol has to be registered manually in the thin client station.**

Check the web browser documentation for security settings in case of any questions on how to configure these settings.

Manually installing the ISSymbol control element

The ISSymbol control element can also be installed manually in the thin client station. The procedure to install ISSymbol in each operating system is described below:

Windows PC

1. Copy the following files…
   - `[…]\WinStudio\ Bin\ISSymbolReg.exe`
   - `[…]\WinStudio\ Bin\ISSymbolVM.cab`

   …into any directory of the thin client station. Ensure that both files are stored in the same directory.
2. Run **ISSymbolReg.exe** to register the ISSymbol control element on the PC.

Windows Embedded

1. Determine the OS version and processor type of the Windows Embedded device and find the corresponding redistribution folder in the "WinStudio" program directory:

   `[…]\WinStudio\ Redist\CE_version\processor_type\Bin\`

   For example, for a MIPS IV processor running Windows CE 5.0 or later:

   `[…]\WinStudio\Redist\WinCE 5.0\MIPSIV\Bin\`
2. From that directory, copy the following files…
   - `IndHTTP.dll`
   - `IndVkStd.dll`
   - `ISSymbolCE.ocx`

   …and **one** of the following Virtual Keyboard layouts…
   - `VKEN.ini` (for English)
   - `VKGE.ini` (for German)

   …into any directory in non-volatile memory on the Windows Embedded device. Ensure that all files are stored in the same directory.
3. In the Windows command prompt, execute the following command:

   `regsvrce.exe "\ ISSymbol_directory\ISSymbolCE.ocx"`

Example:

```
regsvrce.exe "\Storage Card\ISSymbolCE.ocx"
```

4. Save the registry settings to keep **ISSymbolCE.ocx** registered when rebooting the Windows Embedded device.

Windows Mobile / Pocket PC

1. Follow steps 1 and 2 of the instructions for Windows Embedded (above).
2. Run the program

   `RegSvrCE.exe` on the Windows Mobile device. While in the program, proceed as follows:

   2.1 Locate the file `ISSymbolCE.ocx` in the directory to which it has been copied.

   2.2 Select the **Register** option.

   2.3 Click on **OK**.



The Windows mobile device may not be supplied with the program "RegSvrCE.exe" as these consumer devices (i.e., smartphones and PDAs) are typically not meant to be used for this type of application. This program is required – or "VBScript.dll", which is also left off on most Windows Mobile devices – please contact the customer support.

Functionality

Open the web browser and enter the URL for one website available in the server station (e.g., `http://127.0.0.1/main.html`) into the **Address** field. At this point, the thin client executes the following process:

1. The web browser downloads the HTML page of the specified screen.
2. The web browser checks for ISSymbol control element registration on the local computer. If it cannot be found, the web browser attempts to download the ISSymbol component from the URL configured in the project (settings saved in the HTML page). Since the ISSymbol control element is properly registered on the thin client station, the web browser loads it. From this point on, ISSymbol takes over the communication with the server station and the web browser is used only as a host for ISSymbol.
3. ISSymbol connects to the data server. The IP address for the data server is configured in the project web settings (**Web settings** in the WinStudio context menu). This setting is embedded in the HTML page.
4. ISSymbol prompts a window on the thin client, requesting the **User name** and **Password**. The entered data is encoded by binary control on page 67 and sent to the server. The server station checks the validity of the data and whether the rights to open the startup screen are available. If so, the process continues. If not, an error message indicates that user name and password are invalid. In this case, the process is not continued.



Step 4 is skipped if the security system is disabled during configuration of the project.

5. ISSymbol downloads the necessary files to display the screen specified by the user (screen files, tags database, translation files, etc.).

6. ISSymbol connects to the data server and reads the value of the tags that are displayed in the specified screen.
7. ISSymbol displays the screen on the web browser and keeps updating the objects according to the values read from the server. Whenever the value of any tag displayed on the open screen(s) changes on the server, the new value is sent to the thin client (and vice-versa). Therefore, there is no polling between the thin client and the server station. This method increases the communication performance and optimizes the traffic in the network.

Please note that there are two servers in this process:

- **Web server** (HTTP server): Provides the files from the server to the thin client via HTTP protocol over TCP/IP.
- **Data server** (TCP/IP server module from "WinStudio"): Provides tag values and/or history data from the project running on the server to the thin client station(s).

Although both servers are usually running on the same computer, "WinStudio" provides the flexibility to run each server in a different station, if necessary. For further information, refer to Typical architectures for web-based applications on page 519.

15.5 Examples of Client/Server Architecture

This section describes some example architectures applied for web-based solutions and provides information on how to configure the project for each architecture. This section does not describe all possible architectures, but it provides the concepts necessary to design and configure different scenarios based on the basic architectures illustrated below

The web settings are configured in the respective configuration dialog. To open this dialog, in the WinStudio context menu, click on **Tools ▸ Web Settings**. Press the "Advanced" button to access additional settings. The following pictures illustrate these dialogs:

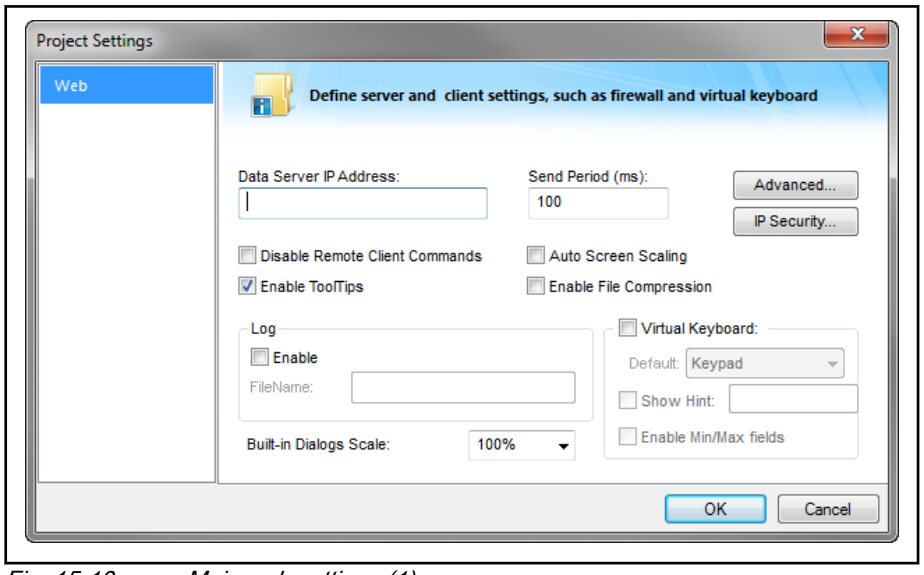


Fig. 15-10: Main web settings (1)

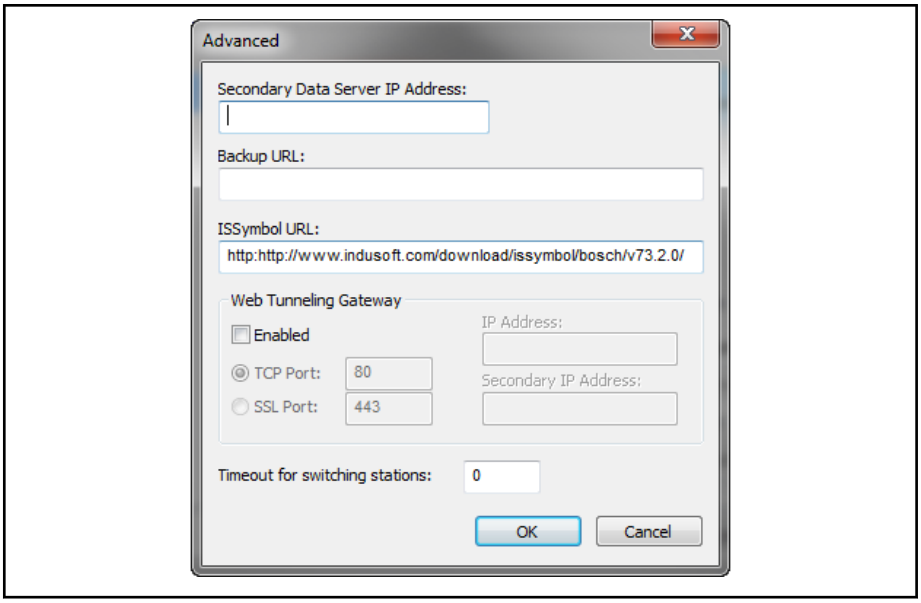


Fig. 15-11: Main web settings (2)

The following table describes the function of the main web settings illustrated in above dialogs:

Setting	Description
Data Server IP Address	If the Web Tunneling Gateway is **disabled**: The Thin Client control element (ISSymbol) uses the Data Server IP Address to connect to the "WinStudio" TCP/IP server task. If the Web Tunneling Gateway is **enabled**: The Web Tunneling Gateway uses the Data Server IP Address to connect to the "WinStudio" TCP/IP Server task.
Secondary Data Server IP Address	Same as the Data Server IP Address. However, the Secondary Data Server IP Address is only used if the connection to the Data Server IP Address fails.
Web Tunneling Gateway IP Address	The Thin Client control element (ISSymbol) uses the Web Tunneling Gateway IP Address to connect to the Web Tunneling Gateway.
Web Tunneling Gateway Secondary IP Address	Same as the Web Tunneling Gateway IP Address. However, the Web Tunneling Gateway Secondary IP Address is used only when the connection with the Web Tunneling Gateway IP Address fails.

The secondary addresses can be used in the following scenarios:

- If the Thin Clients can connect to either one of two redundant servers (web or data); or
- If the Thin Clients can connect to the server via the Intranet (LAN – Local Area Network) or via the Internet (WAN – Wide Area Network). In this case, the primary addresses are configured based on the network used more often by the Thin Clients. In the following examples, the LAN addresses are used as primary and the WAN addresses as secondary addresses.

The following table describes the meaning of some terms used in the following examples:

Term	Description
LAN	Local Area Network (e.g., Intranet)
WAN	Wide Area Network (e.g., Internet)

Term	Description
Server	Station on which the following components are running: Although "WinStudio" does not need to run on the same station as the other components, the following examples are based on that assumption. • "WinStudio" (TCP/IP Server task) • Web server (e.g., Internet Information Services from Microsoft – IIS) • Web Tunneling Gateway for IIS (if enabled)
Thin Client LAN	Thin Client station (web browser + ISSymbol control element) that connects the server via LAN.
Thin Client WAN	Thin Client station (web browser + ISSymbol control element) that connects the server via WAN.
IP_SERVER_LAN	IP Address of the server in LAN.
IP_SERVER_WAN	IP Address of the server in WAN.
IP_ROUTER_LAN	IP address of the router in LAN.
IP_ ROUTER_WAN	IP address of the router in WAN.
ScreenName	Name of the project screen, saved as HTML, that is open on the Thin Client station.

Example 1: Web server and Thin Client in the same Intranet (LAN)

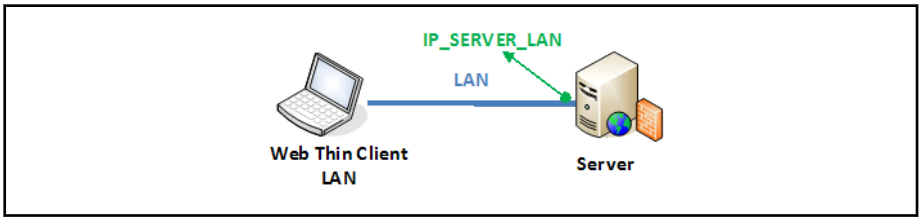


Fig. 15-12: Web server and Thin Client in the same Intranet (LAN)

This is the most common architecture and the easiest to configure. In this architecture, both the web server (e.g., Microsoft IIS) and the data server (i.e., the "WinStudio" TCP/IP server module) are running on the same PC. The Thin Client connects to the web server to download the HTML screen file(s). Afterwards, it connects to the data server to exchange data with the "WinStudio" runtime project. Since both the Thin Client and the server station are connected to the same network, the Thin Client can access the server station directly via its IP address (or host name).

Configuration:

Setting	Web Gateway enabled	Web Gateway disabled
Data Server IP Address	IP_SERVER_LAN	IP_SERVER_LAN
Secondary Data Server IP Address	-	-
Web Tunneling Gateway IP Address	IP_SERVER_LAN	-
Web Tunneling Gateway Secondary IP Address	-	-

- URL of Thin Client LAN: `http://IP_SERVER_LAN/ScreenName.html`
- Root directory of the web server (HTTP server) on the server station: "Web" subfolder of the project

Example 2: Web server with Intranet (LAN) and Internet (WAN) connections

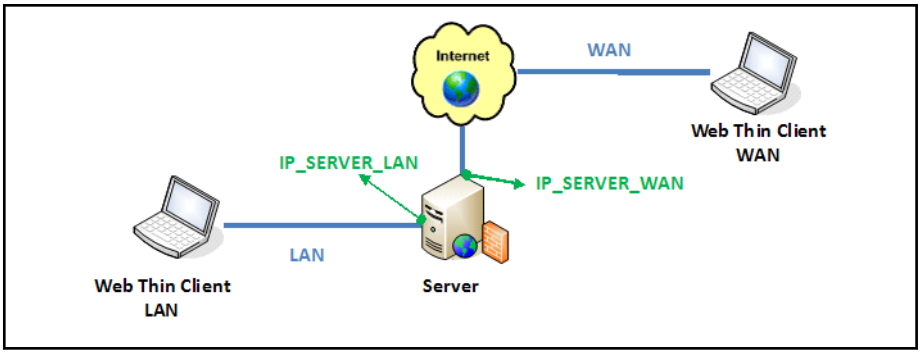


Fig. 15-13: Web server with Intranet (LAN) and Internet (WAN) connections

In this architecture, both the web server (e.g., Microsoft IIS) and the data server (i.e., the "WinStudio" TCP/IP server module) are running on the same PC. Thin Clients can connect to the server either via an Intranet (LAN) connection to the server or an Internet (WAN) connection to the server (e.g., two different Ethernet ports).

Configuration:

Setting	Web Gateway enabled	Web Gateway disabled
Data Server IP Address	IP_SERVER_LAN	IP_SERVER_LAN
Secondary Data Server IP Address	IP_SERVER_LAN	IP_SERVER_WAN
Web Tunneling Gateway IP Address	IP_SERVER_LAN	-
Web Tunneling Gateway Secondary IP Address	IP_SERVER_WAN	-

- URL of Thin Client LAN: `http://IP_SERVER_LAN/ScreenName.html`
- URL of Thin Client WAN: `http://IP_SERVER_WAN/ScreenName.html`
- Root directory of the web server (HTTP server) on the server station: "Web" subfolder of the project folder
- A fixed IP address has to be assigned to the web server on the Internet (WAN) and the project has to be running on this server. Consult the ISP provider or IT department for further information about how to get a fixed IP address for the server.

Example 3: Web server with Intranet (LAN) and router Internet (WAN) connections

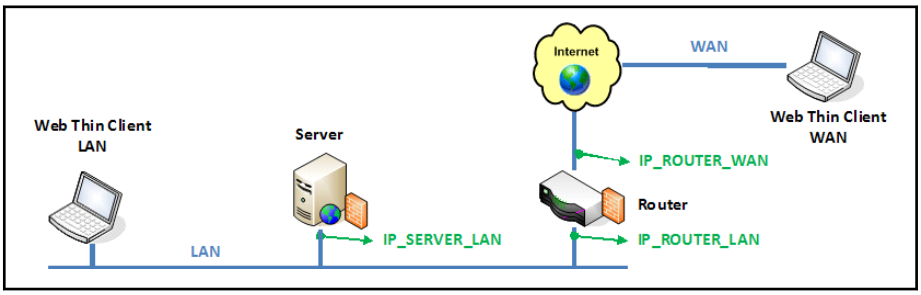


Fig. 15-14: Web server with Intranet (LAN) and router Internet (WAN) connections

In this architecture, both the web server (e.g., Microsoft IIS) and the data server (i.e., the "WinStudio" TCP/IP server module) are running on the same PC. Thin Clients can connect to the server either via an Intranet (LAN) con-

nection or an Internet (WAN) connection. There is a router between the Intranet (LAN) and the Internet (WAN).

Configuration:

Setting	Web Gateway enabled	Web Gateway disabled
Data Server IP Address	IP_SERVER_LAN	IP_SERVER_LAN
Secondary Data Server IP Address	IP_SERVER_LAN	IP_ROUTER_WAN
Web Tunneling Gateway IP Address	IP_SERVER_LAN	-
Web Tunneling Gateway Secondary IP Address	IP_ROUTER_WAN	-

☞
- URL of Thin Client LAN: `http://IP_SERVER_LAN/ScreenName.html`
- URL of Thin Client WAN: `http://IP_SERVER_WAN/ScreenName.html`
- The Router has to be configured to forward the TCP Port(s) from its public IP (IP_ROUTER_WAN) to the private server IP (IP_SERVER_LAN).

  If the Web Gateway is **enabled**, only the HTTP Port (80, by default) or the HTTPS Port (SSL Port 443, by default) have to be forwarded from IP_ROUTER_WAN to the IP_SERVER_LAN.

  If the Web Gateway is **disabled**, both the HTTP Port (80, by default) and the Studio TCP/IP Server Port (1234, by default) have to be forwarded from IP_ROUTER_WAN to the IP_SERVER_LAN. Consult the router documentation for further information about how to configure Port Forwarding on it.
- Root directory of the web server (HTTP server) on the server station: "Web" subfolder of the project folder
- A fixed IP address has to be assigned to the router on the Internet (WAN) and the project has to be running on this server. Consult the ISP provider or IT department for further information about how to get a fixed IP address for the server.

15.6 Configuring the data server

"WinStudio" has a couple of dialogs that are used for configuration of the data server and the web server. This information is embedded in the HTML websites of the respective screens. The data server is part of the "WinStudio" runtime project and uses the TCP/IP Server module. There are three (3) basic steps to this configuration:

1. Step: Configuring "Web settings" dialog

1. In the WinStudio context menu, select **Tools ▸ Web settings**.

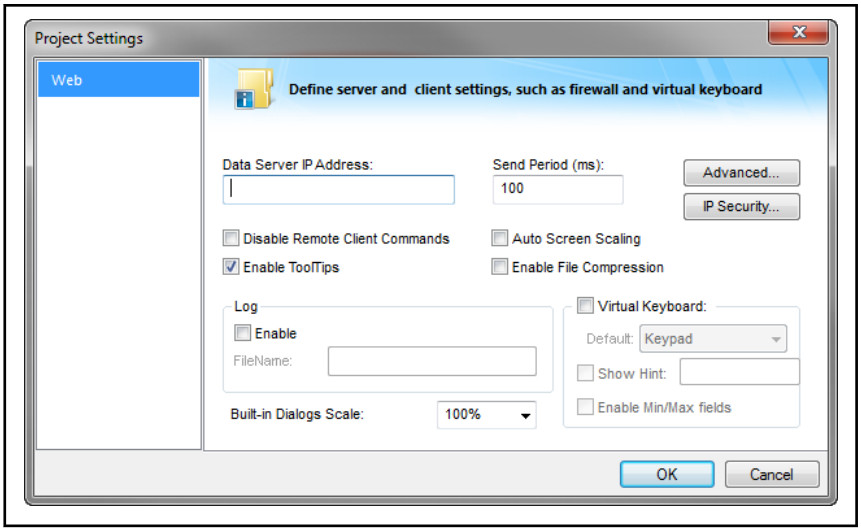


Fig. 15-15: Web settings configuration dialog

2. In the **Data server IP address** field, enter the IP address of the PC where the runtime project is to be created.
3. Ensure that the **Auto screen scaling, enable file compression** and **Enable tooltips** checkboxes are selected. If the web client is not to issue commands to the server, enable the **Disable remote client commands** checkbox.
4. If the web client is to use a virtual keyboard, enable the **Virtual keyboard** checkbox and any additional settings for the virtual keyboard at the thin client.
5. Click on the **Advanced** button to define a backup URL (i.e., backup web-site for websites) and a secondary IP address for the data server. This applies for redundant web servers and/or redundant data servers. Web tunneling can also be enabled in this dialog. The URL for the ISSymbol OCX can also be defined if it is not found on the web server.

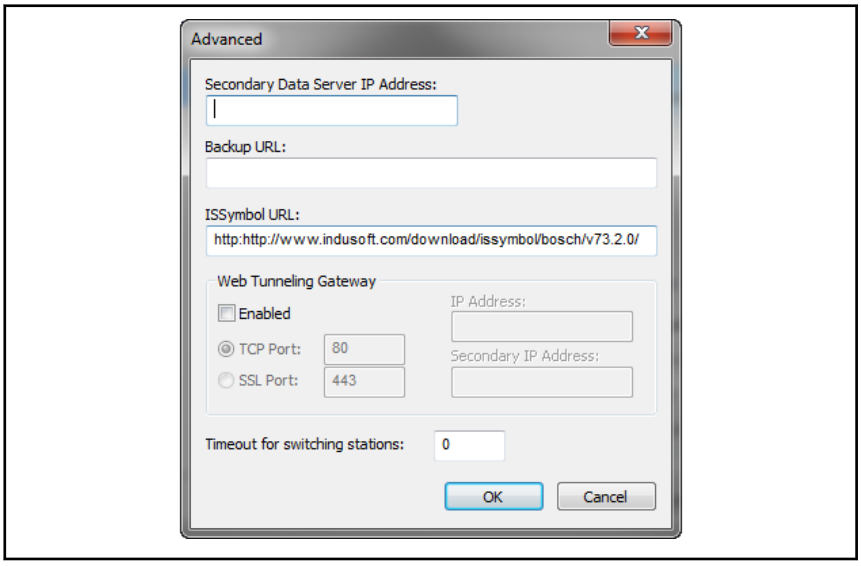


Fig. 15-16: Web settings configuration dialog, Advanced

6. Click on the **IP Security** button to define a range of IP addresses that are to be accepted by the data server. If a data request is sent from an

IP address outside of the defined ranges, the request is not acknowledged. This is an Embedded Firewall function.

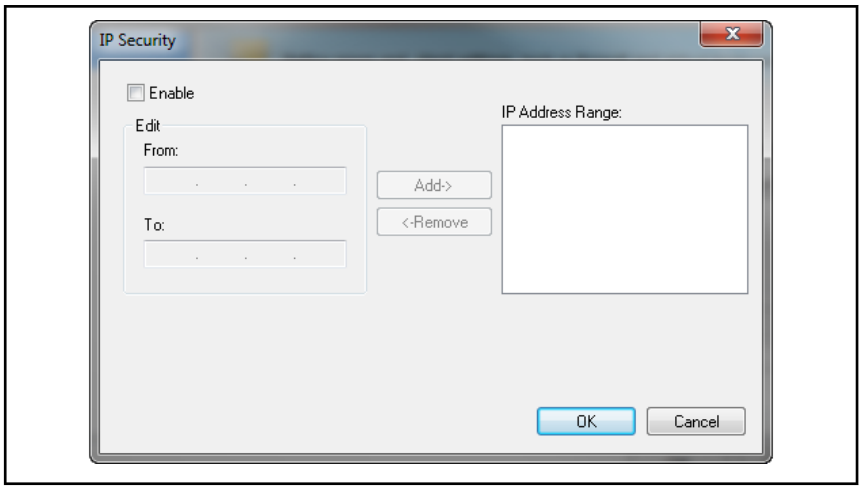


Fig. 15-17: IP Security

2. Step: Configuring "Communication settings" dialog

1. Open the WinStudio Project settings dialog and click on the **Communication** tab.

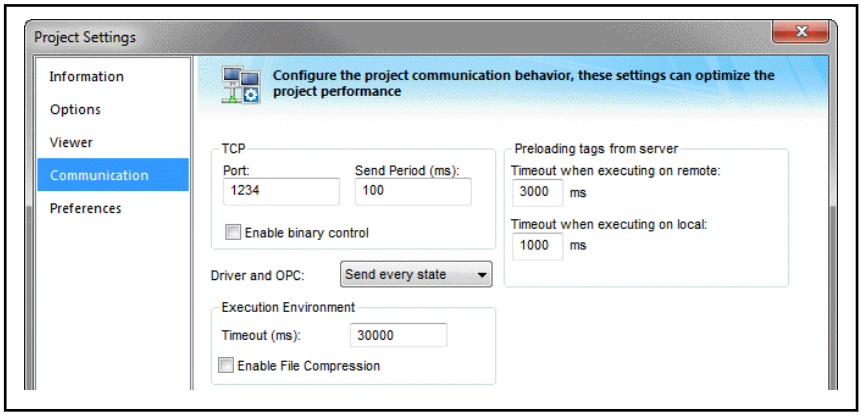


Fig. 15-18: Communication settings configuration dialog

2. Enter the port number (1234 is the default) for the data server. The data send period can also be defined (i.e., time period for updated communication of data values to the web client).
3. Optionally, enable **binary control** of the data. This option is more secure, however, slower. The default setting is disabled.

3. Step: Enabling the TCP/IP task

1. To open the task configuration, go to **Tools** ▸ **Status** in the WinStudio context menu.

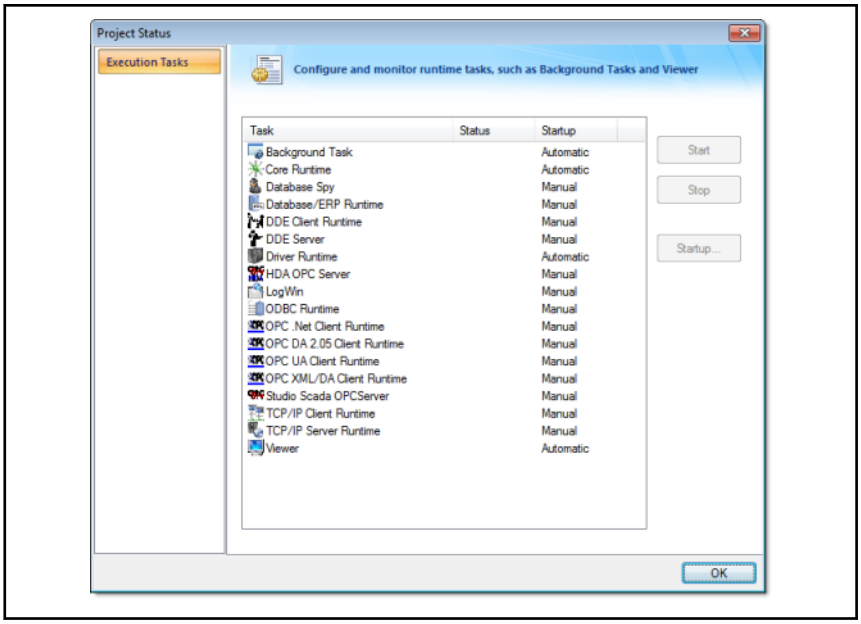


Fig. 15-19: Enable the TCP/IP task

2. Ensure that the TCP/IP server is set to “Automatic”. This setting is to be default, however, it can be configured manually by clicking on the **Start-up** button.
3. Ensure that the TCP port number is set properly (refer to the communication settings above). Otherwise, the TCP/IP server starts and stops right afterwards.



- This configuration information (from steps 1 and 2) is embedded in the websites generated in the "WinStudio" development environment when selecting **Save as HTML**.
- If any of these settings are changed, the **Verify Project** tool **HAS TO BE** run.

Using the SetWebConfig function

The integrated **SetWebConfig()** function allows the developer to configure the data server and web client via a program. The resulting configuration settings are automatically updated in the HTML files of the project (saved in the "Web" subfolder). This function always runs on the "WinStudio" server and works in Windows as well as Windows Embedded environments.

For further information, refer to the function SetWebConfig on page 804.

Configuring the web server to support IE and Secure Viewer

"WinStudio" is capable of supporting both Interned Explorer (IE)-based browsers and Secure Viewer-based browsers at the same time. However, the system has to be configured very carefully.

On the web server side

- All HTML files go into the project "Web" subfolder.
- Configure the web server (e.g., IIS) to have its root folder point to the project folder.
- If no Internet connection can be established to the "Bosch Rexroth" website and if the ISSymbol ActiveX control element is not installed on the IE-based web client, save the `ISSymbolVM.cab` file into the project folder.

On the web client side

- The Secure Viewer-based browser is to be configured to point to the .APP file in the project folder.
- The IE-based browser requires an initial URL as follows:

  `http://192.168.1.100/Web/startup.html`

  …where startup.html is the startup website.

15.7 Configuring a web server as host for project pages

As part of providing the "WinStudio" project over the web, a web server has to be configured to host the project screens.

It is not urgently required to use a Windows PC or Windows Embedded device to host the project pages. These pages are essentially static files waiting to be downloaded; runtime processing is entirely handled by the project viewer (i.e., Internet Explorer with ISSymbol on page 516 or Secure Viewer) on the thin client. As such, all standard web servers can be used on any computer platform as host for pages.

For example, if a Unix-based intranet server is already available, the project "Web" subfolder (or whatever folder in which the project pages have been saved) can be copied to the server to have the thin client point to that server address.

However, please note that the selected web server may not be robust enough to serve as server for the project in a production environment and/or it may not support all "Rexroth WinStudio" functions. If these functions are required, Microsoft IIS is to be used in most cases as described below. In detail:

- To support <Mobile access> the server “Active Server Pages” (ASP) and ASP.NET have to be processed and
- To support web tunneling gateway on page 530 (WTG), the server has to allow Internet Service API (ISAPI) extensions.

Before installation and configuration of any software, please review its documentation carefully.



Web servers typically work or "listen" on the TCP/IP port 80 of a computer. Only one running process can receive a defined port. Respectively, if other processes, e.g., SCADA third-party software, is already receiving on port 80, this process might collide with the server process. Either, one of the processes has to be configured to receive from a different port or the Task Manager has to be used to end the conflicting process. If the conflicting process cannot be identified, open the Windows command prompt and enter the following command to get a list of all networking processes:

```
netstat -a -o
```

NTWebServer and CEWebServer

NTWebServers and CEWebServers are lightweight, zero-configuration web servers that are integrated in "Rexroth WinStudio" for free. They can be configured to demonstrate the project and run basic tests without making the financial and technical investment in a full-featured web server. However, they are not robust and do not support SMA or WTG. For proper applications, it is recommended to use Microsoft IIS as described below.

The NTWebServer can be executed on any supported on page 21 Windows PC. It can be found in the "WinStudio" program directory under

`[…]\WinStudio\Bin\NTWebServer.exe`.

Copy it to the "Web" subfolder of the project before running it. The server has to be located in the same folder as the subsequently used pages. After running it, this folder becomes the top level (or "home" directory) of the website.

CEWebServers can be executed on any supported on page 21 Windows Embedded device. It can be found in the "WinStudio" program directory under

`[…]\WinStudio\Redist\CE_version\CE_processor\CEWebServer.exe`.

Assuming the "Web" subfolder of the project was copied to the non-volatile memory of the device, copy `CEWebServer.exe` to the same folder and execute it. Again, this folder becomes the top level of the website.



Both NTWebServers and CEWebServers have to run as normal programs as they cannot run as Windows services.

Microsoft IIS

Internet Information Services (IIS) is the full-featured server software that is bundled with Windows Server and "professional" versions of Windows:

IIS versions

Windows version	IIS version	Notes
Windows XP Professional	IIS 5.1	Maximum of 10 simultaneous connections
Windows Server 2003 Windows XP Professional x64 Edition	IIS 6.0	No limit on connections
Windows Server 2008 Windows Vista (most versions)	IIS 7.0	No limit on connections
Windows Server 2008 R2 Windows 7 (all versions)	IIS 7.5	No limit on connections
Windows CE and Windows Mobile 5.0 and later	IIS for CE .NET	Has to be integrated into the Platform Builder. It can be up to 10 simultaneous connections by default.

IIS supports all features of "Rexroth WinStudio" and is robust enough to serve almost any "WinStudio" project in a production environment. This web server is recommended for most users. However, to properly install and configure it, experience in Windows network administration is required.

For reasons of system security, IIS is turned off by default if the operating system is installed. To enable it, go to "Windows features" dialog (in Windows 7, **Control Panel ▸ Programs ▸ Programs and Features ▸ Turn Windows features on or off**):

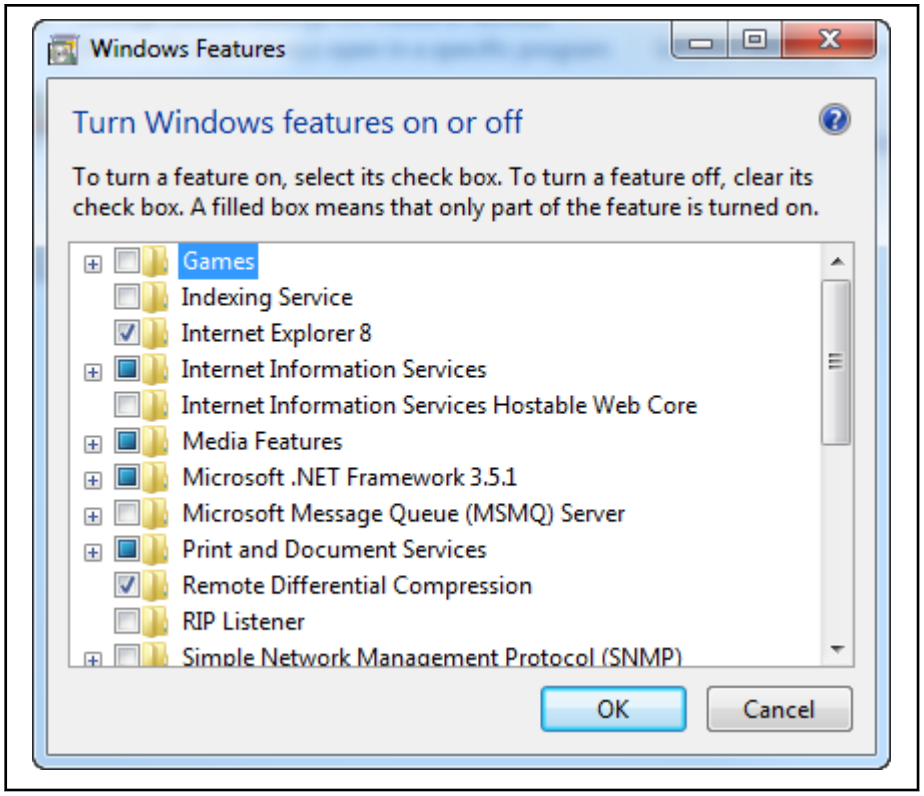


Fig. 15-20: Turning on IIS in the "Windows features" dialog

If SMA and/or WTG is to be used in the "WinStudio" project, Ensure that the ASP and ISAPI extensions are also activated:

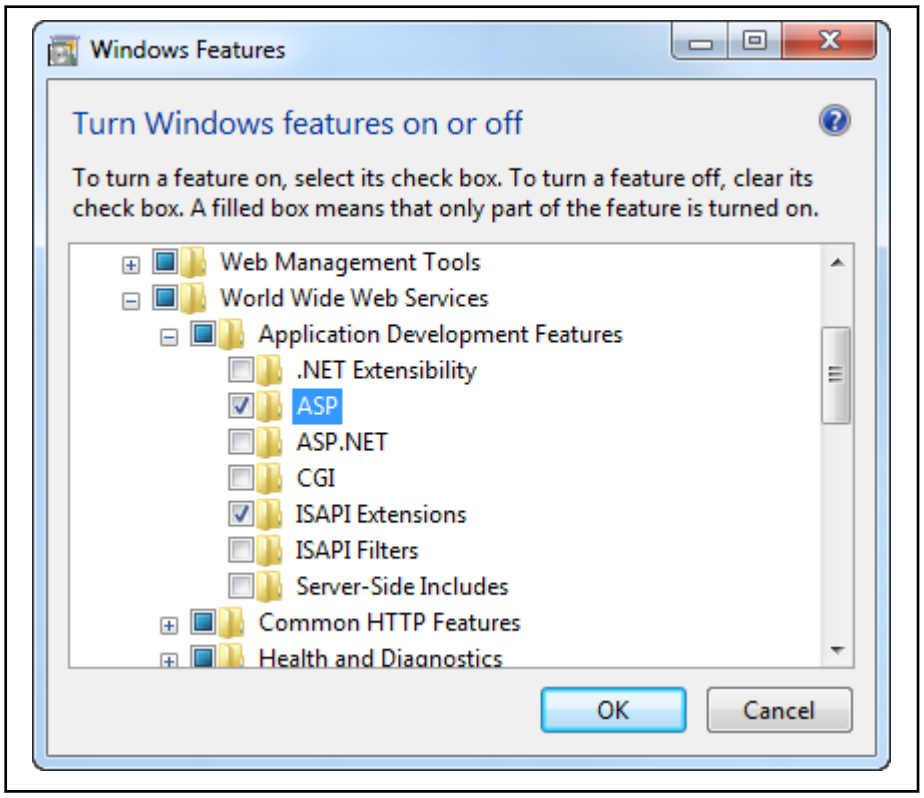


Fig. 15-21: Activating ASP and ISAPI extensions in the "Windows features" dialog

When IIS is enabled, the administrative tools (in Windows 7, **Control Panel ▸ System and Security ▸ Administrative Tools**) can be used for configuration. For further information, please refer to the detailed documentation provided by Microsoft.

Apache for Windows

If IIS is not available or if it is not to be used, the second most popular web server for Windows is the open-source server Apache. However, this server requires even more expertise than IIS for proper installation and configuration. Thus, carefully read the respective documentation before using this type of server.

15.8 Installing the web tunneling gateway

The "web tunneling gateway" ("WTG", also referred to as "IndWebSvc") enables Internet-connected computers – typically the web server – to route project data between remote thin clients and a data server that is "not" connected to the Internet. This section describes the installation process of WTG.

Before you proceed with this task, you should first set up Microsoft IIS as a web server to host your project page. WTG can only run on Microsoft IIS. Also, the web server has to be connected to the Internet and requires a static IP address instead of a dynamically assigned IP address. For information on how to obtain a static IP address, please contact the responsible IT department or internet service provider. Finally, the web server and data server both have to be on the same internal TCP/IP network, even if only the web server is connected to the Internet.

To enable thin clients access to the project, they have to be able to communicate with both the web server hosting the project pages and the data server actually running the "WinStudio" project. This is not a problem if servers and clients are all on the same internal network or if both of the servers are connected to the Internet to enable remote thin clients to communicate with them.

However, if the data server is not to be connected to the Internet (for network security or topology reasons), the project data has to be routed through another computer that is connected.

WTG provides these routing capabilities. It is an ISAPI extension for Microsoft IIS and typically installed on the same web server hosting the project pages. It encapsulates the data sent between the data server and thin clients and ensures project security. Additionally, it can route data simultaneously between multiple data servers.

These instructions apply for:

- Microsoft IIS 7.0 running on Windows Vista and Windows Server 2008; and
- Microsoft IIS 7.5 running on Windows 7 and Windows Server 2008 R2.

1. On the web server (or the Internet-connected computer acting as gateway), ensure that Microsoft IIS is running and ISAPI Extensions are enabled. For further information, refer to Configuring a web server as host for project pages on page 527.
2. Copy the WTG installer from the "WinStudio" application directory to the "root" directory of the website (e.g., `C:\inetpub\wwwroot\`).

   The installer is located at

   `..\WinStudio\Bin\WebTunnelingGateway.exe`
3. Execute the installer and follow the steps of the installation wizard. Options cannot be changed. The installer creates a new directory in the

"root" directory of the website and installs the WTG extension files in that directory. It also creates a new application pool.

4. Open the IIS Manager. The "Internet Information Services (IIS) Manager" window is displayed.
   4.1 Click **Start** and then **Control Panel**.
   4.2 In the Control Panel window, proceed as follows.
   - Under Windows Server 2008, click on **Administrative Tools**.
   - Under Windows Vista, click on **System and Maintenance** and then **Administrative Tools**.
   - Under Windows 7 or Windows Server 2008 R2, click on **System and Security** and then **Administrative Tools**.

   4.3 In the Administrative Tools window, proceed as follows.
   - Under Windows Vista or Windows 7, double-click on **Internet Information Services (IIS) Manager**.
   - Under Windows Server 2008 or Windows Server 2008 R2, double-click on **Server Manager** and click on **Internet Information Services (IIS) Manager** in the Server Manager window.

   The Internet Information Services (IIS) Manager window is displayed.
5. Ensure that the application pool has been created by the installer. If "IndWebSvcPool" is not in the list, the WTG installer did not finish correctly. Return to the beginning of this procedure.
   5.1 In the **Connections** pane, expand the server node and click on Application Pools.
   5.2 On the Application Pools page, in the list of pools, look for IndWebSvcPool.

   If IndWebSvcPool is not in the list, then WTG installer did not finish correctly. Return to the beginning of this procedure.
6. Add an ISAPI restriction for IndWebSvc.
   6.1 In the **Connections** pane, click on the server node.
   6.2 On the **Home** page of the server node, double-click on **ISAPI and CGI Restrictions**.
   6.3 On the **ISAPI and CGI Restrictions** page, in the **Actions** pane, click on **Add**.

   The Add ISAPI or CGI Restriction dialog is displayed.
   6.4 In the **ISAPI or CGI path** field, enter the path of the file `IndWebSvc.dll`

   (e.g., `C:\inetpub\wwwroot \indwebsvc\IndWebSvc.dll`), or click on the browse button (…) and navigate to the file.
   6.5 In the **Description** field, enter IndWebSvc.
   6.6 Select **Allow extension path to execute**.
   6.7 Click on **OK**.
7. Configure the handler mappings of the website to make ISAPI extension files executable.

7.1 In the **Connections** pane, expand **Sites** and click on the website that is acting as gateway (i.e., typically **Default website**, or any site hosting the WinStudio project pages).

7.2 On the **Home** page of the website, double-click on **Handler mappings**.

7.3 On the **Handler mappings** page, select **ISAPI-dll** in the list of mappings.

This item is disabled by default.

7.4 In the Actions pane, click on **Edit feature permissions**.

The “Edit feature permissions” dialog is displayed.

7.5 Select **Execute**.

7.6 Click on **OK**.

8. Ensure that the website and the gateway extension are running and start Internet Explorer on the server and go to http://127.0.0.1/indwebsvc/indwebsvc.dll

   `127.0.0.1` is the "localhost" or "loopback" address, i.e., the browser is displaying the website that is running on the same computer.

   The resulting website generally looks as follows:

   ```
   Studio Web Gateway
   Status: Ok
   Version: 1.3
   Active connections: 0
   ```

   If this is not displayed, either the website is not running or the gateway extension was not correctly configured. Return to the beginning of this procedure.
9. Close the IIS Manager.
10. Configure the "WinStudio" project to include the address of the gateway.

    For further information, refer to Project settings: Web on page 71.

15.9 Configuring the thin client

"WinStudio" supports two types of web browsers to use in combination with thin clients: Microsoft Internet Explorer, supporting ActiveX control elements and Secure Viewer developed by "Bosch Rexroth". Both browsers can be used simultaneously with the web server and the choice of browser depends on the project requirements. The Secure Viewer is designed for a production environment, while Internet Explorer is generally more appropriate for desktop applications or where thin clients need to connect to different project servers.

Configuring Microsoft Internet Explorer

The configuration of Microsoft Internet Explorer to use as web browser on a thin client includes only a few steps. The most important elements include:

- Ensure that ActiveX control elements are enabled. In general, go to **Tools** ▸ **Internet options** ▸ **Security** for the required settings.

  Click on the **Custom level** button and ensure that the ISSymbol ActiveX control can be installed. Afterwards, reset a higher security level. (Otherwise, be sure to install the ISSymbol ActiveX control manually).
- Ensure that VBScript is supported on the PC. This is generally not an issue with Desktop PCs. However, it can be an problem on Windows

Embedded devices. It has to be ensured that the VBRun.dll is installed and registered on the PC.

If Microsoft Internet Explorer is to be used as web browser for thin client projects, individual navigation bars and buttons can be removed to reduce the amount of space taken up by these items.

On the desktop, an icon can be created as shortcut to the startup website on the web server. Click this icon to access the website directly. This shortcut can also be placed in the Startup folder to start the website automatically whenever the browser is started.

Installing Secure Viewer

Installation on a Windows PC

Secure Viewer has to be started from CD (or downloaded from the "Bosch Rexroth" website). Proceed as follows:

1. Request the Secure Viewer installer from the software vendor, save it to the designated computer and start it.
2. Follow the instructions of the installation wizard. There are only two settings that have to be configured during installation:
   - **URL**: Enter the URL or file path of the project file (**project_name.app**) on the web server.
   - **Server IP**: Enter the IP address or hostname of the data server (TCP/IP Server).
3. Finish the installation by clicking on **Finish** to close the installer.

The installation wizard automatically installs and registers the thin client Active X component (ISSymbol.ocx). If the **URL** and **Server IP** settings are configured correctly, Secure Viewer is ready to start.

☞ To increase security of the runtime environment, add a Secure Viewer shortcut (Viewer.exe) to the Startup directory in Windows. Viewer.exe is installed into the \Bin subfolder where the "WinStudio" program is installed.

Installing on Windows Embedded

By copying the necessary files to the non-volatile memory of the device, Secure Viewer can also be installed on a Windows Embedded device. To install Secure Viewer:

1. Determine the OS version and processor type of the Windows Embedded device and browse to the corresponding subfolder in the "WinStudio" program directory. Example: If Windows CE 5.0 is running on a MIPS processor, browse to the `[…]\WinStudio\Redist\WinCE 5.0\MIPS\Bin` subfolder.
2. *Select the following files in the subfolder:*
   - IndVkStd.dll
   - ISSymbolCE.ocx
   - Tagi.bin
   - Viewer.exe
   - ViewerCfg.exe
3. Copy the files to the non-volatile memory on the Windows Embedded device.
4. Register the ISSymbol control on the Windows Embedded device by executing the following command in the **Command prompt** window:

```
regsvrce.exe "\ISSymbolPath \ISSymbolCE.ocx"
```

Example:

```
regsvrce.exe "\Storage Card\ISSymbolCE.ocx"
```

5. Save the registry settings to keep `ISSymbolCE.ocx` registered when rebooting the Windows Embedded device.

Check the documentation of the device manufacturer for information on how to save the registry settings.

6. Start Secure Viewer by running `Viewer.exe`.

Changing the Secure Viewer configuration

After initial installation, the configuration of Secure Viewer is saved in the **viewer.ini** file that is to be in the same directory as **Viewer.exe**. (This is to be in the `\Bin` subfolder where the Secure Viewer program is installed). There are two options to change the configuration of Secure Viewer after it has been installed. First, the also installed viewer configuration utility can be run (**ViewerCfg.exe**).

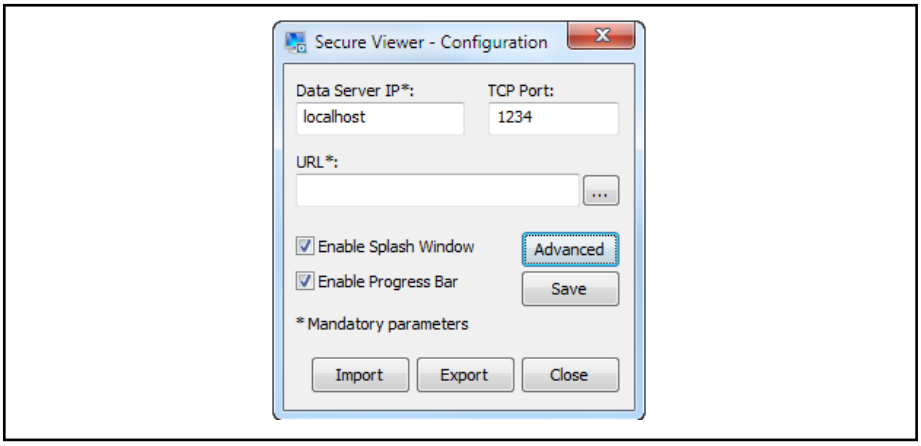


Fig. 15-22: Dialog ViewerCfg

On Windows Vista and Windows 7, administrator rights are required to properly run the Secure Viewer configuration utility. It is generally installed with these rights by default. In case of any further problems, check the file properties of "ViewerCfg.exe" and ensure that **Run as administrator** is selected.

The configuration utility provides the following options:

- **Save** button: Immediately saves the current settings to a "viewer.ini" in the same directory as the "ViewerCfg.exe" utility.
- **Import** button: Loads a "viewer.ini" file from another directory.
- **Export** button: Saves the "viewer.ini" file to another directory.
- **Data server IP** field: Enter the IP address (or host name) of the data server station. The data server station is the computer or device on which the TCP/IP Server module is running.
- **TCP port** field: Enter the port number of the data server, if it is different than the default port of 1234.
- **URL** field: Enter the URL or file path of the project file ("*.app") on the web server.

- **Enable splash window** option: Check (enable) this option to display a splash window when starting Secure Viewer.
- **Enable progress bar** option: Enable this option to display a progress bar while the project file is loaded by Secure Viewer.
- **Advanced** button: Click to access additional configuration options:

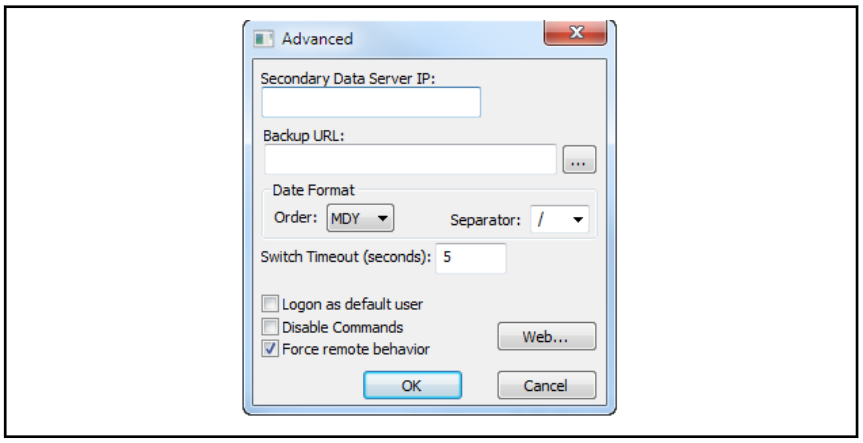


Fig. 15-23: Secure Viewer Configuration utility – Advanced

  - **Secondary data server IP** field: Enter the IP address (or host name) of the secondary data server station. If the primary data server fails, Secure Viewer automatically attempts to connect to the secondary data server.
  - **Backup URL** field: Enter the URL of the backup location of the project file.
  - **Date format**: Adjust the date format on the Secure Viewer station to match the data server. This is to maintain compatibility with any Alarm/Event Control or Trend Control objects in the project.
  - **Switch timeout** field: Enter the number of seconds that Secure Viewer is to wait before attempting to connect to the secondary server if the connection to the primary server is interrupted. (For more information, refer to:Web tab on page 71.)
  - **Log on as default user** option: Enable this option to have Secure Viewer automatically log on as Guest, not requiring a username or password.
  - **Disable Commands** option: Enable this option to prevent Secure Viewer from sending user commands back to the server. Only current information from the server are displayed.
  - **Web** button: If a Web Tunneling Gateway has been configured to connect the Intranet to the Internet, the gateway addresses are to be entered here.

The second option to change the configuration of Secure Viewer is to manually edit the **viewer.ini** file with a text editor. The structure of the file is as follows:

File structure

```
[Options]
nosplash=           // Enable (0) or disable (1) the splash window
noprogressbar=      // Enable (0) or disable (1) the progress bar
ds1=                // Data Server Primary
ds2=                // Data Server Secondary
dsp=                // Data Server Port
wtg1=               // Web Tunneling Gateway Primary
wtg2=               // Web Tunneling Gateway Secondary
```

```
url=                // URL from project file (*.app)
proxyip=            // Proxy Address
proxyp=             // Proxy Port
ceemul=             // Enable (1) or disable (0) CEView emulation
```

Example

PC with "WinStudio" runtime project (has both web server (NTWebServer or IIS) and data server):

- IP Address is 192.168.1.106
- Project name is SecureViewerTest
- Project file is SecureViewerTest.app

Secure Viewer configuration dialog:

- Data Server IP = 192.168.1.106
- TCP Port = 1234
- URL = http://192.168.1.106//SecureViewerTest.app

Alternatively, the viewer.ini file could be edited as follows:

viewer.ini

```
[Options]
url=http://192.168.1.106//SecureViewerTest.app
noprogressbar=1
ds1=192.168.1.106
nosplash=1
ds2=
dsp=1234
wtg1=
wtg2=
user=Guest

[OEM]
Splash=Splash.bmp //this is a splash screen
```



- Your "Rexroth WinStudio" runtime license has to support a Secure Viewer (thin client). Otherwise, the connection to the data server is rejected.
- Ensure that NTWebServer is stored in the project folder (not in the "Web" subfolder) or point IIS to the project folder.
- When configuring your web server, ensure that the secure connections are supported via TLS/SSL (i.e., the URL starts with "https" instead of "http") and that a self-signed certificate was created. This certificate has to be installed manually on every computer that uses the Secure Viewer as trusted Root CA to access the project. The reason is that the Secure Viewer is running in restricted mode and the user cannot accept a self-signed certificate provided by the web server.

  For more information, refer to technet.microsoft.com/library/cc725849.aspx

15.10 Customization of the “Viewer” program icon in the task bar

Customize your “Viewer” program icon shown in the Windows task bar easily by copying your customized icon to the program folder.

This customization is possible for the local “Viewer” program installed on the project runtime server (e.g. Rexroth WinStudio, EmbeddedView, CEView) as well as for the separate WinStudio thin client program (also Secure Viewer) that can be installed on remote stations. It is mainly used for the latter, as the WinStudio thin client is the program mostly seen by the user.

It is possible to customize the “Viewer” program if, for example, you are a system integrator and you want to provide a trademark-protected integrated solution to your client. Customizing the icon approve a better appearance.

This task requires that the customized icon you intend to use, already exists. It should be a default Windows ICO file (`*.ico`).

To customize the icon of the “Viewer” program, copy the ICO file to the folder containing the “Viewer” program (`Viewer.exe`) and rename the ICO file to `Viewer.ico`.

The memory location of the “Viewer” program varies:

- For the WinStudio Thin Client running on Windows, go to:

  `C:\Program Files\Rexroth\WinStudio Thin Client v7.4\BBBE2E0F-084D-484b-AFDF-EA12BF0E52FF\Viewer.exe`

- For the Rexroth WinStudio running on Windows, go to:

  `C:\Program Files\Rexroth\WinStudio\Bin\Viewer.exe`

- For EmbeddedView running on Windows and Embedded Standard and CEView running on Windows Embedded Compact, it is located in the folder to which you copied the utility “Remote Agent” (`CEServer.exe`) and installed the system files.

  For more information, refer to Installing EmbeddedView or CEView on a target device on page 27.

When executing the “Viewer” program again, it automatically has the customized icon and displays it in the task bar.

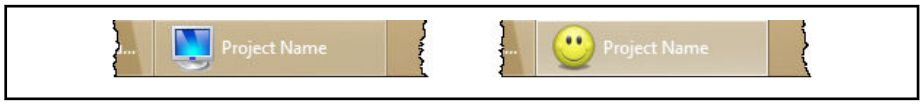


Fig. 15-24: Customizing the icon: Before and afterwards

15.11 Implementing security for web-based applications

There are various methods for implementing security of web-based applications. The required approach may depend on a number of factors and may involve one or more methods of implementing security.

Method 1: Password protection

In "WinStudio" groups of users and individual users within a group can be created. Each group (e.g., operators, supervisors, maintenance) can have different security levels and access different levels of functionality. Individual passwords can be configured for each user.

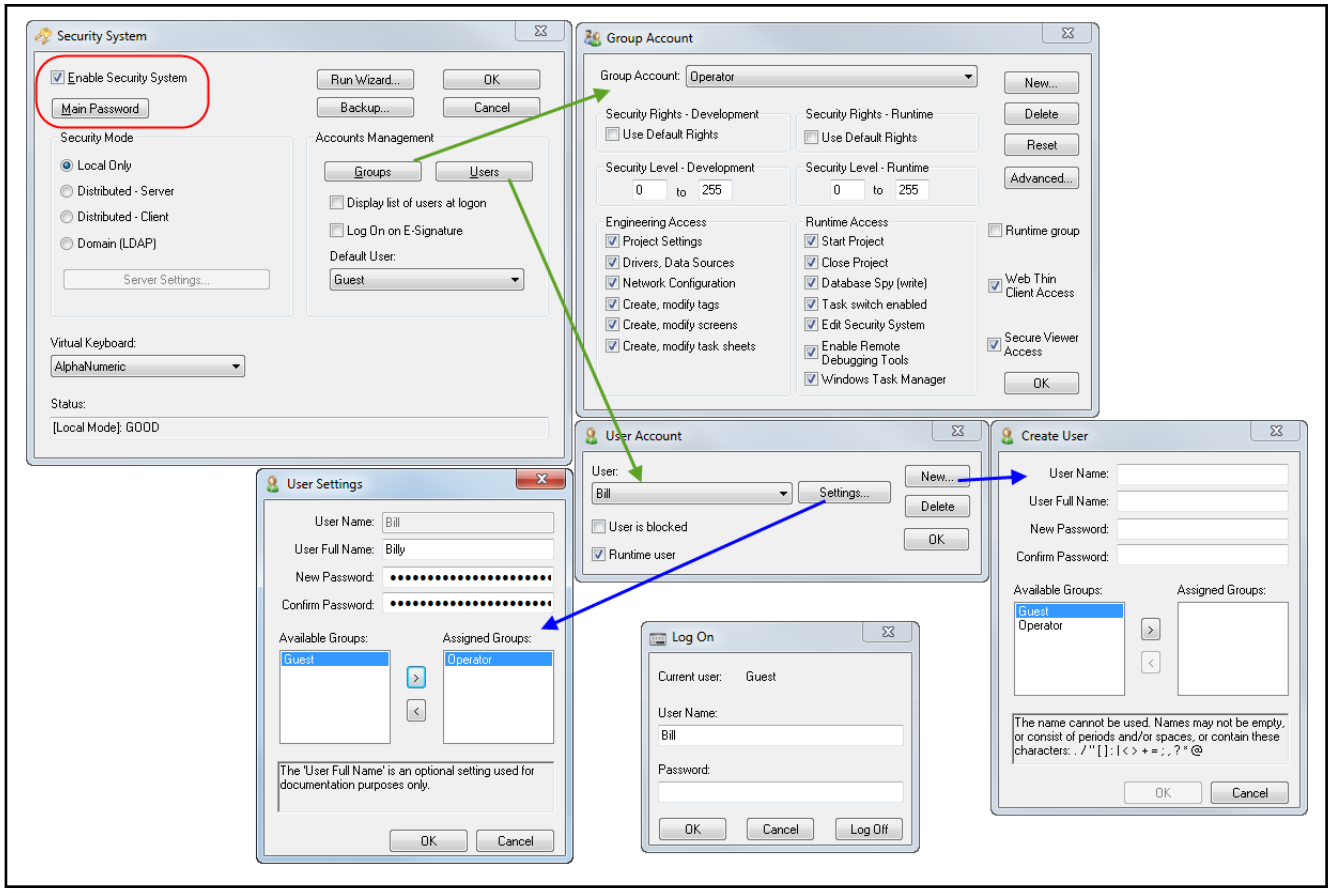


Fig. 15-25: Security groups and users

In addition, groups can be assigned advanced settings, allowing features like minimum password size, password aging, e-signature on objects with command animations, account auto-lockup (e.g., lock up after a number of invalid attempts to access) and user account blocking (temporarily disable – e.g., when employee is on vacation).

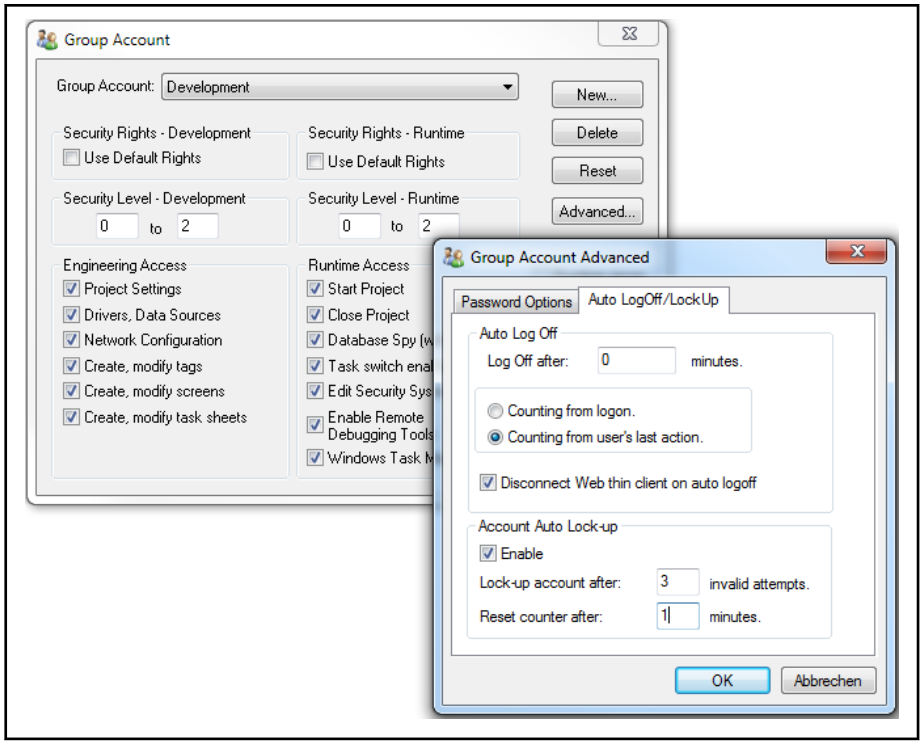


Fig. 15-26: Account auto-lockup and user account blocking

If system security is enabled, these password protection functions are also available at the thin client station. Users at thin client stations attempting to connect to the web server are required to enter a User Name and Password. If these are invalid, the user cannot access the system.

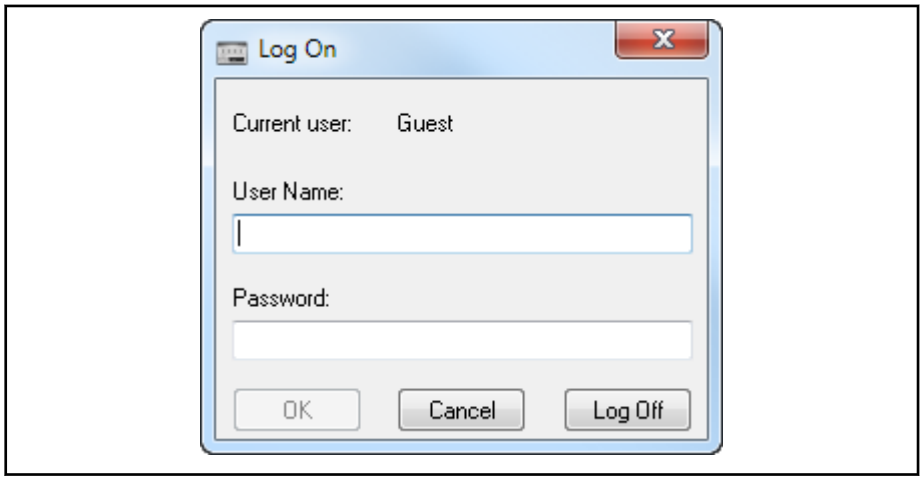


Fig. 15-27: "Log on" dialog

Within a project, the various screen objects and animations, as well as screen access can be assigned a security level. The current user logged on requires an access level range corresponding to the desired object or screen. The following is a representative method of assigning security access levels by group.

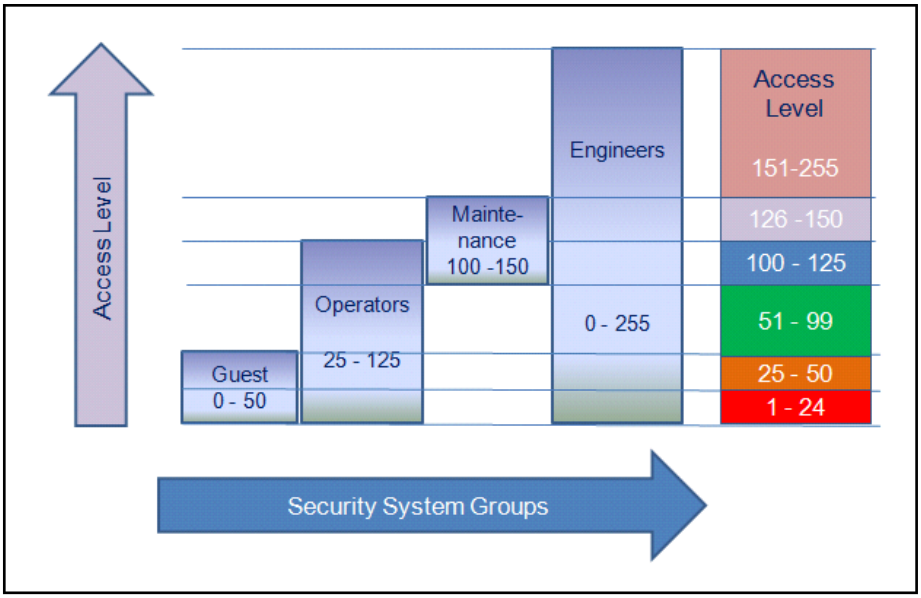


Fig. 15-28: Screen access

For further information, refer to Security on page 461.

Method 2: Disabling thin client commands

"WinStudio" allows bi-directional data exchange between thin client and data server. However, for security reasons it may be advantageous to only allow the thin client to view the process or machine data and not send any data back to the data server.

Enable (check) the **Disable remote client commands** option in the Project Settings (in the WinStudio context menu) to ensure that all commands from thin client stations are blocked. The communication becomes unidirectional (from server to thin clients):

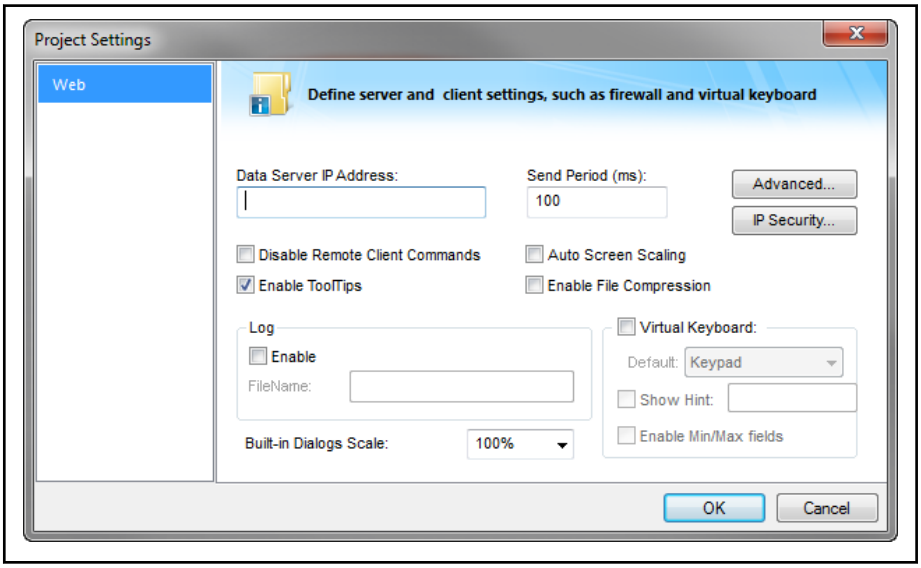


Fig. 15-29: Web Settings dialog

Method 3: Embedded firewall

This function enables filtering the access to the project based on the IP address of the thin clients. If a thin client attempts to connect to the server station, the server checks if the IP address of the thin client station is authorized to access the project. The ranges of authorized IP addresses can be configured by clicking on **IP security** in the Web settings.

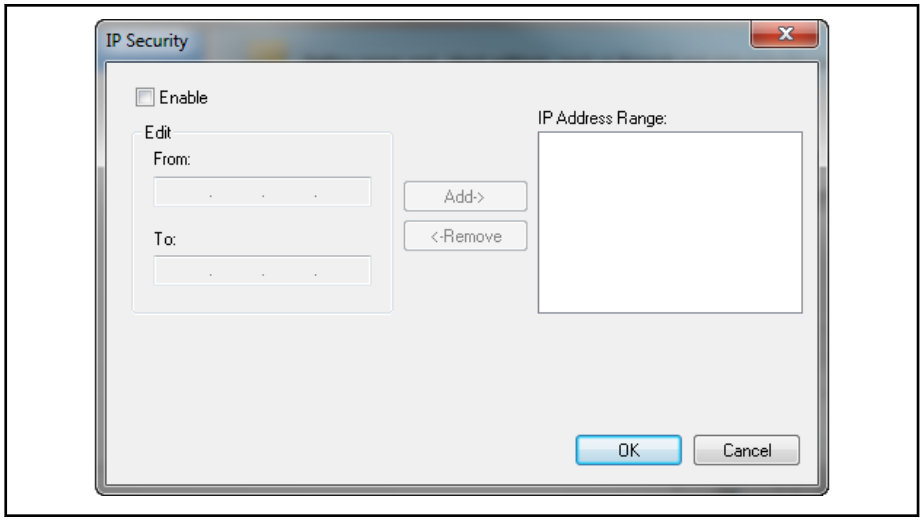


Fig. 15-30: IP Security dialog

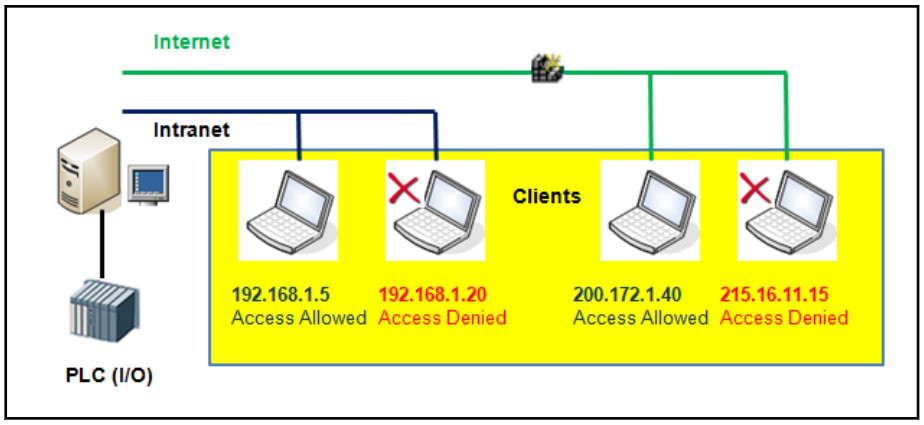


Fig. 15-31: Access allowed by IP address

Method 1: Encrypted communications (SSL)

By enabling the Web Tunneling Gateway (WTG), the entire communication between data server + web server and thin client can be encoded using RC6, a highly-secure 128-bit encryption standard. To use SSL, proceed as follows:

1. In the "Web settings" dialog, click on **Advanced**. Enable the **Web tunneling gateway enabled** option. Click on the **SSL** radio button and ensure that the SSL port is set to 443. Click on **OK**.

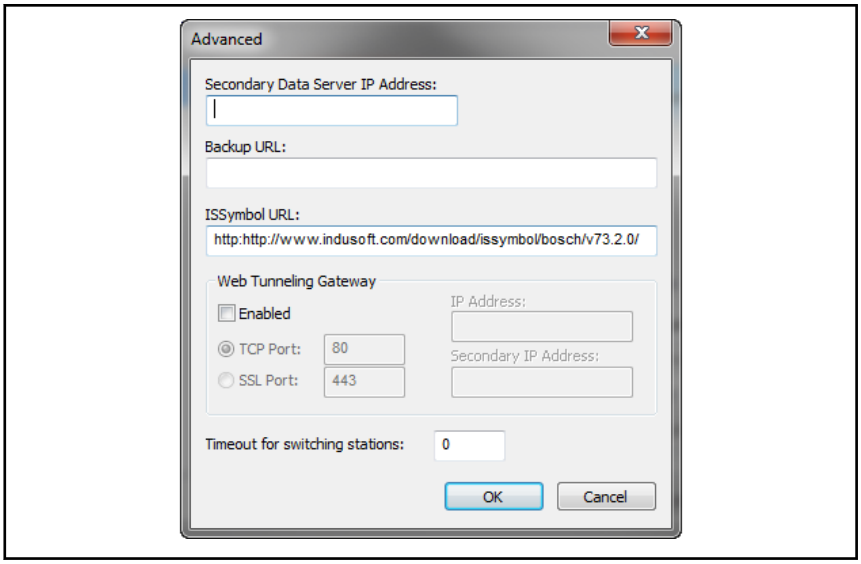


Fig. 15-32: Web Settings – Advanced dialog

2. Ensure that the SSL functions are enabled on the web server and that a SSL certificate of authentication is available.
3. Ensure that SSL is enabled in the web client.
4. Set up all additional web configurations to support the WTG.

Method 5: VPN

A VPN is a Virtual Private Network. This type of network is referred to as virtual as it actually uses the public Internet to transport data from one computer to another. However, since this type of network is encoded and uses other security mechanisms enabled by the ISP, is it a very secure private network. While VPNs are inherently secure, they are more costly than a simple public Internet connection.

15.12 Port Usage

There are various ports that are used by "WinStudio" and/or related software. These are:

Port #	Program
20	FTP Server (Data)
21	FTP Server (Command)
25	SMTP Server
80	Microsoft IIS Server for HTTP packets
110	POP3
118	Microsoft SQL Server Services
161	SNMP
162	SNMP Trap
389	LDAP
443	Microsoft IIS Server for HTTPS packets (SSL)
502	Modbus TCP/IP protocol
663	LDAP over SSL
1028	FTP Client (Command)

Port #	Program
1029	FTP Client (Data)
1234	Project TCP/IP Server
1443	Microsoft SQL Server
1444	Microsoft SQL Server default port (Monitor)
1521	Oracle
1526	Oracle
2030	Oracle
3001	A-B Ethernet TCP/IP Protocol (default)
3306	MySQL (can be configured to use 3306-3309)
3872	Oracle Management Remote Agent
3997	Studio ADO Gateway
4322	Remote Agent (CEServer)
5432	PostgreSQL
47808	BACNet UDP Protocol (default)

You may need to accommodate one or more of these port's usage in your Firewall settings.

15.13 Example: Configuring a WinStudio web application

To view your project as web application, proceed as follows:

1. Expand the "WinStudio" folder and double-click on the **main.scr** screen.
2. To save the screen in HTML format, click on **Save as HTML** in the context menu of the screen.

   Web files are saved in the "Web" folder.

   Thus, open the folder and check whether the "main" image was successfully saved. The file **main.html** is displayed.
3. Open the "Web settings" dialog.

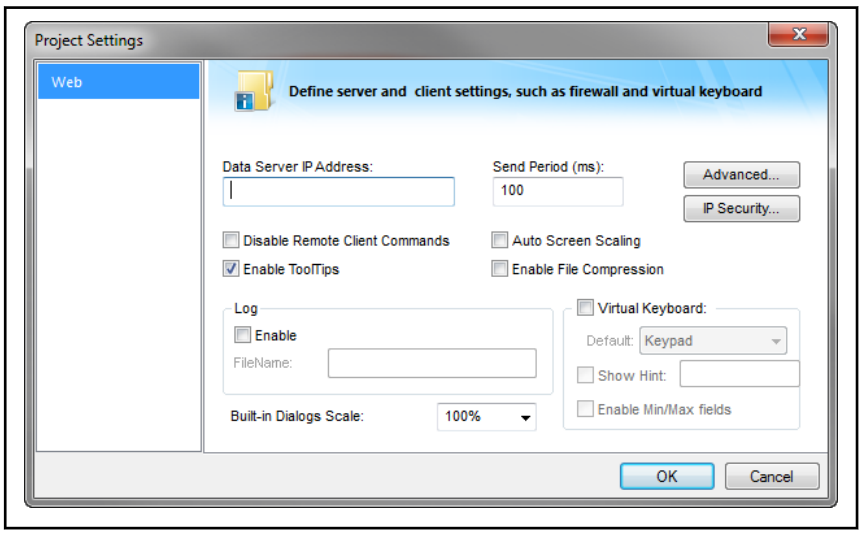


Fig. 15-33: Opening the "Web settings" dialog

4. Configure the **Data server IP address** field to use the IP address of the server station (computer on which the program is running). The thin client station exchanges online data (tag values) with the station specified in this field.
5. Enter the URL path of the file **main.html** (in the "Web" folder) into the **URL** field.

   The URL depends on the Home directory configured in the web server of the server station.

☞ Microsoft provides web servers for any Microsoft operating system. For information on the installation and configuration of web servers, refer to the Microsoft documentation.

6. After configuring the web settings, click on **OK** to close the dialog.
7. Save and close all images and worksheets and verify the project (**Verify application** in the WinStudio context menu) to update the web settings in all WinStudio images and worksheets.

NOTICE **The project has to be verified again after each change to the web settings.**

To test the web-based application, proceed as follows:

1. Click on **Start application** (in the "WinStudio" context menu), to execute the project locally on the server station.
2. Open an Internet browser (Microsoft Internet Explorer or Netscape) and enter the URL to open the main.html screen from the server station.
3. If the "Log On" dialog is displayed in the browser, enter **Guest** into the **User name** field and click on **OK** to open the main.html screen in the browser.

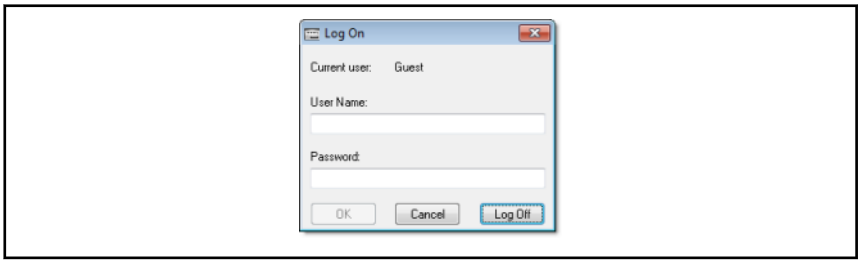


Fig. 15-34: "Log on" dialog

Note that the variable values can be changed locally (server station) with the "Viewer" runtime module or remotely (thin client) using the browser.

☞ A thin client requires an ActiveX component (**ISSymbol.ocx**) to handle screens in the browser. If the thin client is connected to the Internet, this component is downloaded and registered automatically.

16 Downloading to a remote device

16.1 Remote management

Use the remote management tool to download project files to a target device and execute/stop the project on this device.

A "target device" is each computer on which the software of the project runtime server is installed and running. For further information, refer to Installation on page 21.

The actual connection is established by a small program called “Remote Agent” (`CEServer.exe`) on the target device.



In this chapter, the term “Windows Embedded“ also includes Windows Embedded Standard (previously Windows XP Embedded) and Windows Embedded Compact (previously Windows CE, Windows Mobile, Windows Phone).

See also

- Installing EmbeddedView or CEView on a target device on page 27

16.2 Install or update a softkey license for EmbeddedView or CEView (remote)

Use the Remote management tool in Rexroth WinStudio to remotely install or update the softkey license for the EmbeddedView or CEView runtime software.

It is assumed for this task that a Windows computer on which the complete Rexroth WinStudio software is installed and a Windows Embedded device on which the EmbeddedView or CEView runtime software is installed are available, that both stations are connected to your network and that the Remote Agent runs on the Windows Embedded device.

In addition, you should have already purchased a valid license or an upgrade. It can be purchased when sending in the site code (see below) but this extends your downtime. For more information, please contact Rexroth WinStudio provider. Any license can be updated (e.g. product type, number of Thin Clients) or the software can be upgraded to a newer version. The costs of the upgrade depend on the difference between the current and the updated license settings.



Some Windows Embedded devices can be purchased with an EmbeddedView or CEView softkey license already installed. Ask your software vendor about this possibility.

These instructions apply to the installation of a new softkey license and to the update of an existing softkey license. The license used overwrites the previous one if available.

Remote installation or update of a softkey license:

1. In the WinStudio context menu, select **WinStudio target platform**. The "Remote management" dialog is displayed.

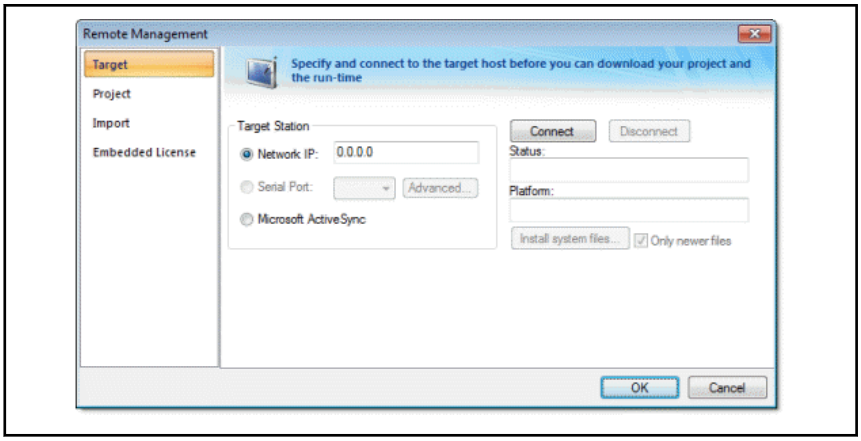


Fig. 16-1: Target tab of the Remote management dialog

2. In the **Target station** group, perform one of the following steps to configure the connection to the target device on which EmbeddedView or CE-View runs:
   - 2.1 Click on **Network IP** and then enter the IP address of the target device.
   - 2.2 Click on **Serial port** and then select a port from the displayed list or
   - 2.3 Click on **Microsoft ActiveSync**.



TCP/IP links provide better communication performance than serial links.

3. When the **Connect** button is enabled, click on the button to connect to the target device. The **Status** and **Platform** fields show whether the connection to the target device has been successfully established.
4. If the connection has been established, click on the Embedded license tab. The current license settings, if available, are displayed and a randomly generated site code for the target device.

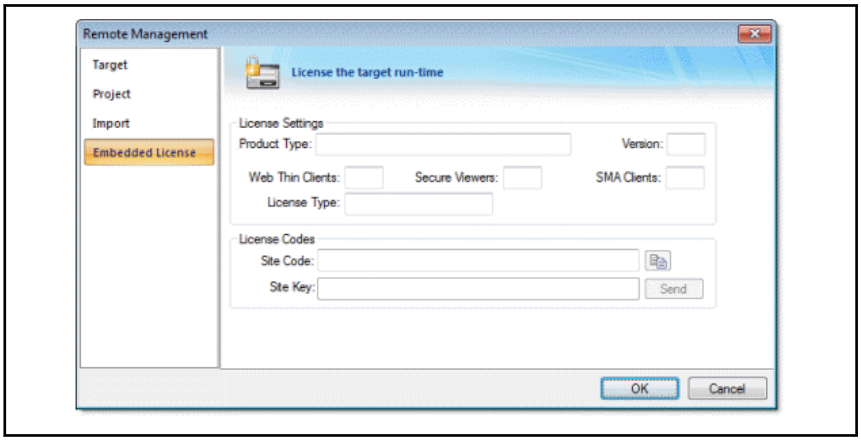


Fig. 16-2: "Embedded license" tab of the "Remote management" dialog

5. Send the site code to your software provider. Normally, the site code is to be inserted into the email to your software provider.
6. When you receive the corresponding site key from your software provider, enter or paste it into the **Site key** field and click on **Send**. (You are prompted to confirm the operation.) The new license settings are saved on the target device and a confirmation message is displayed.

If the new site key is not validated, an error message displays. If this happens, double-check that you typed the Site Key correctly. If you typed the key correctly and get an error message, contact your software vendor for further assistance.

After saving the new license settings on the Windows Embedded device, be sure to save its registry settings. If you do not save these settings, you will lose the license after rebooting the device. For more information on how to save the registry settings, refer to the manufacturer documentation.

16.3 Enabling security in the remote agent and adding users

Enable security in the remote agent on a target device to authorize only specific users to connect to the device and to encrypt the communication between the device and the project development application.

Before starting this task, the project runtime software (Rexroth WinStudio, EmbeddedView or CEView) has to be installed on the target device and the Remote Agent must be executed.

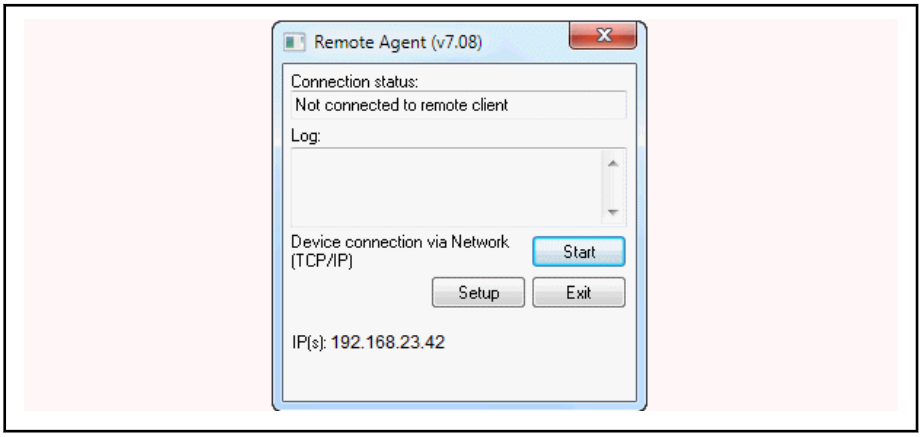


Fig. 16-3: "Remote agent" dialog

Enabling the security and adding users:

1. Click on "Setup" in the remote agent on the target device. The Setup dialog box is displayed.

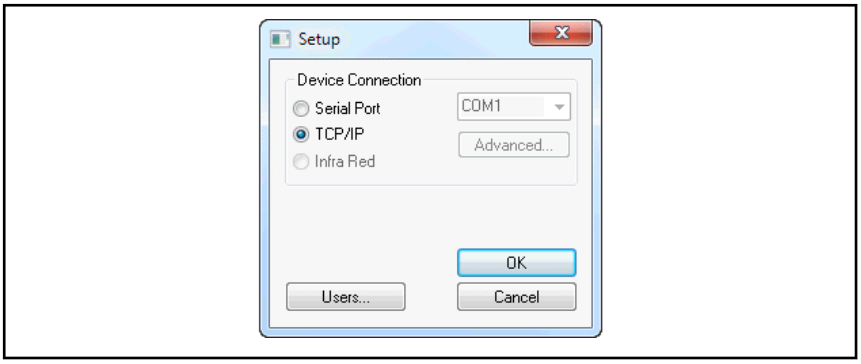


Fig. 16-4: "Setup" dialog

2. Click on **User**. The "Users" dialog is displayed.

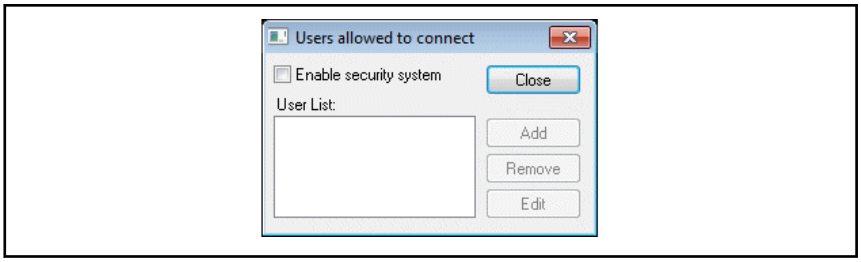


Fig. 16-5: "User" dialog

3. Select **Enable security system**.
4. Click on **Add**. The "Add user" dialog is displayed.

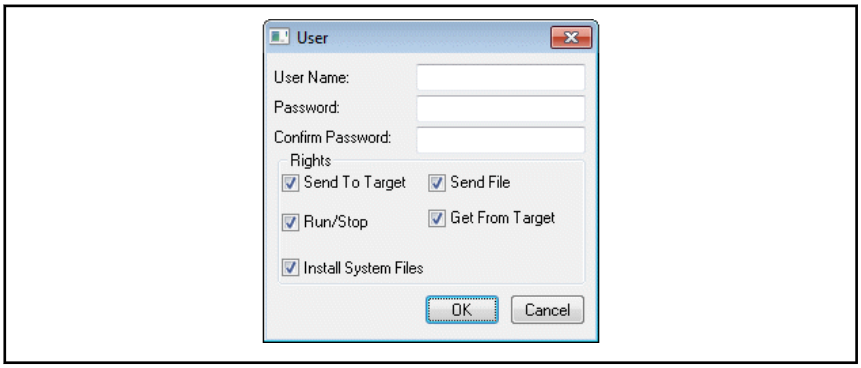


Fig. 16-6: "Add user" dialog

5. Enter **User name** and **Password** for the user and select which rights to assign to this user. The available rights correspond to the commands in the Remote management tool in the project development application.

Option	Description
Send to target	Send a complete project to the device.
Send file	Send a specified file to the device
Execute/pause	Execute or pause the project runtime server on the device
Get from target	Get a complete project from the device
Install system files	Install the runtime software EmbeddedView or CEView on the device

6. Click on **OK**. The "Add user" dialog is closed and the user is included in the **user list**.
7. Repeat these steps for all users that should connect to the device.
8. Click **Close** to close the "Users" dialog.
9. Click **OK** to close the Setup dialog but leave the Remote Agent running on the device.

Customize Remote Agent's encryption key

Customize Remote Agent's encryption key in order to increase the security of remote management.

Before you begin this task, Remote Agent must be installed and running on the target device and security must be enabled.

When you enable security in Remote Agent, connections between the project development application and the target device are automatically encrypted. This prevents third-party programs from intercepting projects or sending unauthorized Run/Stop commands to the target device.

By default, Remote Agent uses a built-in encryption key that should be secure enough for most situations. You may want to customize the key, however, because the more unique the key is, the more secure the connection will be.

To customize the key:

1. Determine what you want the key to be. It does not matter what the key actually is, because it is automatically shared between Remote Agent and the project development application. As such, you can use an online GUID generator to generate a suitable key.
2. On the target device, exit Remote Agent.
3. Edit the Remote Agent initialization file.
   - 3.1 For Remote Agent running on Windows, Windows Server, or Windows Embedded Standard, create a new text file named RemoteAPI.ini in the same directory as Remote Agent (`CEServer.exe`).
   - 3.2 For Remote Agent running on Windows Embedded Compact, edit the file named `CEServer.ini` that is located in the same directory as Remote Agent (`CEServer.exe`).
4. Insert the following lines in the initialization file.

   ```
   [Protection]
   InternalKey=<your custom key>
   ```
5. Save and close the initialization file.
6. Run Remote Agent.

Remote Agent will automatically share the new key with the project development application, when you use the Remote Management tool to connect to the target device and log on as an authorized user.

16.4 Executing or stopping your project on the target device

After configuration of the target system to receive data, proceed as follows to transfer the project data from the development station to the target device:

1. In the WinStudio context menu, select **WinStudio target platform**. The "Remote management" dialog is displayed.

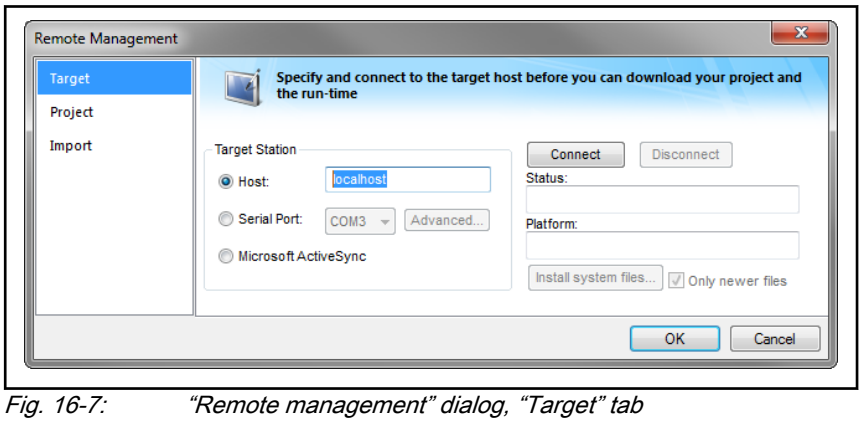


Fig. 16-7: "Remote management" dialog, "Target" tab

2. Select the **Target** tab and use the following options to specify the target system to connect to.
   - **Host**: Select this option and enter the IP address of the target system into the text box if TCP/IP connection was specified when configuring the target system.

- **Serial port**: Select this option and select a port from the combo box if a serial port connection was specified when configuring the target system.
- **Microsoft ActiveSync**: Select this option to connect to devices on which ActiveSync is enabled.

3. Click on **Connect** to connect to the target system.
4. Select the **Project** tab.

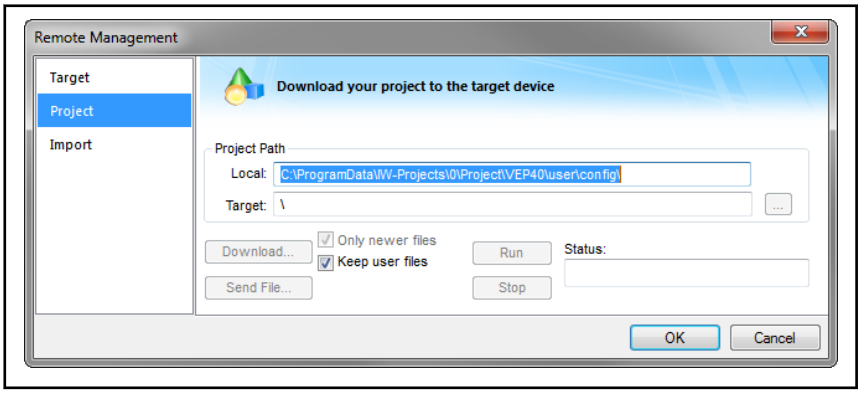


Fig. 16-8: "Remote management" dialog, "Project" tab

5. In the **Target** field, check the directory for the project files on the target system. If the path is incorrect, click on the "Browse" button to the right of the **Target** field to select a new directory. (If a proper connection to the target system is established, it can be browsed like any network drive.)

   The project files can only be downloaded to folders that are under the folder in which CEServer is installed and running.

   For example, if CEServer is installed at `\StorageCard\CEView\`, the project files are to be installed at `\StorageCard\CEView\<project_name>\`.

6. Click on **Download** to download the entire project to the target system. (Or click on **Send file** to select an individual file to send.)

NOTICE	**If a project is sent to the target system, the new project replaces the old one automatically and immediately, even if the old project is still running.** **If the Only newer files option is additionally disabled, the development application deletes all of the old files in the project folder before downloading the new files.** **In case of doubt the old project may be stopped (by clicking on Stop) before sending the new one. This way, it is ensured that the application is closed properly and not influenced if files are deleted.**

However, it is not absolutely necessary to stop the project. If the project is configured in a stable way, new files can be sent at any time.

- After initial configuration of the remote management settings, **Send project to target** can be used to send updated project files at any time without opening this dialog.
- If the download is interrupted, the development application requests confirmation to continue and indicates that the project may not run properly.

7. After the download is complete, click **Start** to start the project on the target system.

The project files can also be compressed to make them download more quickly over a slow network connection. To do this, select the "Enable file compression" check box in the Communication tab of the Project settings dialog box (Communication tab on page 67).

Compressing the file may reduce the download time if there is a slow connection between server and target system. (If there is a fast connection, however, then selecting this option may "decrease" performance because each compressed file has to be decompressed on the target system before the next file is sent. Select this option only if there is an extremely slow connection, such as dial-up connection.) Data compression is disabled by default.

16.5 Configuring the Remote Agent to automatically run a project

By default, the finished project has to be executed manually on the Windows Embedded device either by connecting to the target device from the development computer using the WinStudio target platform dialog and starting the application there (see above) or at the device itself by clicking the **Start** button in the "Remote agent" dialog.

However, the Windows Embedded device can be configured to automatically execute a specified project. To do so, edit the file `CEServer.ini` on the Windows CE device to include the following setting:

```
[Setup] AppName=Application Path
```

Where **project_name** is the location of the "WinStudio" project files on the Windows Embedded device.

Example:

```
[Setup] AppName=\Harddisk\Test\CEserverTest
```

The next time the Windows Embedded device boots up and the "Remote agent" dialog (`CEServer.exe`) is opened, this setting is read out and the specified project is automatically executed.

There are three ways to edit the `CEServer.ini` file:

- Edit the file directly on the Windows Embedded device using an attached keyboard or the touchscreen keypad. The file should be in the same folder as the file CEServer.exe on page 27.
- Connect the Windows Embedded device as a shared drive to the PC and edit the file there.
- Edit the file in the `[…]\WinStudio\ Redist` directory **before** installing the system files on the Windows Embedded device.

NOTICE

This last method changes the default copy of `CEServer.ini` that is provided with "WinStudio".

Use this method only if:

– The files have been saved before editing

– The same system files are installed on multiple, identical Windows Embedded devices and

– The directory (file path) of the "WinStudio" project files on the device is already known (perhaps by using the normal installation method on a test device).

17 Database interface

17.1 Database interface, general information

Configuring a database interface with "WinStudio" is basically linking tasks from "WinStudio" (alarms, events or trends) to tables of external databases via a specific database provider that supports the selected database.

Each history task (alarm, events or trend) can be configured to save data either to files with the proprietary format of "WinStudio" or to external SQL relational databases. Use the “Options” tab to configure the database to save the alarm and event history. (For instructions on saving the history of trend tasks, refer to Trend folder on page 340.)

"WinStudio" supports ADO.NET to provide an intuitive, simple, flexible and powerful interface with standard technologies from MDAC (Microsoft Data Access Components) such as OLE DB (Object Linking Embedded Database) and ODBC (Open Database Connectivity). By using this capability, a connection can be established to any MDAC compatible database (please refer to the conformance test table for a list of databases already tested).

The following tasks support the database interface:

- **Alarms**: WinStudio can save and/or retrieve alarm history messages in a relational database.
- **Events**: WinStudio can save and/or retrieve event messages in a relational database.
- **Trends**: WinStudio can save and/or retrieve the trend history values in a relational database.
- **Viewer**: Database information can be displayed both in table format (Alarm/event control on page 331 and Grid on page 381 objects) or in a graphical format (Trend control on page 349 object).
- **Web**: As the items listed in the following are already available in the "WinStudio" web interface, a project can be deployed that stores/saves data in a relational database and sends it over the web.

Using its embedded database interface, "WinStudio" can easily provide data from the production level to third-party systems (e.g., ERP) or receive data from them.

"WinStudio" can communicate with any relational database supported by a valid ADO.NET provider, OLE DB provider or ODBC driver. However, the conformance tests have been executed with the following databases:

Conformance test table

Database	Database version	ADO.NET provider	Assembly version
Microsoft SQL server 2000	8.0	System.Data.SqlClient	1.0.5000.0
Microsoft Access 2000	9.0.3821 SR-1	System.Data.OleDbClient	1.0.5000.0
Microsoft Excel 2000	9.0.3821 SR-1	System.Data.OleDbClient	1.0.5000.0
Oracle	10g Release 1 for Windows	System.Data.OracleClient	1.0.5000.0
Sybase	Anywhere 9.0.1.1751	iAnywhere.Data.AsaClient	9.0.1.1751
MySQL	4.0.20a	ByteFX.MySqlClient	0.7.6.15073



For information on how to configure a specific database, please refer to the following sections:

- Using ODBC databases on page 571
- Using Microsoft SQL server on page 573
- Using ORACLE databases on page 574
- Using Microsoft Access databases on page 575
- Using SQL server CE on page 578
- Using Sybase on page 580
- Using Microsoft Excel on page 580
- Using MySQL on page 583

17.2 SQL relational databases

SQL relational databases consist of information stored in tables with fields and registers and support SQL commands.

Each database can include one or more tables. Each table is composed of fields (columns) and registers (rows). Typically, the fields are pre-defined and the project adds or reads one or more registers, according to the query condition.

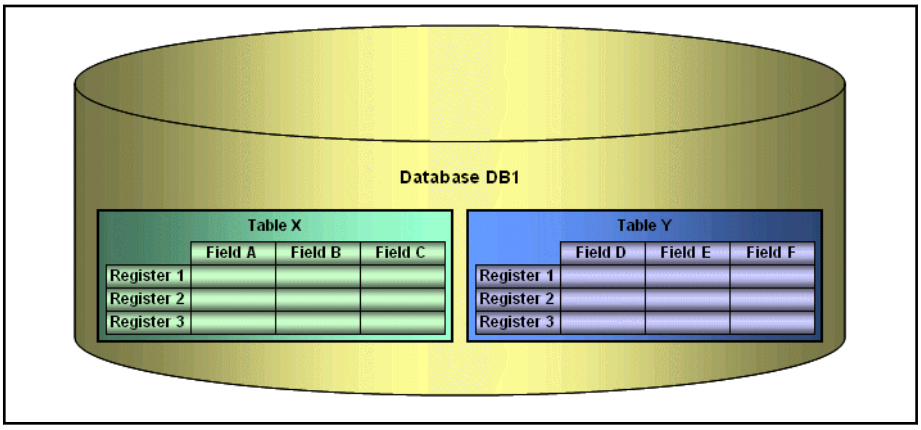


Fig. 17-1: SQL relational databases

"WinStudio" uses database providers (ADO.NET) as interface to SQL relational databases. Database providers are libraries developed to access data from different databases through SQL commands. The ADO.NET provider for a specific database can be provided by the operating system or by the database manufacturer.

The following figure illustrates how "WinStudio" can communicate with different databases using a different database provider for each database.

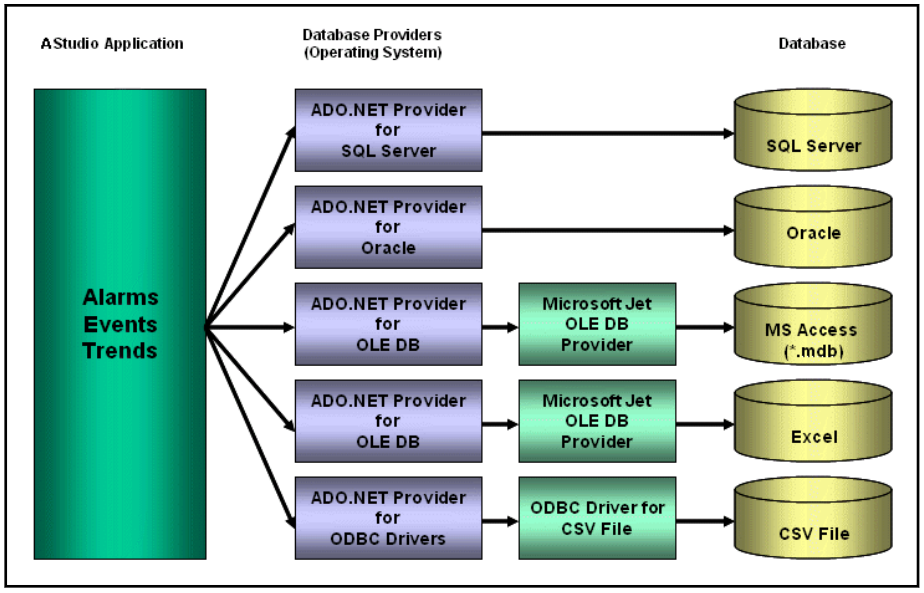


Fig. 17-2: "WinStudio" with different databases...

The previous figure illustrates some of the most popular ADO.NET providers for databases. Notice that the "Microsoft ADO.NET provider for ODBC drivers" provides access to the database through an ODBC driver. Refer to Database appendix A: Using ODBC databases on page 571. It is also possible that an OLE DB provider is available instead of an ADO.NET provider. The "Microsoft ADO.NET provider for OLE DB" provides access to the database. Using this approach, the Microsoft Jet OLE DB provider provides access to applications in the Microsoft Office package.

☞ It is important to note that "WinStudio" provides the interface for ADO.NET providers. However, ADO.NET providers and/or ODBC drivers/OLE DB providers have to be provided either by the operating system or by the database manufacturer. If the connection string does not refer to a valid ADO.NET provider, the OLE.DB provider is to be used.

Although most projects are typically only linked to one type of database, "WinStudio" provides the flexibility to link each task to another specific database supported. Furthermore, by using this architecture, it does not need to be worried about the specific characteristics of each database (this is mostly handled by the database provider for each database or by the "WinStudio" Database Gateway interface). Thus, the project settings are mostly the same, regardless of the specific database selected.

17.3 Linking the database through a remote DB provider

Depending on the architecture of the project, the ADO.NET provider for the SQL relational database may not be available on the same stations where "WinStudio" is running. This scenario is especially common, if the project is run on a Windows Embedded target system (currently, most of the providers are not supported for Windows Embedded). In order to solve this problem, a flexible solution has been developed to configure distributed systems as illustrated in the following figure:

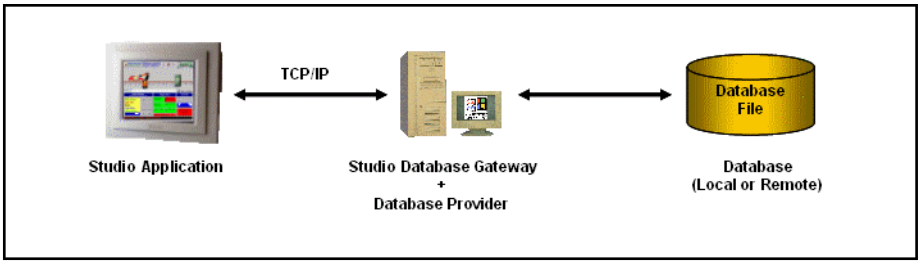


Fig. 17-3: Linking the database through a remote DB provider

The project is running on the server station (where "Rexroth WinStudio" or CEView is installed). The project can communicate with the "WinStudio" database gateway (running on a remote computer) via TCP/IP. The gateway implements the interface with the database through the database provider available on the computer where it is running.

The Studio database gateway does not require complex configuration. The files **STADOSvr.exe** and **StADOSrv.ini** are just to be copied from the **\BIN** subfolder of "WinStudio" to any directory of the computer that is to be used as gateway station and that executes the **STADOSvr.exe** program. There are advanced settings associated with the "WinStudio" database gateway, however, they are to be changed only under special circumstances. For further information on configuration of the "WinStudio" database gateway settings, refer to Studio database gateway on page 556.

☞ The Studio database gateway is a TCP/IP server for the "WinStudio" project and uses TCP port 3997 by default. If the program `STADOSvr.exe` is executed, a different port number can be specified according to the following syntax:

```
STADOSvr.exe port number
```

Example:

```
StADOSvr 3998
```

17.4 Studio database gateway

The Studio database gateway is a TCP/IP server that communicates with databases using the Microsoft .Net framework. It can run either on the same computer that is running the "WinStudio" project or on a different computer. The "Host" entry in the "Database gateway" pane of the advanced settings (refer to Database configuration dialog on page 563) defines whether the gateway is to run on the local computer. For the local computer, either "localhost" or **127.0.0.1** is to be entered as host name. The gateways do not need to be started or stopped as "WinStudio" tasks do this automatically. If the gateway is executed remotely, however, it is to be started manually. To do so, copy the files **StADOSvr.exe** and **StADOSvr.ini** from the **\BIN** folder to the remote computer and execute **StADOSvr.exe**.

The gateway can be started multiple times for different TCP/IP port numbers. The default port number is 3997 and can be changed by specifying the desired port number in the command prompt (e.g., **StADOSvr 1111**). When running the StADOSvr, the following icon is added to the system tray:



Fig. 17-4: Studio database gateway

Right-click on the icon to display the following context menu:



Fig. 17-5: Studio database gateway, context menu

The **Hide** option controls whether the "Studio database gateway" window is displayed on the desktop. (The gateway software runs continuously after starting it, regardless of whether the window is displayed.) Disable the **Hide** option, to display the window:

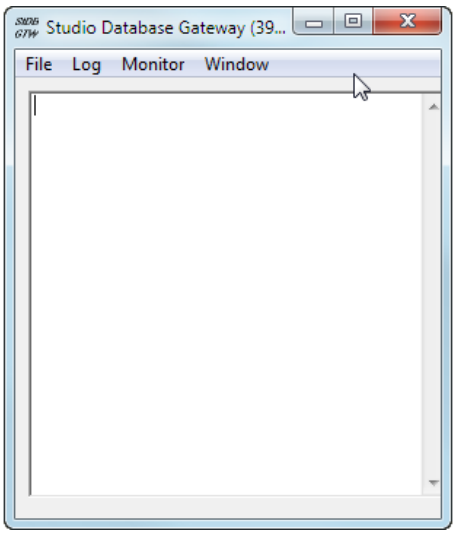


Fig. 17-6: Studio database gateway

Any errors that occur during database operation are displayed in this window and also in the "LogWin" window. The messages are reported by exceptions generated by the ADO.NET provider. (For further information on error messages in the gateway module, refer to Database debugging on page 568.)

The output can be configured in this window by using the **Log** menu:

- **Show log** menu option: Shows the "WinStudio" database gateway log files.
- **Options** menu option: Opens the "Configure messages" dialog.

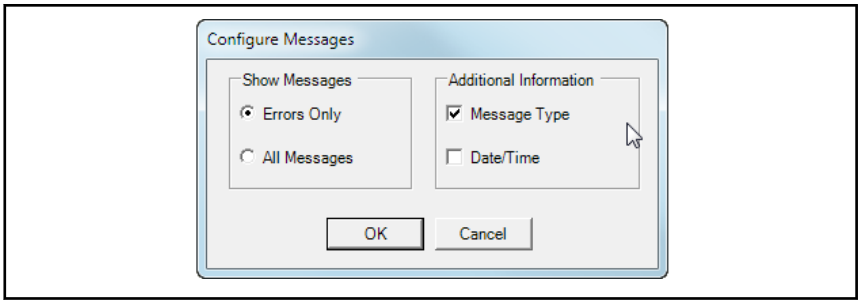


Fig. 17-7: Studio database gateway: “Configure messages” dialog

– "Show messages": Select **Errors only** to show only error messages in the log, or select **All messages** to show all database messages.
– "Additional information": In this pane, the display of additional information on each database message can be configured.
  – **Message type**: Click on (enable) this option to display the type of the message.
  – **Date/time**: Click on (enable) this option to display the time stamp of the message.

Database connections can also be monitored directly via the “Monitor” menu:

- Menu item **Connections**: Displays the “Database connections” window

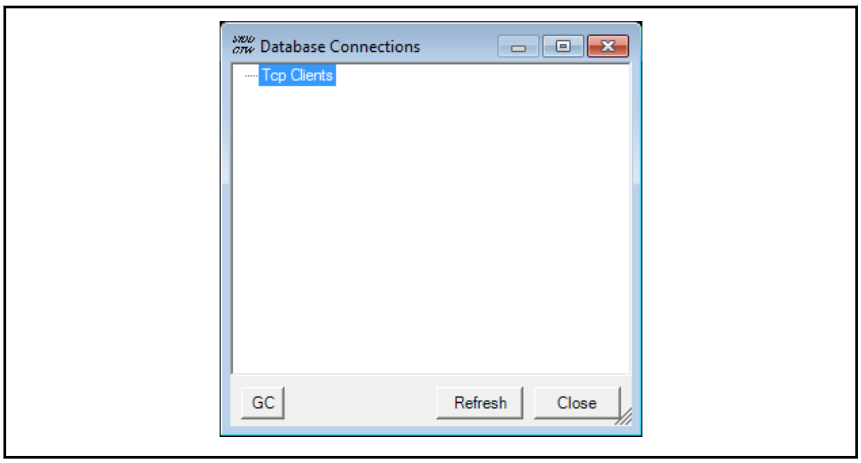


Fig. 17-8: Studio database gateway: “Database connections” window

- Menu item **Log in file**: Logs the monitoring output in a file called `log-date.txt`. It is saved in the same folder as **StADOSvr.exe**.

Extended settings

The Studio database gateway offers advanced settings that are configured in the "StADOSvr.ini" file. If there are problems communicating with a specific database, some of these settings probably have to be changed or new providers added to the file. The following parameters are available:

Section of the .INI file	Parameters	Accepted values	Description
Providers (1)	SaveMSec	1 : Disable 2 : Enable 3 : Separate column	This setting specifies the default behavior for the provider when saving milliseconds. The default can be changed in the “Advanced settings” in the “Database configuration” dialogs.
	Assembly	Any string that contains a .Net framework assembly	Assembly option for all providers. The assembly provides the classes required to communicate with the database. Most providers are inside the System.Data assembly.
	ConnectionClass	Any connection class inside the assembly	The connection class implements the System.Data.IDbConnection interface.
	DateAdapterClass	Any data adapter class inside the assembly	The data adapter class is used on operations where updates to the database are necessary. It has to be compatible with the connection class specified and is to implement IDbDataAdapter.
	CommandBuilderClass	Any command builder class inside the assembly	The command builder class is also responsible for updates of databases. It has to be compatible with the connection class.
	Provider	Name of the provider	One of the parameters in the connection string is the "provider". The Studio ADO Gateway compares the value on the connection string with the value for this parameter in each provider and defines the proper one to be used.
	ColumnDelimiterPrefix	Any character or group of characters	Specify a character that is placed before column names in SQL statements
	ColumnDelimiterSuffix	Any character or group of characters	Specify a character to be placed after column names in SQL statements
	TableDelimiterPrefix	Any character or group of characters	Specify a character that is to be placed before table names in SQL statements
	TableDelimiterSuffix	Any character or group of characters	Specify a character that is to be placed after table names in SQL statements
	ValueString	Any string	This value indicates how constants are identified in SQL statements. For Microsoft SQL databases for instance, the value is to be @Value, for ODBC a question mark (?)
	ValueStringPrefix	Any string	This value indicates a prefix to be used before the values. Oracle values, for instance, require this prefix. The SQL statements use value identifiers by using their prefixes. However, the parameters in the connection class do not use the prefix.

Section of the .INI file	Parameters	Accepted values	Description
Provider (2)	ValueAddNumber	0 or 1	Indicates whether a sequential number is to be added to the ValueString to identify the parameter or not. For Microsoft SQL databases, this parameter is to have the value 1, as parameters are identified by using @Value1, @Value2, …, @ValueN. For ODBC, this parameter is to be 0.
	BoolType	Any string representing a valid data type for the database	When trying to create columns to store Boolean values, the data type specified in this parameter is used. It has to be made sure that the data type specified can save Boolean values.
	IntegerType	Any string representing a valid data type for the database	When trying to create columns to store Integer values, the data type specified in this parameter is used. It has to be made sure that the data type specified can store 32 bit values.
	RealType	Any string representing a valid data type for the database	When trying to create columns to store Real values, the data type specified in this parameter is used. It has to be ensured that the data type specified here can store 64 Real values.
	StringType	Any string representing a valid data type for the database	When trying to create columns to store String values, the data type specified in this parameter is used. It has to be made sure that the data type specified can save the number of characters that are to be saved in the project.
	TimeStampType	Any string representing a valid data type for the database	When trying to create columns to store time stamp values, the data type specified in this parameter is used.
	EnableTop	0 or 1	If this field is set to 1, the ADO places the TOP into the SQL statement to limit the amount of registers required.
	SingleConnection	0 or 1	If this field is set to 1, the ADO opens only one connection to the database, regardless of how many tasks or computers are requesting services from it. Synchronization between the tasks is performed by the gateway and cannot be executed simultaneously if this option is enabled.
Communication	TimeOut	Any integer	Timeout (in seconds) to perform insert and update operations The standard of 2 seconds is used if no value is specified.
	LongTimeOut	Any integer	Timeout (in seconds) to perform connection and query updates. The standard of 5 seconds is used if no value is specified.
	SyncTimeOut	Any integer	Timeout (in seconds) to perform the synchronization The standard of 60 seconds is used if no value is specified.

Section of the .INI file	Parameters	Accepted values	Description
Connection	RegBufSize	Any integer	Size (in bytes) of the internal buffer used by the database API. The standard of 128 bytes is used if no value is specified.
	InsertBufferSize	Any integer	Size (in bytes) of the buffer for all data to be added to the database. This prevents timeouts or an accumulation of individual alarms/events freezing the project. The standard of 1024 bytes is used if no value is specified.
Options	CultureInfo	A common country code for languages (e.g. en-US) representing a combination of an ISO 639-1 language code and an ISO 3166-1 country code.	Language and culture to be used to format the values to SQL statements. This setting specifies for example, whether the decimal separator is a dot (###.##) or a comma (###,##) for numerical values. This is important if the option “Disable SQL variables” is selected in the settings of the database configuration. For more information, refer to Database configuration, page 56. The default value of this parameter is en-US. That means „English – United States“).
	DisableCloseQuestion	0 or 1	If this field is set to 1, no user confirmation is required to close `StADOSvr.exe`. This is important for devices that have alternative methods for exiting and restarting applications.
Monitor	Monitor	0 or 1	If this field was set to 1, the “Monitor” menu is displayed in the studio database gateway. This option is enabled by default. This was not the case in previous Rexroth WinStudio versions.

The parameters are grouped into four sections – "Providers", "Communication", "Connection" and "Options". However, all parameters for database provider configuration are listed in the "Providers" section of the file. The default values are specified in the beginning of the file, using the prefix "Default" in each parameter as illustrated in the following:

Providers

```
[Providers]
DefaultSaveMSec=3
DefaultAssembly=System.Data
DefaultConnectionClass=System.Data.OleDb.OleDbConnection
DefaultDataAdapterClass=System.Data.OleDb.OleDbDataAdapter
DefaultCommandBuilderClass=System.Data.OleDb.OleDbCommandBuilder
DefaultValueString=@Value
DefaultValueAddNumber=1 DefaultBoolType=INTEGER
DefaultIntegerType=INTEGER DefaultRealType=REAL
DefaultStringType=VARCHAR(255) DefaultTimeStampType=DATETIME
DefaultSingleConnection=0
```

The next item on the file lists the number of providers:

Count=5

Providers are identified by the "Provider" parameter followed by a number. When connecting to a database, the Provider parameter in the connection string is compared to the identification of the provider in order to determine which provider is to be used. If there is no provider with the value of the con-

nection string, all the default values are assumed. Besides its identification, each provider can have its own value for each parameter. Again, if no value is specified, the default is used. The following is an example with seven providers:

Example with seven providers

```
Count=7

Provider1=MICROSOFT.JET.OLEDB
SaveMSec1=3
ColumnDelimiterPrefix1=[
ColumnDelimiterSuffix1=]
SingleConnection1=1

Provider2=SQLOLEDB
ConnectionClass2=System.Data.SqlClient.SqlConnection
DataAdapterClass2=System.Data.SqlClient.SqlDataAdapter
CommandBuilderClass2=System.Data.SqlClient.SqlCommandBuilder
ColumnDelimiterPrefix2=[
ColumnDelimiterSuffix2=]
TableDelimiterPrefix2=[
TableDelimiterSuffix2=]
RealType2=FLOAT

Provider3=MSDASQL
ConnectionClass3=System.Data.Odbc.OdbcConnection
DataAdapterClass3=System.Data.Odbc.OdbcDataAdapter
CommandBuilderClass3=System.Data.Odbc.OdbcCommandBuilder
ValueString3=?
ValueAddNumber3=0
StringType3=VARCHAR(128)
EnableTop3=0

Provider4=ORAOLEDB
Assembly4=System.Data.OracleClient
ConnectionClass4=System.Data.OracleClient.OracleConnection
DataAdapterClass4=System.Data.OracleClient.OracleDataAdapter
CommandBuilderClass4=System.Data.OracleClient.OracleCommandBuilder
ValueString4=Value
ValueAddNumber4=1
ValueStringPrefix4=:
BoolType4=Number(1)
IntegerType4=Number(10)
RealType4=Number
StringType4=VARCHAR(255)
TimeStampType4=TIMESTAMP(0)
EnableTop4=0

Provider5=ASAPROV
Assembly5=iAnywhere.Data.AsaClient
ConnectionClass5=iAnywhere.Data.AsaClient.AsaConnection
DataAdapterClass5=iAnywhere.Data.AsaClient.AsaDataAdapter
CommandBuilderClass5=iAnywhere.Data.AsaClient.AsaCommandBuilder
ValueString5=?
ValueAddNumber5=0
ColumnDelimiterPrefix5=[
ColumnDelimiterSuffix5=]
TableDelimiterPrefix5=[
TableDelimiterSuffix5=]

Provider6=MYSQLPROV
Assembly6=ByteFX.MySqlClient
ConnectionClass6=ByteFX.Data.MySqlClient.MySqlConnection
DataAdapterClass6=ByteFX.Data.MySqlClient.MySqlDataAdapter
CommandBuilderClass6=ByteFX.Data.MySqlClient.MySqlCommandBuilder
ValueString6=@Value
ValueAddNumber6=1
StringType6=VARCHAR(128)
EnableTop6=0

Provider7=MSDAORA
Assembly7=System.Data.OracleClient
ConnectionClass7=System.Data.OracleClient.OracleConnection
DataAdapterClass7=System.Data.OracleClient.OracleDataAdapter
CommandBuilderClass7=System.Data.OracleClient.OracleCommandBuilder ValueString7=Value
ValueAddNumber7=1
ValueStringPrefix7=:
BoolType7=Number(1)
```

```
IntegerType7=Number(10)
RealType7=Number
StringType7=VARCHAR(255)
TimeStampType7=TIMESTAMP(0)
EnableTop7=0
```

17.5 Database configuration

Database connection configuration

The data exchange with an external data base file is carried out analogously to the device drivers. First, configure a new data base connection. Afterward, worksheets can be created in which the created data base connection can be selected and WinStudio tags can be linked to individual data sets of the external database.

To configure a new data base connection, open the context menu of the **WinStudio** ▸ **Tasks** ▸ **External database** folder and select the "Default database configuration" entry.



The "External database" folder is only available if "WinStudio 1.5k" or higher is selected as target system in the WinStudio properties and if the device version in the VPx/Vex properties is "13V06" or higher.

Open the configuration dialog for a new database connection using the "Add" button in the overview dialog. Enter a name in the "Name" field that can later be selected in the worksheets.

The “Database configuration” dialog allows to configure the necessary settings to link WinStudio to an external database file.

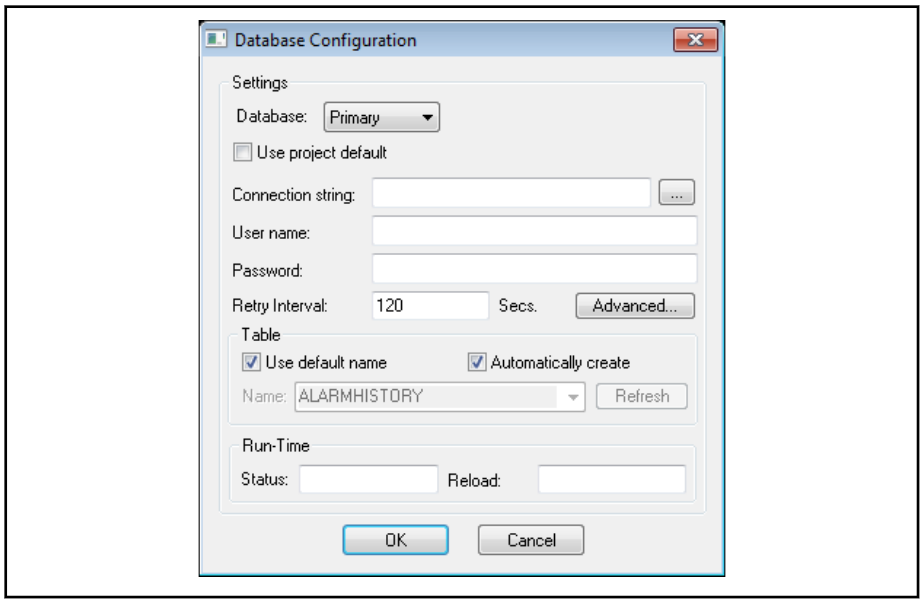


Fig. 17-9: “Default database configuration” dialog

- **Database** combo box: To select either “Primary” or “Secondary”. If "Primary" is selected, all settings of the default database configuration apply to the primary database interface. Otherwise, they apply to the secondary database interface. The secondary database can be configured in the following modes:
  - **Disabled**: In this mode, "WinStudio" saves data in the primary database only. If the primary database is unavailable for any reason,

the data is not saved anywhere else. This option may cause loss of data if the primary database is not available.

– **Redundant**: In this mode, "WinStudio" saves data in both primary and secondary databases. If one of these databases is unavailable, "WinStudio" keeps saving data only in the database that is available. If the database that was unavailable becomes available again, "WinStudio" synchronizes both databases automatically.

– **Store and forward**: In this mode, "WinStudio" saves data in the primary database only. If the primary database becomes unavailable, "WinStudio" saves the data in the secondary database. If the primary database becomes available again, "WinStudio" moves the data from the secondary database into the primary database.



The primary and secondary databases can be different types of databases. However, they have to have the same fields.

Use the secondary database to increase the reliability of the system and to use the secondary database as a backup when the primary database is not available. This architecture is particularly useful when the primary database is located in the remote station. In this case, a secondary database can be configured in the local station in such a way that data is saved temporarily if the primary database is not available (during a network failure, for instance).

- **Use project default**: If this option is checked, "WinStudio" uses the settings in the default database for the configured task (connection string, user name, password, retry interval and advanced). If this option is disabled, these settings can be configured individually for the current task.
- **Connection string**: This field defines the database where "WinStudio" writes and reads out values as well as the main parameters used when connecting to the database. Instead of entering the connection string manually, the “Browse” button (…) can be clicked to select the database type from the **Data link properties** window.

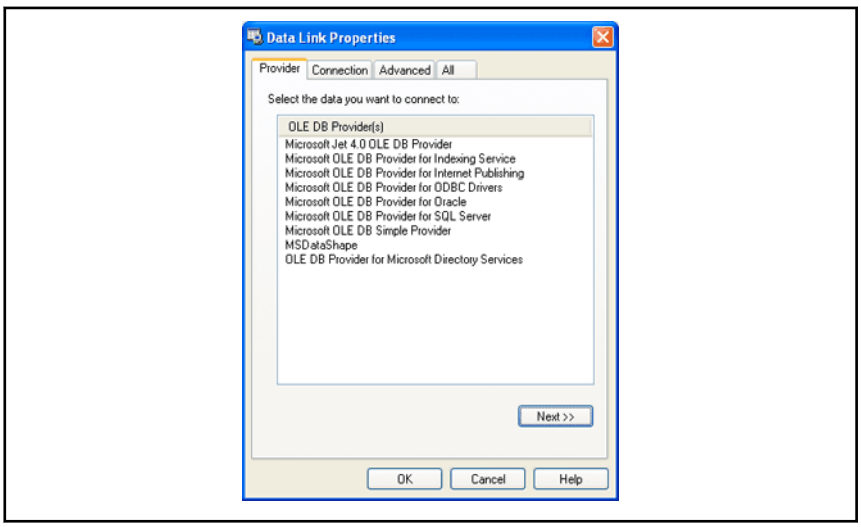


Fig. 17-10: “Data link properties” dialog



The list of database providers shown in the “Data link properties” window depends on the providers actually installed and available on the computer where "WinStudio" is running. Consult the operating system documentation (or the database documentation) for further information on the settings of the database provider for the used database.

- **User name**: User name used to connect to the database. The user name entered in this field has to match the user name configured in the database.
- **Password**: Password used to connect to the database. The password configured in this field has to match the password configured in the database.
- **Retry interval**: If "WinStudio" is unable to connect to the database for any reason, it automatically retries to connect to the database after the seconds configured in this field have passed.
- **Advanced**: Pressing this button provides access to the configuration of advanced settings. For most projects, the default value of these settings does not need to be modified and is to be retained.

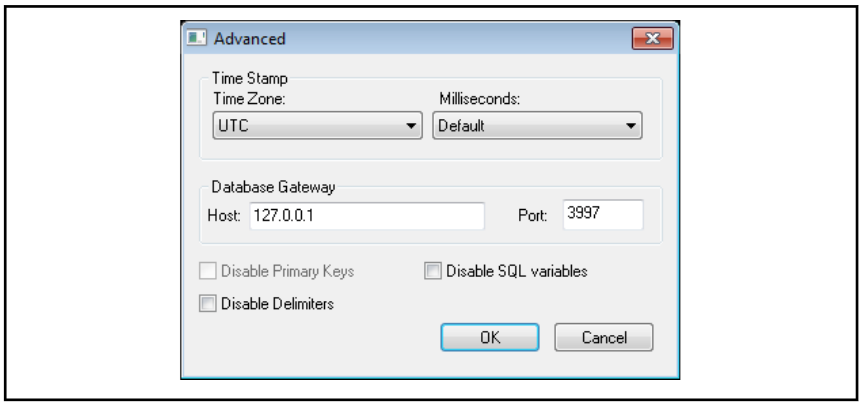


Fig. 17-11: Database configuration: “Advanced” dialog

– **Time zone**
  – **Local time + time difference**: Save the local time on your computer plus the difference (offset) between the local timezone and the Coordinated Universal Time.
  – **Local time**: Only save the local time without offset. This is not recommended.
  – **UTC**: Only save the UTC time. This is the default setting and is highly recommended in most cases.
– **Milliseconds**

  Use this item to set how the milliseconds are saved when saving the date in the database. Different databases save the date in different formats. For example, some databases do not support milliseconds in the **Date field**. The following options are available:
  – **Standard**: Uses the pre-defined format for the current database. The databases that were already tested by "Bosch Rexroth" are preconfigured with the most suitable option. When selecting "Default", "WinStudio" uses the preconfigured setting for the current database type. If a database is used

that has not been preconfigured, it is tried to save the milliseconds in a separate field if the “Default” option is selected.



The default option for each database is pre-configured in the `StADOSvr.ini` file, stored in the `\BIN` sub-directory of "WinStudio". For further information on configuration of the `StADOSvr.ini` file, refer to Studio Database Gateway on page 556.

  - **Disable**: Does not save the milliseconds at all when saving the date in the database.
  - **Enable**: Saves the milliseconds in the same field where the date is saved.
  - **Separate column**: Saves the milliseconds in a separate column. In this case, the date is saved in one field (without milliseconds) and the number of milliseconds is saved in a different column. This option is displayed if time stamps are to be saved with milliseconds precision and the used database does not support milliseconds for **date fields**.
- **Database gateway**: Enter the host name/IP address of the network node where the "WinStudio" Database Gateway is running. The TCP port number can also be specified, however, if the default is not used, the "WinStudio" database gateway has to be configured with the same TCP port. For further information on configuration of the advanced settings of the "WinStudio" ADO gateway, refer to Studio database gateway on page 556.
- **Disable primary keys**: For some modules, "WinStudio" tries to define a primary key to the table in order to speed up the queries. Enable this option if a database is used that does not support primary keys (e.g. Microsoft Excel).
- **Disable delimiters**: Select this error handling option to disable the delimiters that are used to format communications with the database. Delimiters can cause problems when a trend control or grid creates a query that includes aggregates such as min. and max.
- **Disable SQL variables**: Select this error handling option to disable SQL variables, such as "@Value1" and "?", that are often used in SQL commands and requests. Some specific database providers do not support these variables.



If this option is selected, the language or culture to be used to format the values to SQL statements may have to be specified. For further information, refer to “Advanced settings” in the Studio database gateway, page 556.

Table pane

This pane enables configuration of the settings of the table where the data is saved. All tasks can share the same database. However, each task (Alarm, Events and Trend worksheets) has to be linked to its own table. "WinStudio" does not check this field for invalid configuration. Thus, ensure that the respective setting is suitable for the used database.

- **Use default name**: If this option is checked (default), "WinStudio" saves and/or loads the data in the table with the default name written to the **Name** field.

- **Create automatically**: If this option is enabled (default), "WinStudio" automatically creates a table with the name displayed in the **Name** field. If this option is disabled, "WinStudio" does not automatically create the table. Thus, data can only be saved in the database if the database already contains a table with the name that had been preconfigured manually in the field **Name**.
- **Name**: Specifies the name of the table from the database where the history data are saved. To specify a worksheet in a Microsoft Excel spreadsheet file, use the following syntax:

  `[worksheetname$]`
- **Refresh**: If the set database is currently available, press the **Refresh** button to fill the **Name** combo box with the name of the tables currently available in the database. This way, it is possible to select the table where the history data is to be saved instead of manually entering the table name in the **Name** field.

Runtime pane

In this pane, the runtime values can be defined. The following fields are available:

- **Status** (output): The tag in this field receives one of the following values:

Value	Description
0	Disconnected from the database. The database is not available, its configuration is incorrect or an incorrect operation is executed
1	The database is successfully connected
2	The database is being synchronized

- **Reload** (output): Specify a "reload" tag when using curly brackets in the configuration field. To reconnect to the database using the updated values of the tags, set the tag in this field to 1. "WinStudio" updates the configuration when trying to perform an action in the database, setting the tag back to 0 when it is finished.

For further information, refer to:

Configuring a default database for all task histories on page 567.

17.6 Configuring a default database for all task histories

A default database can be configured to save the historical data of all tasks in a project. After configuring a default database, either use the default database or configure a new database for this special worksheet when creating a new task worksheet.

Configuration of the connection settings for the default database:

1. Open the properties dialog of the "WinStudio" folder and select the "Options" tab
2. In the "Default database" pane, click **Configure**. The "Default database configuration" dialog is displayed

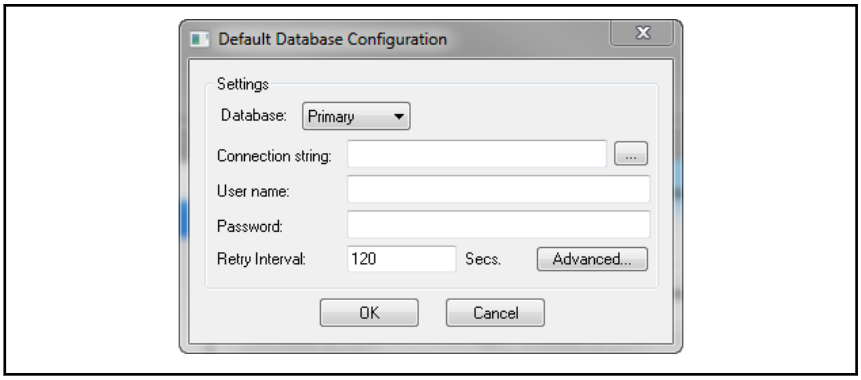


Fig. 17-12: "Default database configuration" dialog

Please refer to Default database configuration dialog on page 563 for help on the completion the fields in this window.

17.7 Database debugging

The "WinStudio" database interface provides powerful tools to identify configuration problems in connection with databases. If there are problems in communicating with a database, first enable **Database messages** in the "LogWin"window. To do so, proceed as follows:

1. Open the WinStudio output window. In the context menu of the "WinStudio" folder, select **Tools ▸ LogWin**.
2. Right-click on the "LogWin" window. By default, it is opened at the bottom of the development environment. In the context menu, click on **Settings**.

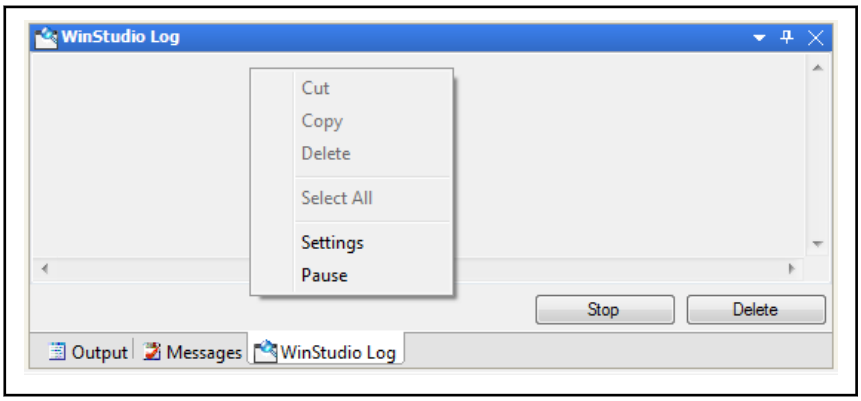


Fig. 17-13: Output window, Settings

3. In the "Log settings" dialog, enable the **Database messages** option:

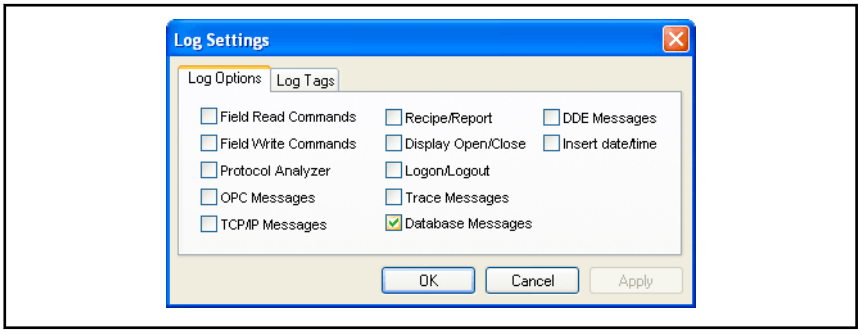


Fig. 17-14: Enabling database messages

After enabling this option, the "LogWin" window displays error messages related to the database. The following FAQ section lists some common errors that are displayed in the "LogWin" window.

Frequently asked questions

Q: The database has been configured, however, the runtime modules (alarm, trend and events) are not being saved to the database. In the "LogWin" window, only the following error message is displayed:

Database error: Error when adding a new register[CMD_ADD].

A: Most database errors in the "Output" window are followed by additional information such as the SQL command being executed, the connection string and the table name. Error messages such as the one described above usually occur after a more detailed message. For example, if your trend task fails to add a register to the database, as the cable is disconnected, a network error should be reported. If the task tries to add more registers before the time specified in the **Retry** field elapses (refer to Database configuration dialog on page 563), the following is displayed: **Database error: Error when adding a new register[CMD_ADD]**. If the configuration seems to be correct and the cause of the problem is to be limited, reduce the retry value. In this case, more detailed information is displayed.

Q: On trying to access the MySQL database server, the following message is displayed: Object is not set to an instance of an object.

A: This problem was detected under the following conditions:

- A known bug in MySQL Connector/Net v6.1.2 could not specify the character set correctly; and
- The database table that is accessed does not exist.

To solve this problem, ensure that MySQL Connector/Net v6.2.0 is used and that the accessed table exists in the database.

Q: Why is the database interface automatically closing some connections?

A: By default, the database interface can have a maximum of 1000 connections. If this maximum is exceeded, the oldest connection is automatically closed to allow the new connection and the "LogWin" window displays a detailed message describing which connection was closed and what was the last command executed.

To increase the maximum number of database connections, open the project file (**project_name.app**) in a text editor and change the following setting:

```
[StDB]
MaxConnections= "number_of_connections"
```

Please note that increasing the maximum number of connections may decrease project performance.

Q: The connection string has been configured with the browser and the Data Link Properties window. When clicking the Test button, the message "Test successful" is displayed. However, when I run my project, the database interface displays error messages and I am not able to save data.

A: The “Data link properties” window uses OLE DB as interface to the database. The "WinStudio" database interface uses ADO.NET. Thus, the OLE DB provider can be available on the computer, but not the ADO.NET provider. It is also possible that an ADO.NET provider is used that is not listed in the StADOSvr.ini file. For further information on adding ADO.NET providers to the "StADOSvr.ini" file, please refer to Studio database gateway on page 556.

Q: Why is more than one line updated in the database if data in one line of a grid object is updated?

A: The grid object issues an update command in the database using the values in all the columns for the specific line is to be updated. This problem can occur for lines with double values. If the table has a primary key or any other unique field that is not to be displayed in the "grid" object, it can be added to the **columns**, however, the **width** is to be set to 0. This fixes the problem.

Q: Why do I have to use a separate **column** to save the milliseconds to my database?

A: Some databases do not support milliseconds in the **Time stamp** field. By default, the "WinStudio" database interface requires an additional column for milliseconds. If the database can process milliseconds or if the milliseconds are not to be recorded, the default behavior can be changed in the advanced settings. Note that some databases are able to store milliseconds, however, with less precision. If different databases with different precisions are mixed in redundant mode, synchronization problems can occur.

Q: The project works fine when running in emulation mode. However, if it is sent to the Windows Embedded device, it cannot communicate with my database.

A: It might be the case that .Net framework or the used provider is not available on the Windows Embedded device. Try to use the gateway remotely by following the instructions in Linking the database through a remote DB provider on page 555.

Q: Why is the message "Error to create connection class" received when trying to connect to the database?

A: The .Net provider that is to be used is not installed on the computer. This error message is usually followed by the provider name. For example, if the Sybase database is used, the message is followed by "[iAnywhere.Data.AsaClient.AsaConnection]". The provider is iAnywhere.Data.AsaClient. To check whether the provider is installed, go to **Control Panel ▸ Administrative Tools ▸ Microsoft .Net framework x.x configuration**. The provider is generally listed in the "Assembly cache".

Q: What if the provider assembly is available (usually a .dll file), however, it is not listed in the "Assembly cache"?

A: If the assembly does not have a strong name, it can be registered in the "Assembly cache" by means of the "gcautil" program. This problem is also generally solved by copying the assembly to the same folder as the `StADOSvr.exe` (typically `[…]\WinStudio\Bin`).

Q: Under a specific condition, the table cannot be accessed from the grid. However, if no condition is applied, it works fine. What is the problem?

A: You should check for the following items:

1. Follow the Debugging steps on page 568 and check the log for error messages. An error message can indicate whether an error has been made. For example, an entry with an incorrect column name or a specification of an invalid data format.
2. In some databases, problems occur if reserved words are used as column names. Therefore, column names such as Time, Date, Numeric, etc. are to be avoided
3. If the column name starts with AND or OR (e.g., ORange), enter the name surrounded by square brackets.

   Example:

   `[ORange]=10` instead of `ORange=10`.

4. If SQL server CE is used, problems may occur when querying String fields. It has been identified that filters do not work with NCHAR data types. However, they work if these fields are declared as **NVARCHAR(<Number>)**.

   Try to recreate the table using this data type.

Example command creating a table with strings for querying:

```
CREATE TABLE Table1 (Name NVARCHAR(128), Age Numeric, Sex NVARCHAR(1))
```

MYSQL

Q: Accessing the database from a local computer does not cause any problems. However, if the project is moved to a remote device, the message **Access denied** is received.

A: Each user in a MySQL database has a property associated with it that indicates the computer from which it can get access to the database. By default, this property is set to "localhost", i.e., the database can only be accessed from the local computer. For information on changing this setting, refer to the MySQL manual.

Q: Sometimes when trying to synchronize a remote MySQL database with a local MySQL database or to use application redundancy, a connection to the ADO.NET interface is opened and never closed.

A: Open the Database configuration dialog on page 563 and disable the **Automatically create** option.

SYBASE

Q: The Sybase database has been configured with the “Browse” button. When clicking on the “Test” button, the test is successful. However, if the project is to be executed, the following error is displayed:

Database: Error: Parsing error: DSN 'MyDatabase' does not exist.

What is the problem?

A: Please refer to Database appendix F – Using Sybase on page 580 for more information on this problem.

Q: Why is the error **Error to create connection class [iAnywhere.Data.AsaClient.AsaConnection]** displayed on trying to connect to the Sybase database?

A: The **ADO.NET** provider is not installed on the computer. The database setup program has an option to install the provider. Execute the setup program again and ensure that this option is enabled.

SQL SERVER CE

Q: Why does the gateway display **TypeLoad error** when trying to access the SQL server CE database?

A: This problem usually occurs if the SQL server CE .NET provider is not installed on the CE device.

Q: What is the meaning of the error message **There is a file sharing violation. A different process might be using the file**?

A: Another program with the SQL server CE database is running. For example, this error occurs if the SQL server CE configuration software is used.

17.8 Appendices

17.8.1 Using ODBC databases

Almost every database provides an ODBC interface that can be used to communicate with it. The database functions provided by "WinStudio" can be used with ODBC drivers via the ADO.NET interface for ODBC. In order to use this option, Microsoft .NET framework 1.1 or higher has to be used.

Microsoft .NET framework 2.0 is automatically installed with "Rexroth WinStudio v7.2" or higher.

The Database configuration dialog on page 563 enables to provide connection strings that connect to an ODBC DSN. The connection string can be created automatically by clicking on the “Browse” button (...). If the "Data Link" window is displayed, the option Microsoft OLE DB Provider for ODBC Drivers is to be selected:

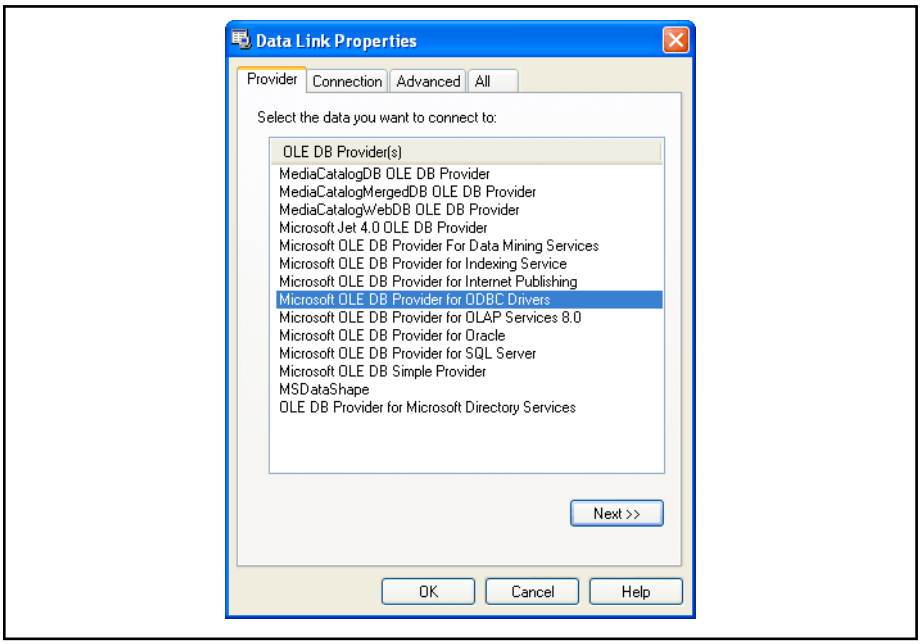


Fig. 17-15: Data link properties, provider – ODBC

Click on **Next** to open the following window:

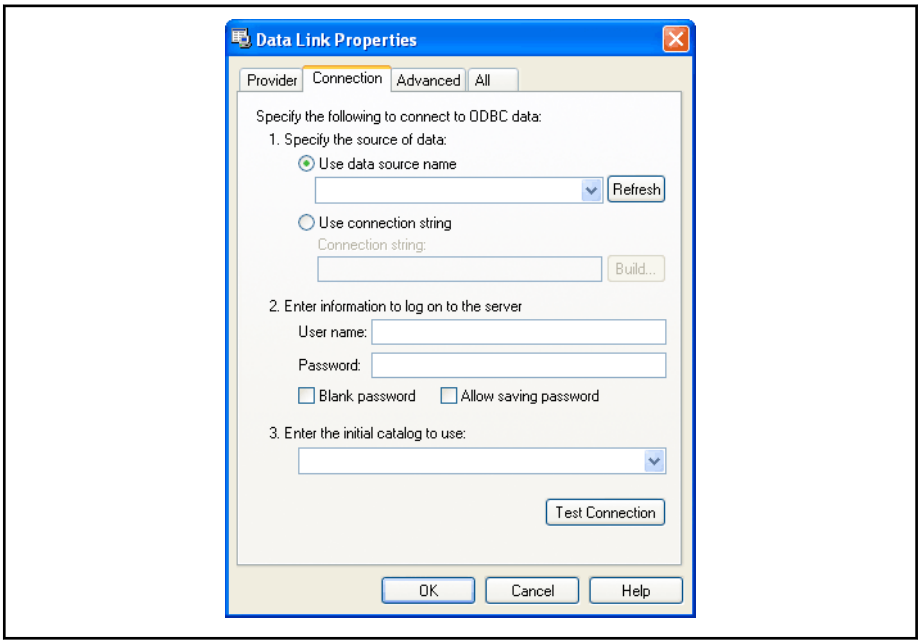


Fig. 17-16: Data link properties, provider – Connection

Select the DSN to connect to and click on **OK**. To specify the user name and password on this window instead of specifying it on the "Object properties" dialog, the **Allow saving password** checkbox has to be enabled.

17.8.2 Using Microsoft SQL server

"WinStudio" database interface enables retrieval and storage of information on Microsoft SQL server databases. To configure the SQL server database, proceed as follows:

1. Click on the **Browse** button in the Database configuration dialog on page 563. The following window is displayed:

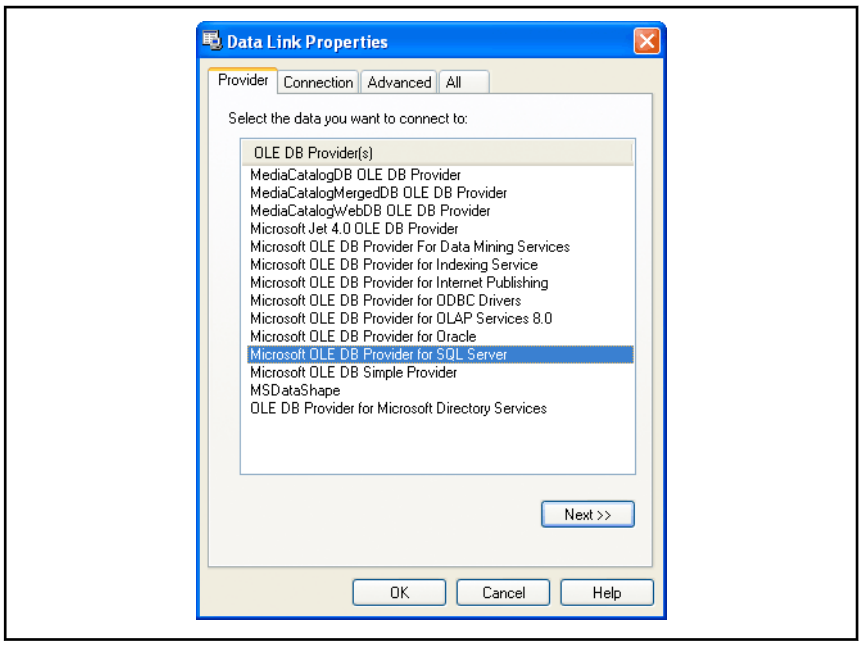


Fig. 17-17: Data link properties, provider – SQL server

2. Select the Microsoft OLE provider for SQL server and click on **Next**. The following window is displayed:

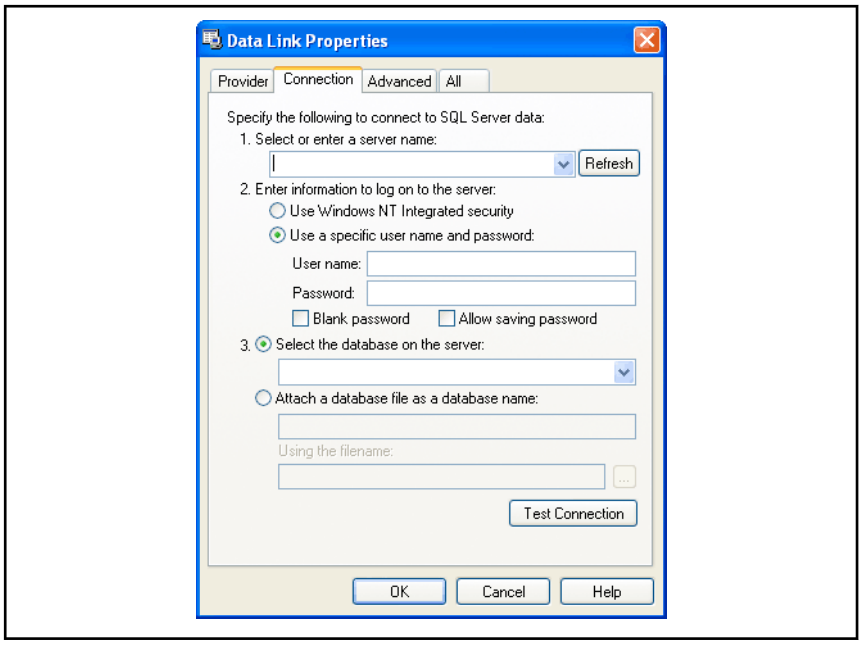


Fig. 17-18: Data link properties, connection – SQL server

3. Enter the database information into the fields of this window. If Windows NT Integrated Security is not to be used, make sure to enable the **Allow saving password** checkbox when closing the "Data link properties" window.

4. Click on **OK** to finish the "Connection strings" configuration.

The connection string should be very similar to this one:

Provider=SQLOLEDB.1; Integrated Security=SSPI; Initial Catalog=MyDatabase;

Data Source=192.168.23.200

These procedures have been tested using Microsoft SQL server 2000.

The database gateway (StADOSvr.exe) now uses a refreshed time stamp when saving data to the Microsoft SQL server databases. Thus, milliseconds do not have to be saved to a separate column. This works only with the Microsoft SQL server 2008 or higher. If you use an older SQL server version, edit the program settings to undo the changes:

1. If your project is running, stop it.
2. Search for the file “Program settings” in: `C:\Program Files\Rexroth\WinStudio\Bin\StADOSvr.ini`
3. Open the file in a text editor such as notepad.
4. Delete the following line:

   `TimeStampType2=DATETIME2`
5. Save your changes and exit the text editor.

17.8.3 Using ORACLE databases

The "WinStudio" database interface enables retrieval and storage of information on ORACLE relational databases.

To configure the ORACLE database, proceed as follows:

1. Click on the **Browse**button in the Database configuration dialog on page 563. The following window is displayed:

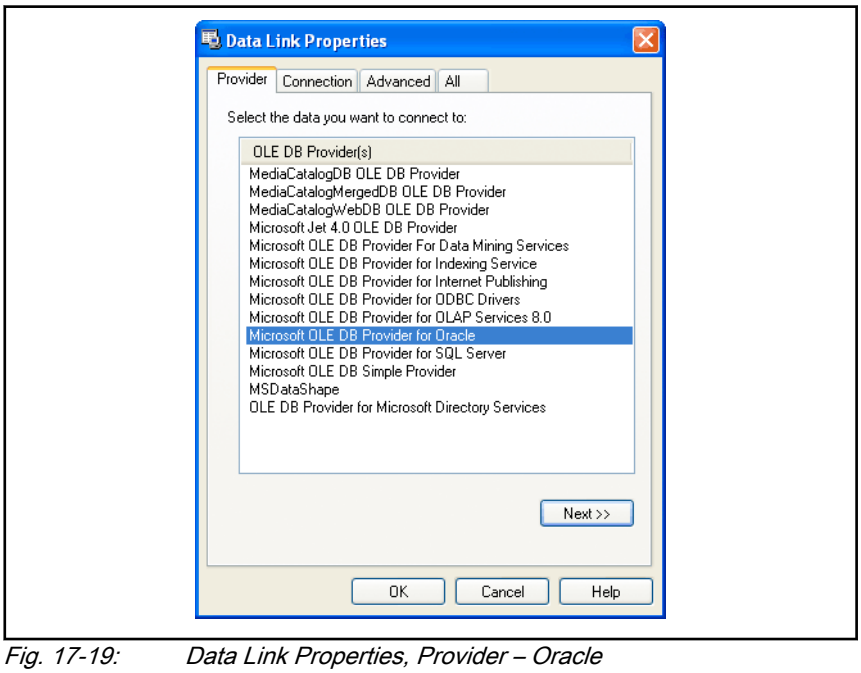


Fig. 17-19: Data Link Properties, Provider – Oracle

2. Select the Microsoft OLE provider for Oracle and click on **Next**. The following window is displayed:

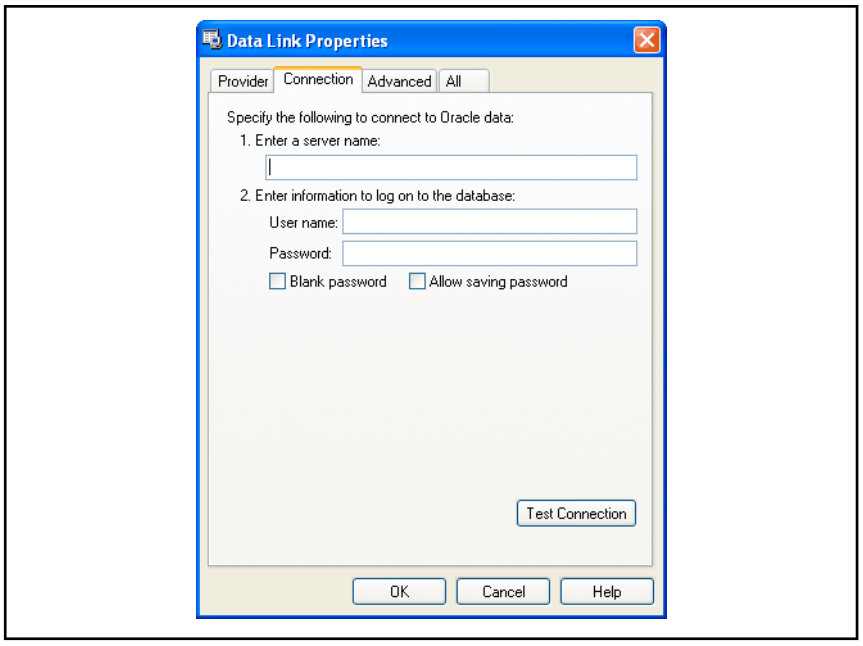


Fig. 17-20: Data link properties, connection – Oracle

3. Enter the database information into the fields of this window. Remember to check the **Allow saving password** checkbox to save the password when the "Data link properties" window is closed.

   The server name information has the following format:

   ```
   Server / TNS
   ```

   Where:

   - **Server**: Computer where the Oracle database is running
   - **TNS**: Oracle TNS name

   NOTICE

   At the time that this document was created, the Server name field could not be configured with more than 16 characters.

   If more than 16 letters are specified, the following error occurs:

   ORA-00162: external dbid length 19 is greater than maximum (16).

   Whereas 19 refers to the number of characters in the **Server name**.

   For more useful information on avoiding this problem, refer to Database debugging on page 568.

4. Click on **OK** to finish the "Connection strings" configuration.

These procedures have been tested using ORACLE 10g Release 1.

17.8.4 Using Microsoft Access databases

The "WinStudio" database interface enables storage and retrieval of information in Microsoft Access database (ACCDB) files.

Access has to be installed on the same computer as "WinStudio". "WinStudio" can directly read from or write to existing ACCDB files. However, Access has to be used to create the initial file – either a blank database for new data or a populated database for reference data. After creating the file, it can be moved to any location and a connection to it can be established.



This procedure has been last tested using Microsoft Access 2007 (12.0.6211.1000).

To establish a connection between the "WinStudio" project and an ACCDB file, proceed as follows:

1. In the Database configuration dialog on page 563, click on the “Browse” button (**...**). The "Data link properties" are displayed:

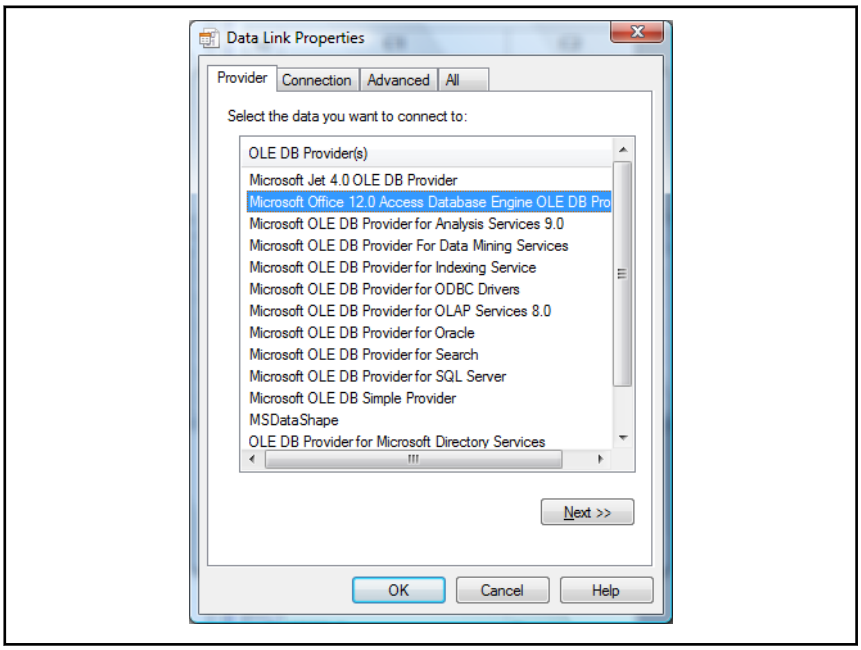


Fig. 17-21: Selecting the OLE DB provider

2. Select the respective OLE DB provider for the database:
   - For a Microsoft Access 2003 database file, select **Microsoft Jet 4.0**.
   - For a Microsoft Access 2007 database file, select **Microsoft Office 12.0 Access Database Engine**.
3. Click on **Next**.
   The "Connection" tab of the dialog is displayed:

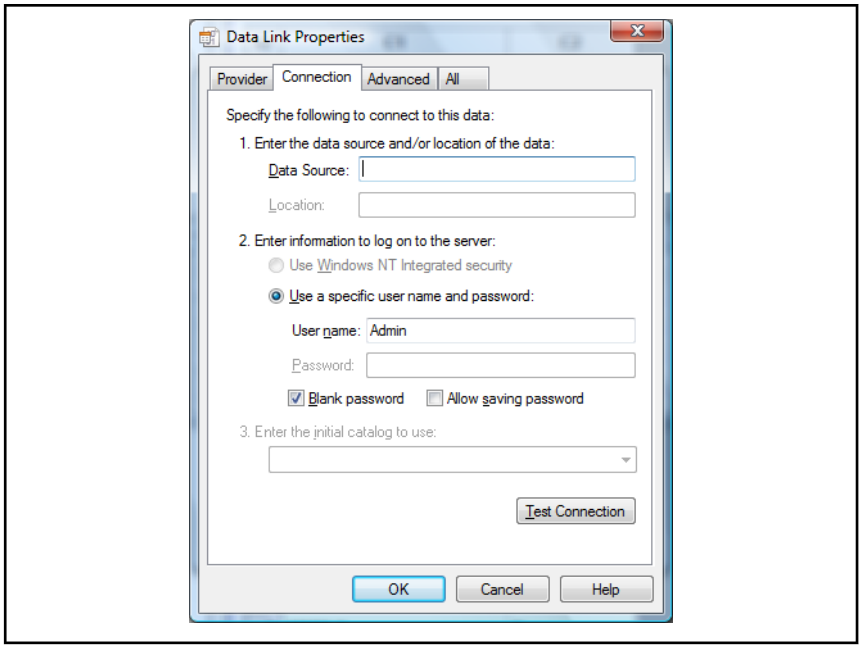


Fig. 17-22: Specifying the location of the file

4. In the **Data source** text field, enter the complete file path for the ACCDB file. Although the file can be located anywhere on the computer or network, it is recommended to keep it in the project folder. Example:

   C:\Users\username\My Documents\WinStudio-Projects\project_name\Database1.accdb

   In this case, however, the file path has to be updated if the project folder is moved.
5. If required, specify the **User name** and **Password** for the ACCDB file.
6. Click on **Test connection**.

   If a connection can be successfully established, a respective message is displayed.
7. Click on **OK** to save the changes and return to the "Database configuration" dialog.

☞ Ensure that the **Disable primary keys** option in the "Database configuration" dialog is enabled (checked). If not, "WinStudio" cannot connect to the Access database

☞ Desktop Office applications such as Microsoft Access and Microsoft Excel cannot efficiently handle large amounts of data. Trying to save all historical project data in an Access database or Excel spreadsheet, results in slow queries that may lead to unexpected results. Thus, it is recommended to use Access or Excel only as secondary database with the **Store and Forward** option enabled or to manage data with third-party software.

To handle large amounts of historical data, it is recommended to use either the proprietary format of "WinStudio" or a dedicated relational database such as Microsoft SQL server or ORACLE.

17.8.5 Using SQL server CE

Using SQL server CE, general information

"WinStudio" projects running in CEView can communicate with Microsoft SQL server CE (SQL CE) databases on the same device by using the OLE DB provider for SQL server CE. This provider has to be installed on the Windows Embedded device.

As the provider is only installed on the device and not on the development workstation, it cannot be selected in the Data link properties dialog on page 563 when developing the project. Instead, the connection string has to be entered manually using the following format:

Version	Connection string
SQL CE 2.0	Provider=SQLCE; Data Source=database_path
SQL CE 3.0	Provider=SQL_CE3.0; Data Source=database_path
SQL CE 3.5	Provider=SQL_CE3.5; Data Source=database_path



The database interface has been fully tested with these versions of SQL CE. However, other versions can also be supported. Please contact the customer service for more information.

Examples:

- Accessing a data base file at a fixed storage location, e.g. `\Harddisk\MyDatabase.sdf`:

  ```
  Provider=SQLCE; Data Source=\Harddisk\MyDatabase.sdf
  ```

  The exact path depends on the organization of the non-volatile memory on the device. Please refer to the documentation by the manufacturer.

- Accessing a database file with a storage location defined by the value of the "DatabaseFile" String tag:

  ```
  Provider=SQLCE; Data Source={DatabaseFile}
  ```

  The curly brackets ("{}") indicate that it is a tag reference.

Please note that SQL CE is only intended for lightweight databases and simple transactions. It cannot efficiently handle large amounts of data, such as the alarm and event histories. In this case, it is recommended to either use the proprietary history format or a more powerful relational database like the full version of Microsoft SQL server. If SQL CE "has to be" used, it is recommended to use it only as a secondary database with the **Store and forward** option enabled. For more information, refer to Saving the alarm history/event log to an external database on page 322.

NOTICE

When using a database worksheet on page 404 or the DB/ERP functions on page 595 to access a SQL CE database, please note that…

- Column/field names have to fit exactly or the database commands fail; and
- WinStudio data types on page 111

  are converted to potentially unexpected SQL data types.

The following table describes the conversion:

WinStudio	SQL CE
Boolean	int
Integer	int
Real	real
String	nvarchar

Installing SQL server on a Windows Embedded device

Microsoft SQL server is not included by the manufacturers of most Windows Embedded and Windows Mobile devices. If the "WinStudio" project requires SQL server, it may have to be downloaded and installed on the device.

Before proceeding, consult the documentation provided by the manufacturer to check whether the device includes Microsoft SQL server. If this is the case, the software does not need to be installed again unless it is causing problems.

If not, check at least if the device meets the System requirements on page 21 for running a "WinStudio" project and note the device processor.

As only the device manufacturer can "permanently" install software on the device with platform builder, the CAB files are actually only copied to the non-volatile memory of the device and then configured to load every time it starts.



When this documentation was created, the latest stable version of SQL Server was **Microsoft SQL server Compact Edition 3.5 Service Pack 2**.

1. To download the Microsoft SQL server Compact Edition to the Desktop, go to: http://www.microsoft.com/sqlserver/en/us/editions/compact.aspx
2. Extract and install the software from the file on the Desktop. A new directory is created with the Microsoft SQL server files including customized files for various device processors.
3. Open the directory: `C:\Program Files\Microsoft SQL server Compact Edition\v3.5\Devices\wce500`
4. In the directory, select the correct sub-directory for the respective device processor.

   For example, if the device uses a x86 processor, select the "x86" sub-directory.
5. In the sub-directory, select the CAB files to be copied to the device. For example, for x86, select the following files:
   - `sqlce.dev.ENU.wce5.x86.CAB`
   - `sqlce.repl.wce5.x86.CAB`
   - `sqlce.wce5.x86.CAB`
6. Copy the CAB files into the non-volatile memory of the device and note the storage location. The exact procedure to copy files to the device depends on the installation. You might use desktop syncing or a network connection or a USB flash drive. Check the documentation of the manufacturer for information on the available options.
7. On the desktop, create a new text file named **install.bat**.
8. Edit **install.bat** to create a batch script that loads the CAB files on start-up.

Continuing with the x86 example, edit the file to include the following code:

```
wceload /noui /delete 0 "filepath\sqlce.dev.ENU.wce5.x86.CAB"
wceload /noui /delete 0 "filepath\sqlce.repl.wce5.x86.CAB"
wceload /noui /delete 0 "filepath\sqlce.wce5.x86.CAB"
```

…where **filepath** is the location to which the CAB files were copied.

For further information, refer to the MSDN article about the Wceload tool.

9. Copy `install.bat` to the `\Windows\Startup` directory. If the `\Windows\Startup` directory is not available on the device, follow the instructions in the documentation of the manufacturer for configuration of startup elements.
10. Restart the device. After selecting and installing the files properly, SQL server runs on startup and can be accessed with the "WinStudio" project.

17.8.6 Using Sybase

The AsaClient provider has to be installed on the computer. The tests with "WinStudio" have been performed using the architecture described in section Linking the database through a remote DB provider on page 555.

If the "Browse" button is used to automatically generate the connection string, the string returned has the following format:

```
Provider=ASAProv.90; Data Source=Test
```

For this format, an ODBC DSN with the same name as the **Data source** (in this case, "Test") has to be created to communicate with the database. If the DSN is not created, the following error is displayed in LogWin when connecting to the database:

```
Database: Error: Parsing error: DSN "Test" does not exist.
```

To avoid an ODBC DSN error, the connection string can be manually used for access as illustrated in the following example:

```
Provider=ASAProv.90; DBF=C:\ Test.db
```



These procedures have been tested using Sybase Server Anywhere 9.0.1.1751.

17.8.7 Using Microsoft Excel

The "WinStudio" database interface enables storage of information in and retrieval of information from Microsoft Excel spreadsheet (XLS or XLSX) files.

Excel has to be installed on the same computer as "WinStudio". "WinStudio" can directly read from or write to existing XLS/XLSX files. However, Excel has to be used to create the initial file – either a blank spreadsheet for new

data or a populated spreadsheet for reference data. After creating the file, it can be moved to any location and a connection to it can be established.



This procedure has been last tested using Microsoft Excel 2007 (12.0.6331.5000).

To establish a connection between the "WinStudio" project and the XLS/ XLSX file, proceed as follows:

1. In the Database Configuration dialog on page 563, click on the Browse button (...). The "Data link properties" are displayed:

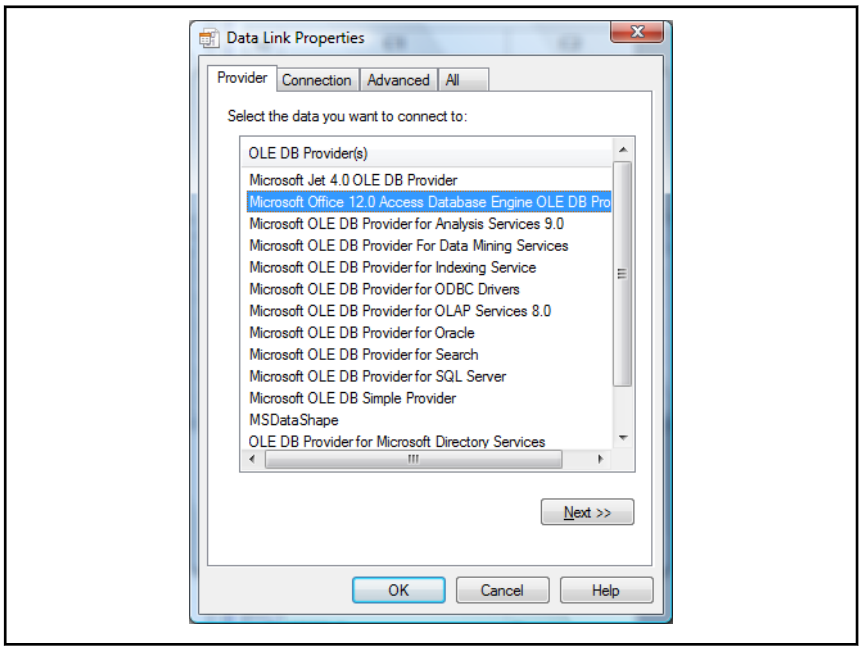


Fig. 17-23: Selecting the OLE DB provider

2. Select the respective OLE DB provider for the database:
   - For a Microsoft Excel 2003 spreadsheet file, select **Microsoft Jet 4.0**.
   - For a Microsoft Excel 2007 spreadsheet file, select **Microsoft Office 12.0 Access Database Engine**.
3. Click on **Next**.

   The "Connection" tab of the dialog is displayed:

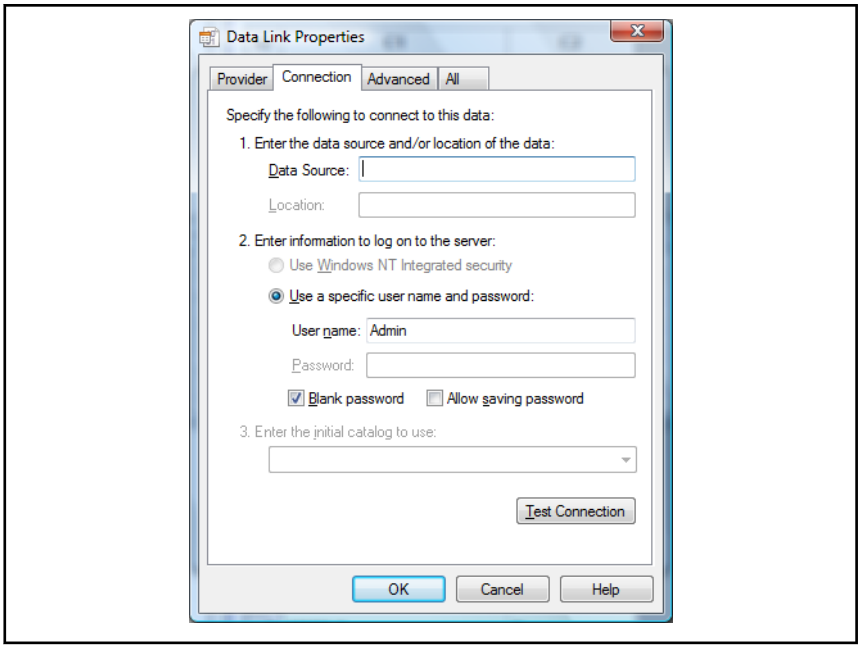


Fig. 17-24: Specifying the location of the file

4. In the **Data source** text field, enter the complete file path of the XLS/ XLSX file. Although the file can be located anywhere on the computer or network, it is recommended to keep it in the project folder. Example:

   `C:\Users\username\My Documents\WinStudio-Projects \project_name\workbook1.xls`

   In this case, however, the file path has to be updated if the project folder is moved.
5. Click on the **All** tab.
6. Select **Extended properties** and click on **Edit value**. The "Edit property value" dialog is displayed.
7. In the **Property value** text field, enter one of the following values:
   - For a Microsoft Excel 2003 spreadsheet file, enter "Excel 11.0".
   - For a Microsoft Excel 2007 spreadsheet file, enter "Excel 12.0".
8. Click on **OK** to close the "Edit property value" dialog.

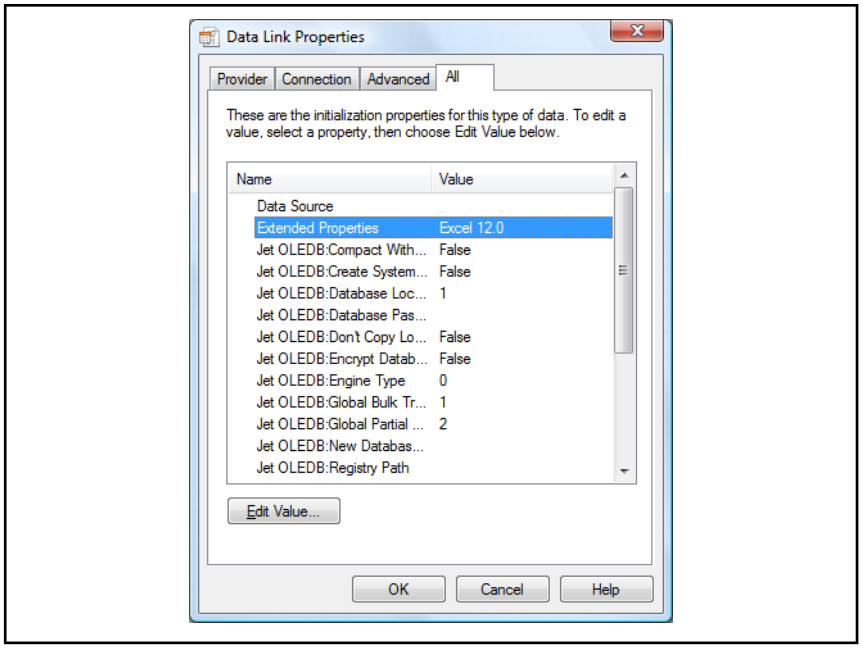


Fig. 17-25: Editing the value of extended properties

9. Click on the **Connection** tab.
10. Click on **Test connection**. If a connection can be successfully established, a respective message is displayed.
11. Click on **OK** to save the changes and return to the "Database configuration" dialog.

☞ Ensure that the **Disable primary keys** option in the "Database configuration" dialog is enabled (checked). If not, "WinStudio" cannot connect to the Excel spreadsheet.

☞ Desktop Office applications such as Microsoft Access and Microsoft Excel cannot efficiently handle large amounts of data. Trying to save all historical project data in an Access database or Excel spreadsheet, results in slow queries that may lead to unexpected results. Thus, it is recommended to use Access or Excel only as a secondary database with the **Store and forward** option enabled or to forward data to third-party software.

To handle large amounts of historical data, it is recommended to use either the proprietary format of "WinStudio" or a dedicated relational database such as Microsoft SQL server or ORACLE.

17.8.8 Using MySQL

"WinStudio" can communicate with MySQL databases, however, an ADO.Net provider for MySQL has to be installed.

The provider required for "WinStudio" is MySQL Connector/Net. When this documentation was created, the necessary software could be downloaded from the official MySQL site. (Please note that the linked site is a third-party site and may change without notice.)

After the provider is installed, the Database configuration dialog on page 563 can be used to configure a MySQL database connection. However, unlike for other database types, the "Data link properties" dialog (which is accessed by clicking **...** to the right of the **Connection string** box) cannot be used to create

the connection string. Instead, the connection string has to be entered directly using the following format:

Provider=MYSQLCLIENT; Server=myServerAddress; Database=myDataBase; Uid=myUsername; Pwd=myPassword;

The following optional parameters can be added to the connection string:

Optional parameters for the MySQL Connector/Net connection string

Parameters	Description
Port=number;	Specifies the port to be used for connection. The default port is 3306. However, any port can be specified as long as it matches the server configuration. If a port of -1 is specified, the connection uses the Named Pipes network protocol (see "Protocol" below).
Server=myServerAddress1 & myServerAddress2 & … & myServerAddressN;	Uses any server in a replicated server configuration.
Encryption=true;	Enables SSL encryption for all data sent between the client and the server. The server requires a valid certificate installed.
Encrypt=true;	An alternative to "Encryption" above, in case of errors.
Default Command Timeout=milliseconds;	Specifies a default command timeout for the connection. This does not superimpose any timeout properties on individual commands.
Connection Timeout=seconds;	Specifies how long the client waits for a server connection before terminating the attempt.
Ignore Prepare=true;	Instructs the database provider to ignore **Command.Prepare** statements to prevent corruption by server commands.
Protocol=myProtocol;	Specifies the network protocol to be used. The default is "socket" or "tcp". However, "pipe" can be specified to a use a named pipes connection or "memory" to use a commonly shared memory object.
Shared Memory Name=MySQL;	Specifies the name of the shared memory object to be used for communication. (This parameter applies only if the "Protocol" parameter above is set to "memory".)
CharSet=UTF8;	Specifies the character set to be used to encode queries to the server. Please note that query results are encoded in the same character set used to record the data itself.



These procedures have been tested using MySQL v5.1.11 and MySQL Connector/Net v6.2.0.

18 Debugging and FAQs

18.1 General Troubleshooting

If technical assistance is required, there is certain information required before contacting Technical Support. Regardless of the problem, the sequence of events that led to the detection of the problem is useful. It has to be explained as detailed as possible and is not to be improvised as it may strongly influence the time for and process of troubleshooting.

Before contacting Technical Support

Some things to try before contacting Technical Support:

- **Consult the documentation** The help function is available in all WinStudio editors. Press"F1" to open the WinStudio help.
- **Consider recent changes on the system** If something used to work, think about what may have changed. New software installation or general system changes can affect performance and general functionality of other software on the system.
- **Try reproducing the problem in a new file** If the problem can not be reproduced in a new test file, compare the new file with the original file to find and eliminate any differences. This helps to narrow down the cause of the issue.
- **Try reproducing the problem on another computer** If the problem does not occur on another computer, compare the two systems. If this is the case, there is most likely a system conflict.

Verifying the project

If any tags are changed, reordered or deleted in the tags database or if any settings in the Web tab of the Project Settings dialogue are reconfigured, the project has to be verified to update all of the screens and worksheets to the current state of the database. In the WinStudio context menu, select **Verify Application**.

Related publications

The **WinStudio Getting Started Guide** is designed for first-time users. This publication contains information about the basic functions of "Rexroth WinStudio" and it is provided in the **Documentation** folder on the "Rexroth WinStudio" installation CD.

The driver "User Guides" explain how to configure the direct communication drivers according to their unique protocol characteristics. A documentation is provided for each driver. The documentation of a driver can be displayed by selecting the respective driver folder in the "Communication" folder and pressing "F1". Alternatively, the driver description can be found in the

[…]\WinStudio\Drv folder.

Contacting Technical Support

If no answer to the technical question can be found in the product documentation or help system, the Bosch Rexroth Technical Support specialists are available to assist any customer at current product maintenance.

Please try to define the problem before contacting Technical Support to be able to repeat the steps that led to the problem and specifically identify when and how the problem occurred. The support representative needs to know exactly what the problem is in order to provide help. These steps help to pinpoint and solve the problem more quickly.

Please have the following information available:

- Hardware environment: available memory, processor type, output device
- Software environment: operating system, version of Windows®, network platform

- Product name, version number and product serial number
- Memory space installed on the system
- Free hard disk space on the system
- Screen resolution (screen size in pixels, for example, 1024 x 768)
- Screen color depth (number of colors or bits, for example, 256 colors or 8-bit color)
- Graphics card manufacturer, model name and driver version number
- Sound card manufacturer and model name
- A list of external devices connected to the computer
- Brief description of the problem or error and the specific text of any error messages
- Description of the steps taken to troubleshoot the issue. For example, how many computers have been tested and whether the issue is reproducible in a new file
- Steps to reproduce the issue if it is reproducible. If the issue is not reproducible, it may be a development issue rather than an issue with the product.

If the project crashes completely during runtime, a debugging report is generated under:

[…]\project_name\Web\Dump\WindDump.dmp.

Please have this file ready to send to Technical Support for analysis.

When contacting Bosch Rexroth, please have the system information ready.

18.2 Frequently asked questions

18.2.1 Database and security system

What does the “Shared tags” folder store?

The “Shared tags” folder stores the tags imported from the PC-based control linked to the WinStudio project. The PC-based control is linked to the project by the New Project wizard.

How to count the number of tags that are configured in the project database?

The number of tags currently used in the project is displayed in the status bar at the bottom of the development environment. Individual array elements and class members are counted as tags.

18.2.2 Graphics

How to insert and configure ActiveX objects in a project?

Inserting an ActiveX object into a project screen:

1. Open project image.
2. Click on **Active X Control** under WinStudio screen tools
3. Select the ActiveX Control and click on “OK”. The ActiveX object appears on the screen. (Unregistered ActiveX objects are not available in this list field.)
4. Double-click on the ActiveX object and assign a name to it (enter a value in the **Name** field). The animations and methods list can be viewed by selecting **Methods**. The static properties can be set with the **Properties** button (a detailed description of the object properties can be found in the component documentation provided by the component developer).

There are three functions to access the ActiveX component during runtime:

- `XGet(strName,strProperties)`:

  Returns the value of the properties strProperties from the object strName. The properties which can be read from the object are listed in the Object Properties dialog of the object, with the syntax name of the property `(PropGet)` (e.g. `Color(PropGet)`).

- `XSet(strName,strProperties,Value)`:

  Writes the value “Value” to the properties strProperties of the object strName. The properties which can be set for the object are listed in the Object Properties dialog of the object, with the syntax name of the property `(PropPut)` (e.g. `Color(PropPut)`).

- `XRun(strName, strMethod, Parameter1, Parameter2, …, ParameterN)`:

  Executes the method strMethod from the object strName according to the parameters `Parameter1, Parameter2, …, ParameterN`. The methods available in the object are listed in the “Object properties” dialog of the object with the syntax name of the method `(Method)` (e.g. `OpenFile(Method)`).



Before inserting an ActiveX control element (usually an OCX file) into the project, make sure it has been properly registered on the computer.



The number of parameters set in the "XRun on page 892" function can vary from 0 up to 255 and depends on the respective ActiveX component. To set the parameters, tags can be used. However, the tag type has to correspond to the type of component parameter (Boolean, Integer, String or Real).

How to determine a screen that opens each time the project is started?

Go to **WinStudio** ▸ **Properties** and click on the project properties on **Viewer** and enter the name of the startup screen in the startup screen field in the “Project settings” dialog.

How to insert a background picture on the screen?

Right click on the screen and select the option Screen Attributes from the popup menu. Enable the **Enable background** checkbox and select the picture format in the combo box next to the label. Copy the picture file to the screen sub-folder of the project and rename it with the same name of the screen (`ScreenName.scr` file). Using the **Shared image** option, it's possible to copy a bitmap file to the Screen sub-folder and share this picture with more than one screen. In this case, enter the bitmap name into the **Shared image** field.

Why are my graphics links no longer displayed on the Windows Embedded devices?

Rexroth WinStudio v7.3+SP4 uses a new sign system to display images on Windows Embedded devices. This new system requires that the new Windows Embedded version runs on your device and that all supported image file formats are integrated into the runtime image of the device. If an image is not correctly displayed on your device, there are two solutions.

First: Disable the new sign system and use the Legacy sign system instead. However, only .jpg, .png and .bmp are supported. To use the Legacy system, open your project file (Projectname.APP) in a text editor and edit the following property:

```
[Objects]
DrawingSystemID=<0 for new system, 1 for legacy system>
```

Second: It can be required to update the runtime image on your Windows Embedded device to ensure that all supported image file formats are integrated. (The runtime image is normally defined by the device manufacturer.) Doing so, however, is beyond the scope of this documentation. For more information, please contact technical Bosch Rexroth support

18.2.3 Tasks

How do I convert a trend history of a proprietary binary file into an ASCII text file?

Select the history file ((`*.HST`) to be converted in Windows and drag it into the utility program `HST2TXT` in the program folder (`[…]\WinStudio\Bin\HST2TXT.exe`). The converted file is automatically saved in this folder.

Alternatively, the function HST2TXT, page 746 can be used in a script or math worksheet to automatically convert file at project runtime.

How to set a DATE field for an ODBC interface with an Oracle package?

Configure the "Column" cells in the ODBC worksheet with the syntax `ColumnName.ts` (for example: `MyDate.ts`).

How to execute a Math worksheet during the startup and another Math worksheet during the project shutdown?

Startup: Execute a Math worksheet during the startup by creating a Math worksheet and filling in its **Execution** field with the expression `<TagName>=0` (for example, `StartTag=0`). In the last line of the Math worksheet, set the value 1 to the `<TagName>` tag. The type of `<TagName>tag` should be **Boolean**.

Shutdown: Instead of executing the `ShutDown()` function directly, execute one Math worksheet and configure the `ShutDown()` function in the last line of this Math worksheet.

18.2.4 Communication

How to specify the "communication error" alarm?

Configure a tag in the **Write status** or **Read status** field of the driver worksheets and configure an alarm if this tag is not 0 (zero).

How to start and stop communication drivers at runtime?

There are three functions available to handle the execution of the communication drivers at runtime:

- Start all drivers configured in the project: `StartTask("Driver")`

  Example: `StartTask("Driver")`

- Start a specific driver configured in the project:

  ```
  WinExec("StudioPath\bin\Studio Manager.exe" + " + " StudioPath\bin\Driver.dll" + " + "DriverName")
  ```

  Example:

  ```
  WinExec( Asc2Str(34) + "C:\Program Files\Rexroth \WinStudio\Bin\Studio Manager.exe" + Asc2Str(34) + " " + Asc2Str(34) + "C:\Program Files\Rexroth\WinStudio \Bin\Driver.dll" + Asc2Str(34) + " " + Asc2Str(34) + "MODBU" + Asc2Str(34) )
  ```

- Stop a specific driver configured in the project: (`EndTask "Driver-DriverName"`)

  Example: `EndTask ("DriverMODBU")`



The `Asc2Str(34)` function is used to add quotation marks for paths that contain space characters.



With the `StartTask(TaskName)` and `EndTask(TaskName)` functions, other tasks can be started or stopped.

For example, `StartTask("Viewer"), Endtask("Viewer").`

What are the parameters of the WinStudio DDE server?

WinStudio DDE server and NetDDE server parameters are listed in the following table:

Comm. Type	Application	Topic	Element
Network DDE	\\<Computer Name>\NDDE$	UNISOFT$	<TagName>
Local DDE	UNIDDE	DB	<TagName>
Network DDE	\\<Computer Name>\NDDE$	UNISOFT$	<TagName>
Local DDE	UNIDDE	DB	<TagName>

How to exchange data with Excel using NetDDE?

NetDDE can be used to exchange data via DDE protocol between networked stations.

1. Start the DDE server module from the development application.
2. Run Excel on the remote station.
3. Open an Excel worksheet and fill the cells that are to exchange data with WinStudio, using the following syntax:

   ```
   ='\computer name\NDDE$'|'UNISOFT$'!tag name
   ```

   Example:

   ```
   ='\PC\NDDE$'|'UNISOFT$'!second
   ```



- When running under Windows NT or Windows2000, it has to be made sure that the services Network DDE and Network DDE DSDM are started. (Use the Services shortcut from the Control Panel to start these services).
- When running under Windows 98, it is necessary to run the program `WindowsPath etdde.exe` in both computers (for example, `c:\Windows etdde.exe`).

Is the WinStudio OPC interface compliant with OPC specification v1.0a or v2.0?

The WinStudio modules “OPC Client” and “OPC Server” are compliant with both OPC specification v1.0a and v2.0.

How to get errors from Intellution / GE Fanuc iFIX applications?

If your project is communicating via TCP/IP page 453 with an iFIX application, then you should add the following key to your project file (i.e., `project_name.APP`):

```
[TCP]
SetQualityToBadOnError=1
```

Afterwards, if the iFIX application generates an error at runtime, the quality of the affected tags in the project is set to BAD.

This information can be obtained by reading the “Quality” tag field on page 123 (i.e., `tagname->Quality`).

18.2.5 General Information

How to start WinStudio automatically when the computer is switched on?

Create a shortcut to `[…]\WinStudio\Bin\RunStudio.exe` and place it in the Startup folder (`C:\WINDOWS\Documents and Settings\All Users\Start Menu\Programs\Startup\`).

Which features are not supported by EmbeddedView and CEView?

The main features not supported by EmbeddedView and CEView are: DDE, NetDDE, ODBC, PasteLink and a number of functions.

How to display a splash window when starting a project in EmbeddedView or CE-View?

To enable the project to display a splash window during startup, add the following key to the `CEView.ini` file of the device:

```
[OEM]
SpashWnd = Path to bitmap File // default is Splash.bmp
SplashWndTime = Time in milliseconds // default is 1000
```

How to see runtime messages and errors on a "blind" Windows Embedded device — that is, a device that has no display?

If the device has a network connection, use the tool Remote LogWin, page 503 to directly display the messages generated by the device.

The messages can be saved to a dump file (`dump.txt`) and accessed if necessary.

To create the file, add the following key to the `CEView.ini` file of the device:

```
[OEM]
BlindDevice = 1
```

What are the main steps to create a web-based application?

Proceed as follows:

1. Develop the project locally. Do not use features that are not supported by thin clients for the screens saved as HTML format.
2. After saving the screens in the standard format (**Save** in the Application menu), save the screens that have to be available for the thin client in HTML format (**Save as HTML** in the Application menu).
3. To open the configuration dialog for the web settings, go to the WinStudio context menu and click on **Tools** ▸ **Web settings**.
4. In the **Data Server IP** field, enter the IP address of the server station (i.e., the station on which the project is running).
5. To set the TCP/IP server module to **Startup=Automatic**, go to **WinStudio** ▸ **Tools** ▸ **Status** dialog “Project status”, execution task.
6. Open the project tags database and set the option **Server** instead of **Local** in the **Scope** column for all tags that have to exchange data between the server and the thin client station.
7. Verify the project (**WinStudio** ▸ **Verify**) to update the HTML files with these new settings.
8. If no web server is running on the computer, copy the program `NTWebServer.exe` from the Bin subfolder of the program directory into the Web directory (e.g., `\project name\Web`) and run it. The path where this web server program is executed is the root directory of the server. The web server is required to export data (web files) in HTTP protocol to the thin clients.
9. Run the project on the server station.
10. Using a browser (for example, Internet Explorer v4.1+SP1 or newer version) in the thin client station, type the URL address to download the screen that had been saved in HTML format (for example `http://ServerIPAddress/ScreenName.html`).

☞ The thin client requires an ActiveX component (`ISSymbol.ocx`) to handle the screens on the browser. If the thin client is connected to the Internet, this component is downloaded and registered automatically. Otherwise, it is necessary to copy it to the `\OSPath\System32` directory of the thin client and register it by the command `regsvr32 ISSymbol.ocx`. This file can be found in the `\BIN` folder in the WinStudio installation directory.

How to maintain communication between a thin client connecting via proxy and a web gateway application running on Microsoft IIS?

Microsoft Internet Information Services (IIS) has a configuration option to maintain HTTP connections. If this option is enabled, it may conflict with thin clients that are connecting via proxy. To disable this option:

1. Start the Internet services manager.
2. In the Internet information services window, open the local server (`* server name`).
3. Right-click on **Default websites** and select **Properties** from the context menu. The “Default website properties” dialog is displayed.
4. Select the **Website** tab of the “Default website properties” dialog.
5. In the connections pane of the **Web Site** tab, disable the **HTTP Keep-Alives Enabled** option.
6. Click on **OK** to save the change and close the dialog.

How to send an email from the WinStudio project?

Proceed as follows:

- Execute the function

  `CNFEMail(strSMTP,strFrom,strPOP3,strUser, strPassword,numTimeOut)`

  to configure the overall parameters used to send emails. After executing this function, the set parameters are kept in the system until the project is shut down. Thus, most projects execute this function just once after they are started:

- Execute the function

  `SendEMail(strSubject,strMessage,strTO)` and/or

  `SendEMailExt(strSubject,strMessage,strTO,strCC,strBCC, strFile1,...,strFileN)`

  each time an email message is to be sent. The main differences between both functions are listed in the following table:

Property	SendEmail	SendEmailExt
Execution	Synchronous	Asynchronous
Supports “Subject” text	Y	Y
Supports “Message” text	Y	Y
Supports TO addresses	Y	Y
Supports CC addresses	N	Y

Property	SendEmail	SendEmailExt
Supports BCC addresses	N	Y
Supports attached files	N	Y

The runtime task (TCP/IP, OPC, DDE, ODBC, etc.) does not work.

Ensure that the runtime task is set to **Automatic** in the “Execution tasks” dialog (**Tasks** on the “Home” tab of the ribbon).

Select a runtime task that has to be executed (e.g., TCP/IP server), click on **Startup** and set it to **Automatic**.

The browser of the thin client launches an error message missing the ISSymbol.ocx and does not display the screens from the server.

`ISSymbol.ocx` is the ActiveX object used by the browser on the thin client to view the websites.

If the thin client is connected to the Internet, the `ISSymbol.ocx` control is automatically downloaded and registered in the thin client station.

Otherwise, it is required to copy it to the folder `\WinNT\System32` of the thin client station and to register it manually. If it is registered, your browser can display the pages.

☞ Use the `regsvr32 ISSymbol32.ocx` command to register the ActiveX component on the thin client.

The screens are shown on the thin client (browser); however, the data (tags values) are not read from the server.

Ensure that the parameters in the column **Web data** of the project tags database are set to **Server** instead of **Local**. The tags set to **Server** maintain the same value in the server and in the thin client (browser). The tags set to **Local** have independent values in the server and in the thin client (browser).

NOTICE

It's necessary to verify the project (Verify on the Home tab of the ribbon) after modifying the tags settings. Otherwise, the changes for the websites are not applied.

The "On Up" expressions configured in the command animation are not executed.

The "On Up" expressions from the command animation are not executed if the cursor is dragged out of the object area before releasing it. If the checkbox **Release from the command object properties** window is enabled, the On Up expression is executed even if the mouse pointer is dragged out the object area before releasing it.

The trend history does not work after adding or removing tags in the Trend worksheet.

If a tag is inserted into or removed from a Trend worksheet, the format of the history files (`*.hst`) is modified. The same `.hst` file cannot have two different formats. Otherwise, the data cannot be retrieved from it properly by the trend object. To add or remove tags for history files, there are the two following options: Create a new Trend worksheet or delete the old `*.hst` files.

Which functionalities are not supported by PocketPC platforms (for example, IPaq, Cassiopeia, Jornada)?

PocketPC devices with Windows CE do not support some functionalities supported by Windows CE devices operated with the "standard" Windows CE version:

Functionality not supported by PocketPC devices
DCOM (Distributed Component Object Model): All features based on DCOM (e.g., remote OPC communication) are not supported by PocketPC devices.
The function DialGetClientIP on page 877 does not work on PocketPC devices.

19 Appendix: WinStudio-internal scripting language

19.1 Appendix: Internal scripting language, general information

This section describes all functions of the internal WinStudio scripting language. Each functional description includes complete syntax, possible output values, and examples of usage.

19.2 List of Available Functions

The functions described in the following are available in "Rexroth WinStudio".

ActiveX Control and .NET Control

Function	Execution	Windows	Embedded	Thin Client
XGet, page 891	Asynchronous	Supported	ActiveX controls only	Supported
XRun, page 892	Asynchronous	Supported	ActiveX controls only	Supported
XSet, page 893	Asynchronous	Supported	ActiveX controls only	Supported

Arithmetic

Function	Execution	Windows	Embedded	Thin Client
Abs, page 611	Synchronous	Supported	Supported	Supported
Div, page 612	Synchronous	Supported	Supported	Supported
Format, page 612	Synchronous	Supported	Supported	Supported
GetBit, page 616	Synchronous	Supported	Supported	Supported
Mod, page 617	Synchronous	Supported	Supported	Supported
Pow, page 617	Synchronous	Supported	Supported	Supported
ResetBit, page 618	Synchronous	Supported	Supported	Supported
Round, page 619	Synchronous	Supported	Supported	Supported
SetBit, page 619	Synchronous	Supported	Supported	Supported
SQRT, page 620	Synchronous	Supported	Supported	Supported
Swap16, page 620	Synchronous	Supported	Supported	Supported
Swap32, page 621	Synchronous	Supported	Supported	Supported
Trunc, page 621	Synchronous	Supported	Supported	Supported

Database/ERP functions

Function	Execution	Windows	Embedded	Thin Client
DBCursorClose, page 901	Synchronous	Supported	Supported	Supported
DBCursorColumnCount, page 902	Synchronous	Supported	Supported	Supported
DBCursorColumnInfo, page 903	Synchronous	Supported	Supported	Supported
DBCursorCurrentRow, page 904	Synchronous	Supported	Supported	Supported

Function	Execution	Windows	Embedded	Thin Client
DBCursorGetValue, page 905	Synchronous	Supported	Supported	Supported
DBCursorMoveTo, page 905	Synchronous	Supported	Supported	Supported
DBCursorNext, page 906	Synchronous	Supported	Supported	Supported
DBCursorOpen, page 907	Synchronous	Supported	Supported	Supported
DBCursorOpenSQL, page 909	Synchronous	Supported	Supported	Supported
DBCursorPrevious, page 911	Synchronous	Supported	Supported	Supported
DBCursorRowCount, page 912	Synchronous	Supported	Supported	Supported
DBDelete, page 913	Synchronous	Supported	Supported	Supported
DBExecute, page 914	Synchronous	Supported	Supported	Supported
DBInsert, page 916	Synchronous	Supported	Supported	Supported
DBSelect, page 917	Synchronous	Supported	Supported	Supported
DBUpdate, page 919	Synchronous	Supported	Supported	Supported
SyncAlarm, page 920	Asynchronous	Supported	Supported	Executed on the server
SyncAlarmStatus, page 921	Synchronous	Supported	Supported	Executed on the server
SyncEvent, page 921	Asynchronous	Supported	Supported	Executed on the server
SyncEventStatus, page 922	Synchronous	Supported	Supported	Executed on the server
SyncTrend, page 923	Asynchronous	Supported	Supported	Executed on the server
SyncTrendStatus, page 924	Synchronous	Supported	Supported	Executed on the server

Date & Time

Function	Execution	Windows	Embedded	Thin Client
ClockGetDate, page 648	Synchronous	Supported	Supported	Supported
ClockGetDayOfWeek, page 649	Synchronous	Supported	Supported	Supported
ClockGetTime, page 651	Synchronous	Supported	Supported	Supported
DateTime2Clock, page 652	Synchronous	Supported	Supported	Supported
DateTime2UTC, page 653	Synchronous	Supported	Supported	Supported
GetClock, page 655	Synchronous	Supported	Supported	Supported
GetTimeZone, page 655	Synchronous	Supported	Supported	Supported
GetTimeZoneCount, page 657	Synchronous	Supported	Supported	Supported
GetUTC, page 657	Synchronous	Supported	Supported	Supported
Hour2Clock, page 657	Synchronous	Supported	Supported	Supported
SetSystemDate, page 658	Synchronous	Supported	Supported	Supported
SetSystemTime, page 658	Synchronous	Supported	Supported	Supported

Function	Execution	Windows	Embedded	Thin Client
SetTimeZone, page 659	Synchronous	Supported	Supported (see note)	Supported
UTC2DateTime, page 660	Synchronous	Supported	Supported	Supported

Dial-up

Function	Execution	Windows	Embedded	Thin Client
DialError, page 871	Synchronous	Supported (see note)	Supported	Supported
DialGetClientIP, page 877	Synchronous	Supported (see note)	Supported	Supported
DialGetServerIP, page 879	Synchronous	Supported (see note)	Not supported	Supported
DialStatus, page 880	Synchronous	Supported (see note)	Supported	Supported
DialUp, page 882	Asynchronous	Supported (see note)	Supported	Supported
DialUpToCE, page 884	Asynchronous	Supported (see note)	Not supported	Supported
FindAllDevices, page 885	Synchronous	Supported (see note)	Supported	Supported
FindModem, page 886	Synchronous	Supported (see note)	Supported	Supported
HangUp, page 886	Synchronous	Supported (see note)	Supported	Supported
PhoneDialUp, page 887	Asynchronous	Supported (see note)	Not supported	Supported
PhoneDisableListen, page 888	Synchronous	Supported (see note)	Not supported	Supported
PhoneEnableListen, page 888	Synchronous	Supported (see note)	Not supported	Supported
PhoneHangUp, page 889	Asynchronous	Supported (see note)	Not supported	Supported
PhoneStatus, page 890	Synchronous	Supported (see note)	Not supported	Supported

The Dial-Up functions are not supported on Windows 7.

Email

Function	Execution	Windows	Embedded	Thin Client
CnfEmail, page 865	Synchronous	Supported	Supported	Supported
GetStatusSendEmailExt, page 867	Synchronous	Supported	Supported	Supported
SendEmail, page 868	Synchronous	Supported	Supported	Supported
SendEmailExt, page 869	Asynchronous	Supported	Supported	Supported

Event logger

Function	Execution	Windows	Embedded	Thin Client
SendEvent, page 894	Synchronous	Supported	Supported	Supported

File

Function	Execution	Windows	Embedded	Thin Client
DeleteOlderFiles, page 728	Synchronous	Supported	Supported	Supported
DirCreate, page 728	Synchronous	Supported	Supported	Executed on the server
DirDelete, page 729	Synchronous	Supported	Supported	Executed on the server
DirLength, page 730	Synchronous	Supported	Supported	Executed on the server
DirRename, page 730	Synchronous	Supported	Supported	Executed on the server
FileCopy, page 731	Synchronous	Supported	Supported	Supported
FileDelete, page 732	Synchronous	Supported	Supported	Supported
FileLength, page 732	Synchronous	Supported	Supported	Supported
FileReadFields, page 733	Synchronous	Supported	Supported	Not supported
FileReadMessage, page 734	Synchronous	Supported	Supported	Not supported
FileRename, page 736	Synchronous	Supported	Supported	Supported
FileWrite, page 736	Synchronous	Supported	Supported	Supported (see note)
FileWriteFields, page 737	Synchronous	Supported	Supported	Not supported
FileWriteMessage, page 739	Synchronous	Supported	Supported	Not supported
FindFile, page 740	Synchronous	Supported	Supported	Supported
FindPath, page 741	Synchronous	Supported	Supported	Supported
GetFileAttributes, page 742	Synchronous	Supported	Supported	Supported
GetFileTime, page 742	Synchronous	Supported	Supported	Supported
GetHstInfo, page 743	Synchronous	Supported	Supported	Supported
GetLine, page 744	Synchronous	Supported	Supported	Supported
HST2TXT, page 746	Asynchronous	Supported	Supported	Supported
HST2TXTIsRunning, page 748	Synchronous	Supported	Not supported	Executed on the server
PDFCreate, page 749	Synchronous	Supported (see note)	Supported	Not supported
LookupContains, page 751	Synchronous	Supported	Supported	Not supported
LookupGet, page 751	Synchronous	Supported	Supported	Not supported
LookupLoad, page 752	Synchronous	Supported	Supported	Not supported
PDFCreate, page 752	Synchronous	Supported	Not supported	Supported

Function	Execution	Windows	Embedded	Thin Client
Print, page 753	Asynchronous	Supported	Supported	Supported
RDFileN, page 754	Synchronous	Supported	Supported	Supported
WebGetFile, page 755	Synchronous	Supported	Supported	Supported

FTP

Function	Execution	Windows	Embedded	Thin Client
CnfFTP, page 895	Synchronous	Supported	Supported	Supported
FTPGet, page 896	Asynchronous	Supported	Supported	Supported
FTPPut, page 898	Asynchronous	Supported	Supported	Supported
FTPStatus, page 899	Synchronous	Supported	Supported	Supported

Graphical

Function	Execution	Windows	Embedded	Thin Client
AutoFormat, page 756	Synchronous	Supported	Supported	Supported
ExecuteCustomCall, page 757	Asynchronous	Not supported	Not supported	Supported
GetScrInfo, page 758	Synchronous	Supported	Supported	Supported
PrintSetup, page 760	Asynchronous	Supported	Supported	Supported
PrintWindow, page 761	Asynchronous	Supported	Supported	Not supported
ResetDecimalPointsTable, page 762	Synchronous	Supported	Supported	Supported
RGBColor, page 763	Synchronous	Supported	Supported	Supported
RGBComponent, page 763	Synchronous	Supported	Supported	Supported
SaveScreenShot, page 764	Synchronous	Supported	Supported	Supported (see note)
SetDecimalPoints, page 767	Synchronous	Supported	Supported	Supported
SetDisplayUnit, page 768	Synchronous	Supported	Supported	Supported
SetTagDisplayUnit, page 768	Synchronous	Supported	Supported	Supported

Log message

Function	Execution	Windows	Embedded	Thin Client
Trace, page 610	Synchronous	Supported	Supported	Not supported

Logarithmic

Function	Execution	Windows	Embedded	Thin Client
Exp, page 625	Synchronous	Supported	Supported	Supported
Log, page 626	Synchronous	Supported	Supported	Supported
Log10, page 626	Synchronous	Supported	Supported	Supported

Logical

Function	Execution	Windows	Embedded	Thin Client
False, page 628	Synchronous	Supported	Supported	Supported
If, page 628	Synchronous	Supported	Supported	Supported
Toggle, page 629	Synchronous	Supported	Supported	Supported
True, page 630	Synchronous	Supported	Supported	Supported

Loop

Function	Execution	Windows	Embedded	Thin Client
For, page 846 & "Next"	N/A	Supported	Supported	Not supported

Module activity

Function	Execution	Windows	Embedded	Thin Client
AppActivate, page 692	Asynchronous	Supported	Supported	Supported
AppIsRunning, page 693	Synchronous	Supported	Supported	Supported
AppPostMessage, page 694	Synchronous	Supported	Supported	Supported
AppSendKeys, page 695	Synchronous	Supported	Not supported	Supported
CleanReadQueue, page 696	Synchronous	Supported	Not supported	Executed on the server
CloseSplashWindow, page 696	Synchronous	Supported	Not supported	Executed on the server
DisableMath, page 697	Asynchronous	Supported	Supported	Executed on the server
EnableMath, page 697	Asynchronous	Supported	Supported	Executed on the server
EndTask, page 697	Asynchronous	Supported	Supported	Executed on the server
ExitWindows, page 698	Asynchronous	Supported	Not supported	Supported
IsScreenOpen, page 699	Synchronous	Supported	Supported	Not supported
IsTaskRunning, page 699	Synchronous	Supported	Supported	Executed on the server
IsViewerInFocus, page 700	Synchronous	Supported	Not supported	Not supported
KeyPad, page 700	Asynchronous	Supported	Supported	Supported
LogOff, page 702	Asynchronous	Supported	Supported	Supported
LogOn, page 702	Asynchronous	Supported	Supported	Supported
Math, page 703	Synchronous	Supported	Supported	Executed on the server
PostKey, page 703	Synchronous	Supported	Supported	Supported
Recipe, page 704	Synchronous	Supported	Supported	Executed on the server

Function	Execution	Windows	Embedded	Thin Client
Report, page 705	Synchronous	Supported	Supported	Executed on the server
RunGlobalProcedureAsync, page 706	Synchronous	Supported	Supported	Not supported; see note
RunGlobalProcedureAsyncGetStatus, page 708	Synchronous	Supported	Supported	Not supported; see note
RunGlobalProcedureOnFalse, page 710	Synchronous	Supported	Supported	Supported
RunGlobalProcedureOnServer, page 711	Synchronous	Supported	Supported	Supported
RunGlobalProcedureOnTrigger, page 712	Synchronous	Supported	Supported	Supported
RunGlobalProcedureOnTrue, page 714	Synchronous	Supported	Supported	Supported
RunVBScript, page 715	Synchronous	Supported	Supported	Supported
SecureViewerReload, page 716	Synchronous	Not supported	Not supported	supported (Secure Viewer only)
SendKeyObject, page 716	Synchronous	Supported	Supported	Supported
SetAppPath, page 719	Synchronous	Supported	Not supported	Executed on the server
SetViewerInFocus, page 720	Synchronous	Supported	Not supported	Supported
SetViewerPos, page 720	Synchronous	Supported	Not supported	Supported
ShutDown, page 721	Synchronous	Supported	Supported	Not supported
StartTask, page 721	Asynchronous	Supported	Supported	Executed on the server
ViewerPostMessage, page 722	Asynchronous	Supported	Supported	Supported
Wait, page 723	Synchronous	Supported	Supported	Supported
WinExec, page 724	Asynchronous	Supported	Supported	Supported
WinExecIsRunning, page 726	Synchronous	Supported	Supported	Supported

Multimedia

Function	Execution	Windows	Embedded	Thin Client
Play, page 775	Synchronous/ asynchronous	Supported	Supported	Supported

ODBC

Function	Execution	Windows	Embedded	Thin Client
ODBCBeginTrans, page 848	Synchronous	Supported	Not supported	Executed on the server
ODBCBindCol, page 849	Synchronous	Supported	Not supported	Executed on the server
ODBCCanAppend, page 850	Synchronous	Supported	Not supported	Executed on the server

Function	Execution	Windows	Embedded	Thin Client
ODBCCanTransact, page 850	Synchronous	Supported	Not supported	Executed on the server
ODBCCanUpdate, page 851	Synchronous	Supported	Not supported	Executed on the server
ODBCClose, page 851	Synchronous	Supported	Not supported	Executed on the server
ODBCCommitTrans, page 852	Synchronous	Supported	Not supported	Executed on the server
ODBCDelete, page 852	Synchronous	Supported	Not supported	Executed on the server
ODBCExecuteSQL, page 853	Synchronous	Supported	Not supported	Executed on the server
ODBCInsert, page 853	Synchronous	Supported	Not supported	Executed on the server
ODBCIsBOF, page 854	Synchronous	Supported	Not supported	Executed on the server
ODBCIsDeleted, page 854	Synchronous	Supported	Not supported	Executed on the server
ODBCIsEOF, page 855	Synchronous	Supported	Not supported	Executed on the server
ODBCIsFieldNULL, page 856	Synchronous	Supported	Not supported	Executed on the server
ODBCIsFieldNullable, page 856	Synchronous	Supported	Not supported	Executed on the server
ODBCMove, page 857	Synchronous	Supported	Not supported	Executed on the server
ODBCMoveFirst, page 857	Synchronous	Supported	Not supported	Executed on the server
ODBCMoveLast, page 858	Synchronous	Supported	Not supported	Executed on the server
ODBCMoveNext, page 858	Synchronous	Supported	Not supported	Executed on the server
ODBCMovePrev, page 859	Synchronous	Supported	Not supported	Executed on the server
ODBCOpen, page 859	Synchronous	Supported	Not supported	Executed on the server
ODBCQuery, page 860	Synchronous	Supported	Not supported	Executed on the server
ODBCRollback, page 861	Synchronous	Supported	Not supported	Executed on the server
ODBCSetFieldNULL, page 861	Synchronous	Supported	Not supported	Executed on the server
ODBCSetFilter, page 862	Synchronous	Supported	Not supported	Executed on the server

Function	Execution	Windows	Embedded	Thin Client
ODBCSetSort, page 863	Synchronous	Supported	Not supported	Executed on the server
ODBCUnbindCol, page 863	Synchronous	Supported	Not supported	Executed on the server
ODBCUpdate, page 864	Synchronous	Supported	Not supported	Executed on the server

Screen functions

Function	Execution	Windows	Embedded	Thin Client
Close, page 665	Asynchronous	Supported	Supported	Not supported
Open, page 666	Asynchronous	Supported	Supported	Supported
OpenPrevious, page 670	Asynchronous	Supported	Supported	Supported
ShowInplaceInput, page 671	Asynchronous	Supported	Supported	Supported
ShowMessageBox, page 673	Synchronous	Supported	Supported	Supported

Security

Function	Execution	Windows	Embedded	Thin Client
BlockUser, page 675	Synchronous	Supported	Supported	Supported
CheckESign, page 676	Synchronous	Supported	Supported	Supported
CheckSecurityLevel, page 677	Synchronous	Supported	Supported	Supported
CreateUser, page 678	Synchronous	Supported	Supported	Supported
ExportSecuritySystem, page 680	Synchronous	Supported	Supported	Supported
GetLastESignUser, page 681	Synchronous	Supported	Supported	Supported
GetSecuritySystemStatus, page 681	Synchronous	Supported	Supported	Supported
GetUserFullName, page 683	Synchronous	Supported	Supported	Supported
GetUserNames, page 683	Synchronous	Supported	Supported	Executed on the server
GetUserPWDAging, page 684	Synchronous	Supported	Supported	Supported
GetUserState, page 685	Synchronous	Supported	Supported	Supported
ImportSecuritySystem, page 685	Synchronous	Supported	Supported	Supported
RemoveUser, page 687	Synchronous	Supported	Supported	Supported
SetPassword, page 688	Synchronous	Supported	Supported	Supported
UnblockUser, page 689	Synchronous	Supported	Supported	Supported

Statistical

Function	Execution	Windows	Embedded	Thin Client
Avg, page 622	Synchronous	Supported	Supported	Supported
Max, page 623	Synchronous	Supported	Supported	Supported

Function	Execution	Windows	Embedded	Thin Client
Min, page 624	Synchronous	Supported	Supported	Supported
Rand, page 625	Synchronous	Supported	Supported	Supported

String Functions

Function	Execution	Windows	Embedded	Thin Client
Asc2Str, page 631	Synchronous	Supported	Supported	Supported
CharToValue, page 632	Synchronous	Supported	Supported	Supported
CharToValueW, page 633	Synchronous	Supported	Supported	Supported
ClassMembersToStrVector, page 633	Synchronous	Supported	Supported	Supported
NCopy, page 634	Synchronous	Supported	Supported	Supported
Num, page 635	Synchronous	Supported	Supported	Supported
Str, page 636	Synchronous	Supported	Supported	Supported
Str2Asc, page 636	Synchronous	Supported	Supported	Supported
StrCompare, page 636	Synchronous	Supported	Supported	Supported
StrCompareNoCase, page 637	Synchronous	Supported	Supported	Supported
StrFromInt, page 638	Synchronous	Supported	Supported	Supported
StrFromReal, page 638	Synchronous	Supported	Supported	Supported
StrFromTime, page 639	Synchronous	Supported	Supported	Supported
StrGetElement, page 640	Synchronous	Supported	Supported	Supported
StrLeft, page 641	Synchronous	Supported	Supported	Supported
StrLen, page 641	Synchronous	Supported	Supported	Supported
StrLower, page 642	Synchronous	Supported	Supported	Supported
StrRChr, page 642	Synchronous	Supported	Supported	Supported
StrRight, page 642	Synchronous	Supported	Supported	Supported
StrSetElement, page 643	Synchronous	Supported	Supported	Supported
StrStr, page 643	Synchronous	Supported	Supported	Supported
StrStrPos, page 644	Synchronous	Supported	Supported	Supported
StrTrim, page 644	Synchronous	Supported	Supported	Supported
StrTrimAll, page 645	Synchronous	Supported	Supported	Supported
StrUpper, page 646	Synchronous	Supported	Supported	Supported
ValueToChar, page 646	Synchronous	Supported	Supported	Supported
ValueWToChar, page 647	Synchronous	Supported	Supported	Supported

System info

Function	Execution	Windows	Embedded	Thin Client
DbVersion, page 778	Synchronous	Supported	Supported	Supported
GetAppHorizontalResolution, page 778	Synchronous	Supported	Not supported	Executed on the server
GetAppPath, page 779	Synchronous	Supported	Supported	Executed on the server
GetAppVerticalResolution, page 779	Synchronous	Supported	Not supported	Executed on the server
GetComputerIP, page 779	Synchronous	Supported	Supported	Supported
GetComputerName, page 780	Synchronous	Supported	Not supported	Supported
GetCursorX, page 780	Synchronous	Supported	Supported	Supported
GetCursorY, page 780	Synchronous	Supported	Supported	Supported
GetDisplayHorizontalResolution, page 781	Synchronous	Supported	Supported	Supported
GetDisplayVerticalResolution, page 781	Synchronous	Supported	Supported	Supported
GetFreeMemoryCE, page 781	Synchronous	Not supported	Supported	Supported
GetHardKeyModel, page 782	Synchronous	Supported	Not supported	Executed on the server
GetHardKeySN, page 782	Synchronous	Supported	Not supported	Executed on the server
GetIPAll, page 783	Synchronous	Supported	Supported	Supported
GetLicenseGracePeriod, page 784	Synchronous	Supported	Not supported	Supported (see note)
GetMemoryCE, page 784	Synchronous	Not supported	Supported	Supported
GetNetMACID, page 785	Synchronous	Supported	Supported	Supported
GetOS, page 786	Synchronous	Supported	Supported	Supported
GetPerformanceMetric, page 787	Synchronous	Supported	Supported	Supported
GetPrivateProfileString, page 791	Synchronous	Supported	Supported	Supported
GetProductPath, page 791	Synchronous	Supported	Supported	Supported
GetRegValue, page 791	Synchronous	Supported	Supported	Not supported
GetRegValueType, page 792	Synchronous	Supported	Supported	Not supported
GetServerHostName, page 793	Synchronous	Not supported	Not supported	Supported
GetTickCount, page 794	Synchronous	Supported	Supported	Supported
InfoAppAlrDir, page 794	Synchronous	Supported	Supported	Supported
InfoAppHSTDir, page 794	Synchronous	Supported	Supported	Supported
InfoDiskFree, page 795	Synchronous	Supported	Not supported	Supported
InfoResources, page 795	Synchronous	Supported	Supported	Supported
IsActiveXReg, page 796	Synchronous	Supported	Supported	Supported

Function	Execution	Windows	Embedded	Thin Client
IsAppChangedOnServer, page 796	Synchronous	Supported	Supported	Supported
NoInputTime, page 797	Synchronous	Supported	Supported	Supported
ProductVersion, page 798	Synchronous	Supported	Supported	Supported
RegSaveCE, page 798	Synchronous	Not supported	Supported	Not supported
ReloadAppFromServer, page 799	Synchronous	Supported	Supported	Supported
SaveAlarmFile, page 800	Synchronous	Supported	Supported	Not supported
SetAppAlarmPath, page 800	Synchronous	Supported	Supported	Executed on the server
SetAppHSTPath, page 801	Synchronous	Supported	Supported	Executed on the server
SetDateFormat, page 801	Synchronous	Supported	Supported	Supported
SetKeyboardLanguage, page 802	Synchronous	Supported	Supported	Supported
SetRegValue, page 802	Synchronous	Supported	Supported	Not supported
SetWebConfig, page 804	Synchronous	Supported	Supported	Executed on the server
SNMPGet, page 806	Synchronous	Supported	Not supported	Supported
SNMPSet, page 807	Synchronous	Supported	Not supported	Supported
WritePrivateProfileString, page 808	Synchronous	Supported	Supported	Supported

Tags database

Function	Execution	Windows	Embedded	Thin Client
ExecuteAlarmAck, page 811	Synchronous	Supported	Supported	Supported
ForceTagChange, page 812	Synchronous	Supported	Supported	Supported
GetAlarmCount, page 812	Synchronous	Supported	Supported	Not supported
GetAlarmInfo, page 813	Synchronous	Supported	Supported	Not supported (see note)
GetTagValue, page 815	Synchronous	Supported	Supported	Supported
SetTagValue, page 816	Synchronous	Supported	Supported	Supported
TagsDBAddClass, page 817	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBAddClassMember, page 817	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBAddTag, page 819	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBBeginEdit, page 820	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBEndEdit, page 821	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetAlarm, page 822	Synchronous	Supported (see note)	Not supported	Not supported

Function	Execution	Windows	Embedded	Thin Client
TagsDBGetClassMember, page 825	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetClassMemberCount, page 826	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetFirstClass, page 826	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetFirstClassMember, page 827	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetFirstTag, page 828	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetNextClass, page 829	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetNextClassMember, page 829	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetNextTag, page 830	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetTagCount, page 831	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetTagProperty, page 832	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBGetTrend, page 833	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBRemoveAlarm, page 834	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBRemoveClass, page 836	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBRemoveClassMember, page 836	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBRemoveTag, page 837	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBRemoveTrend, page 838	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBSetAlarm, page 839	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBSetTagProperty, page 842	Synchronous	Supported (see note)	Not supported	Not supported
TagsDBSetTrend, page 844	Synchronous	Supported (see note)	Not supported	Not supported

Translation

Function	Execution	Windows	Embedded	Thin Client
Ext, page 770	Synchronous	Supported	Supported	Supported
SetLanguage, page 770	Synchronous	Supported	Supported	Supported

Function	Execution	Windows	Embedded	Thin Client
SetTranslationFile, page 771	Synchronous	Supported	Supported	Supported
TranslationLoad, page 772	Synchronous	Supported	Supported	Not supported
TranslationLookupClose, page 773	Synchronous	Supported	Supported	Not supported
TranslationLookupGet, page 774	Synchronous	Supported	Supported	Not supported
TranslationLookupLoad, page 774	Synchronous	Supported	Supported	Not supported

Trigonometric

Function	Execution	Windows	Embedded	Thin Client
ACos, page 661	Synchronous	Supported	Supported	Supported
ASin, page 662	Synchronous	Supported	Supported	Supported
ATan, page 662	Synchronous	Supported	Supported	Supported
Cos, page 662	Synchronous	Supported	Supported	Supported
Cot, page 663	Synchronous	Supported	Supported	Supported
Pi, page 663	Synchronous	Supported	Supported	Supported
Sin, page 664	Synchronous	Supported	Supported	Supported
Tan, page 664	Synchronous	Supported	Supported	Supported

19.3 Structure of the Functional Descriptions

This is an explanation of the description of the internal scripting functions.

The functional descriptions are divided into several sections.

Function attributes

Each function has certain attributes that are described in a single-line table:

Function	Group	Execution	Windows	Embedded	Thin Client
Function name	Group name	synchronous or asynchronous	supported or not supported	supported or not supported	supported or not supported

The first column specifies the exact name of the function as it is to be used in the project.

The second column specifies the name of the **group** the function is assigned to. Each group combines functions according to the type of calculation or the part of the project it affects. The group name supports searching for a specific function in the function browser or the documentation.

The **Execution** column describes whether the function is executed synchronously or asynchronously:

- **Synchronous** means that the respective station requires a response or confirmation by the remote station if the function is executed on the project server or the project client. If a synchronous function is requested, the runtime environment waits until the function has been processed. This way, the execution remains synchronous on server and client. This is no problem if functions are executed very quickly. However, if a client executes many functions or the network is slow, the performance of the application may be affected.

- **Asynchronous** means that the function can be executed on either the project server or the project client and the executing function does not have to wait for the result. Thus, the project continues to run without interruption.

The following columns contain information on which target systems support the function:

- The **Windows** column provides information on whether the function can be executed on client or server stations that are running on a full desktop or server version of Microsoft Windows.
- The **Embedded** column provides information on whether the function can be executed on client and server stations under Windows CE.
- The **Thin Client** column provides information on whether the function can be executed on client stations with Secure Viewer or a web browser.

For further information about these target systems, refer to System requirements, page 21.

Syntax diagram and parameters

Syntax diagrams show how the function is to be entered and what parameters it requires.

In most cases, parameters can either contain a constant value or a variable (tag). The data type of the parameter is indicated by its prefix:

- "bool" indicates that the parameter requires a Boolean value or a Boolean variable.

  For example, either `0` or `boolTag`.
- "num" indicates that the parameter requires a numeric value or an Integer or Real variable.

  For example, either `45.6543` or `numTag`.
- "str" indicates that the parameter requires a string in quotation marks or a text variable (String tag).

  For example, either `"My string"` or `strTag`.

The additional prefix "opt" indicates that a parameter is optional. If no value is specified for the parameter, the function takes the default value specified in the parameter description.

If a parameter requires a variable or other special input, this is explained in detail in the parameter description.

Returned value

This section describes the value that is potentially returned by the function.

Some functions return a calculated value, depending on the nature of the function.

Other functions return an error code that indicates how well the function was executed. The possible codes and their meanings are provided in a table.

Notes

This section describes additional instructions or warnings on the use of the function.

Examples

This section contains code examples of how the function can be executed. Various examples illustrate how constant values, variables and optional parameters can be used to execute the function.

19.4 Log Message Functions

19.4.1 Log Message Functions – General Information

These functions are used to display status and error messages in the "Output" (for local runtime) or "Remote LogWin" window.

Function	Description
Trace, page 610	Trace is a built-in function that displays a text message in the Output window. It is typically used to debug the project.

19.4.2 Trace

"Trace" is an internal scripting function that displays a text message in the "Output" window. It is typically used to trace the program sequence.

Function	Group	Execution	Windows	Embedded	Thin Client
Trace	Log message	synchronous	supported	supported	not supported

Syntax

```
Trace(strOutputMessage)
```

Inputs

strOutputMessage

The text of the message to be displayed.

Notes

On Mobile Access, trace messages are displayed in the activity log in the browser console. For more information, see Use the activity log to troubleshoot a project screen.

Examples

Display static text that reports a specific event

```
Trace("Beginning step 5.")
```

Display a date or time stamp by referencing the appropriate system tag

```
Trace(Date)
```

Concatenate static text, tag references, and function calls to form a complex message:

```
Trace("The current second of the minute is " + Second + " and the system tick is " +
 GetTickCount() + " ms.")
```

19.5 Arithmetic Functions

19.5.1 Arithmetic Functions – General Information

These functions are used to perform arithmetic operations and bit manipulation on numeric values.

Function	Description
Abs, page 611	Gets the absolute value of a specified numeric value.
Div, page 612	Returns the dividend of two whole numbers.
Format, page 612	Formats a numerical value and returns it as a string.
GetBit, page 616	Gets the value of a single bit in a numeric value.
Mod, page 617	Gets the remainder from a division operation.
Pow, page 617	Gets the result of raising a numeric value to a specified exponent.
ResetBit, page 618	Resets a single bit in an Integer tag to 0.
Round, page 619	Rounds numValue to the nearest integer.
SetBit, page 619	Sets a single bit in an Integer tag to 1.

Function	Description
SQRT, page 620	Takes the square root of numValue.
Swap16, page 620	Swaps the two lower bytes of a tag.
Swap32, page 621	Swaps two words in a tag.
Trunc, page 621	Truncates the value of numValue.

19.5.2 Arithmetic Operators

The internal scripting language supports the following arithmetic operators.

Arithmetic operators

Operator	Usage	Description
+	X + Y	Add (plus)
-	X - Y	Subtract (minus)
*	X * Y	Multiply by
/	X / Y	Divide by

Arithmetic operators are resolved from left to right according to the standard order of operations. To change the order, the part of the equation to be resolved first is to be enclosed in parentheses. For example, the following equation produces a result of 11 as multiplication is carried out before addition. The equation multiplies 2 by 3 and then adds 5 to the result:

5+2*3

In contrast, if parentheses are used to change the syntax, 5 and 2 are added and then multiplied by 3 to produce a result of 21:

(5+2)*3



For more complex mathematical operations, use the Arithmetic on page 610, Statistical on page 622, Logarithmic on page 625 and Trigonometric on page 661 functions.

19.5.3 Abs

"Abs" is an internal scripting function that gets the absolute value of a specified numeric value.

Function	Group	Execution	Windows	Embedded	Thin Client
Abs	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Abs(numValue)
```

Inputs

numValue
The numeric value out of which the function determines the absolute value.

Returned value

The absolute value of the specified numeric value.

Examples

Expression	Returned value
`ABS( "-54.9788" )`	`54.9788`
`ABS( numValue )`	absolute value of the number in the `numValue` tag

19.5.4 Div

"Div" is an internal scripting function to calculate the quotient of two specified numeric values.

Function	Group	Execution	Windows	Embedded	Thin Client
Div	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Div( numNumerator, numDenominator )
```

Inputs

numNumerator

The numerator of the division (dividend).

Please note that if you specify a decimal value, then it will be truncated.

numDenominator

The denominator of the division (divisor).

Please note that if you specify a decimal value, then it will be truncated.

Returned value

This function returns the quotient only as a whole number. The remainder is omitted.



Use the Mod page 617 function to get the remainder instead of the quotient.

Examples

Expression	Returned value
`Div( 100, 8 )`	`12`
`Div( 16, 4 )`	`4`
`Div( 100, 2.5 )`	`50`

19.5.5 Format

"Format" is an internal scripting function that formats a numeric value and returns it as a string.

Function	Group	Execution	Windows	Embedded	Thin Client
Format	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Format(strFlag,numValue,optStrDecimalMark,optStrThousandSep)
```

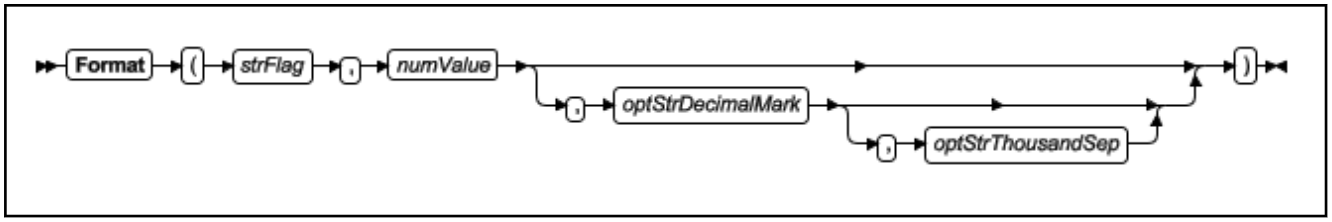


Fig. 19-1: Syntax

Inputs

strFlag

A description of how the given numeric value is to be formatted according to the syntax

`%width.precisionFormat` , where:

- **width** is the minimum number of characters to be returned by the function. If the value to be returned is shorter than this, it is padded either with blank spaces (" ") or zeroes ("0"). Refer to the following examples.

  The value is not shortened even if the result is longer than the specified width.

  Applicable for flags `d, x, X, o, b, f, e, E, g, G, s, c` and `h`.

- **.precision** is the number of decimal places for a floating-point number.

  Applicable for flags `f, e, E, g` and `G`.

- **Format** is the specific format:

Format	Description
d	Decimal
x	Hexadecimal (alphabetic characters in lowercase)
X	Hexadecimal (alphabetic characters in uppercase)
o	Octal
b	Binary
f	Floating-point
e	Scientific notation (e in lowercase)
E	Scientific notation (E in uppercase)
g	Rounded (e in lowercase, if applicable)
G	Rounded (E in uppercase, if applicable)
s	String (i.e., no change in number)
c	ASCII character (i.e., the numeric value is interpreted as an ASCII character code)
h	Hours (hh:mm:ss)

Alternatively, the format can be set using the syntax "##.###", where the numeric value is rounded to the specified number of decimal places.

numValue

The numeric value to be formatted.

optStrDecimalMark

The character used as the decimal mark, which separates the integer and fractional parts of the numerical value.

This is an optional parameter; if no value is specified, then the default is a period (.).

For example: "123.45"

optStrThousandSep

The character used as the thousands separator, which separates the hundreds and thousands digits of the numerical value.

This is an optional parameter; if no value is specified, then the default is a comma (,).

For example: "12,345"

Returned value

This function returns a string that contains the formatted numeric value.

Notes

"Format" is similar to the "printf" function in other programming languages and provides most similar formatting options. However, unlike "printf", "Format" can be used to format only one numeric value at a time.

This function is particularly useful for formatting values to be output in reports.

Examples

Expression	Returned value
`Format("%d", 12.34)`	`12`
`Format("%04d", 12.34)`	`0012`
`Format("%4d", 12.34)`	`  12`

Expression	Returned value
`Format( "%x", 26 )`	`1a`
`Format( "%04x", 26 )`	`001a`
`Format( "%4x", 26 )`	`  1a`

Expression	Returned value
`Format( "%X", 26 )`	`1A`
`Format( "%04X", 26 )`	`001A`
`Format( "%4X", 26 )`	`  1A`

Expression	Returned value
`Format( "%o", 16 )`	`20`
`Format( "%04o", 16 )`	`0020`
`Format( "%4o", 16 )`	`  20`

Expression	Returned value
Format("%b",2)	10
Format("%04b",2)	0010
Format("%4b",2)	0010

Expression	Returned value
Format("%0.1f", 12.34)	12.3
Format("%06.1f", 12.34)	0012.3
Format("%6.1f", 12.34)	12.3

Expression	Returned value
Format("%e", 12.34)	1.234000e+001
Format("%0.1e", 12.34)	1.2e+001
Format("%09.1e", 12.34)	01.2e+001
Format("%9.1e", 12.34)	1.2e+001

Expression	Returned value
Format("%E", 12.34)	1.234000E+001
Format("%0.1E", 12.34)	1.2E+001
Format("%09.1E", 12.34)	01.2E+001
Format("%9.1E", 12.34)	1.2E+001

Expression	Returned value
Format("%0.1g", 12.34)	1e+001
Format("%0.2g", 12.34)	12
Format("%0.3g", 12.34)	12.3
Format("%05.3g", 12.34)	012.3
Format("%5.3g", 12.34)	12.3

Expression	Returned value
Format("%0.1G", 12.34)	1E+001
Format("%0.2G", 12.34)	12
Format("%0.3G", 12.34)	12.3
Format("%05.3G", 12.34)	012.3
Format("%5.3G", 12.34)	12.3

Expression	Returned value
`Format( "%s", 12.34 )`	`12`
`Format( "%04s", 12.34 )`	`0012`
`Format( "%4s", 12.34 )`	`  12`

Expression	Returned value
`Format( "%c", 97 )`	`a`
`Format( "%4c", 97 )`	`   a`
`Format( "%04c", 97 )`	`000a`

Expression	Returned value
`Format( "%h", 30 )`	`00:00:30`
`Format( "%h", 60 )`	`00:01:00`
`Format( "%h", 90 )`	`00:01:30`
`Format( "%h", 3600 )`	`01:00:00`

Expression	Returned value
`Format( "##.#", 26.56789 )`	`26.6`
`Format( "#.##", 26.56789 )`	`26.57`
`Format( "##.##", 26.56789 )`	`26.57`

19.5.6 GetBit

"GetBit" is an internal scripting function that determines the value of a single bit in a numeric value.

Function	Group	Execution	Windows	Embedded	Thin Client
GetBit	Arithmetic	synchronous	supported	supported	supported

Syntax

```
GetBit( tagName, numBitNumber )
```

Inputs

tagName

The name of an Integer tag from which the bit value is called.



To directly specify the name of a tag instead of using the value of the tag, the tag name has to be enclosed in quotation marks.

For example, `GetBit( "Second", 1 )`.

numBitNumber

A numeric tag or value specifying the position (0…31) of the bit to call.

Returned value Returns the value ("0" or "1") of the specified bit.

Notes Individual bits in Integer tags can also be read and written by means of the "Bit field" (variable attribute).

For example, enter "Second->b0" to access the LSB (Least Significant Bit of the Second tag) and "Second->b31" to access the MSB (Most Significant Bit of the Second tag).

Examples

Expression	Returned value
`GetBit( "numSource", 4 )`	If the tag numSource holds a value of 15 (2#0000 1111), then this function returns `0`
`GetBit( "numSource", 1 )`	If the tag numSource holds a value of 19 (2#0001 0011), then this function returns `1`

19.5.7 Mod

"Mod" is an internal scripting function that returns the remainder from a division.

Function	Group	Execution	Windows	Embedded	Thin Client
Mod	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Mod( numNumerator, numDenominator )
```

Inputs

numNumerator

Integer or Real tag containing the numerator of the function (dividend).

numDenominator

Integer or Real tag containing the denominator of the function (divisor).

Returned value

Returns the remainder (as a real number) after dividing "numNumerator" by "numDenominator".



Use the Div, page 612 function to receive the whole-number dividend of the operation.

Examples

Expression	Returned value
`Mod( 50, 4 )`	`2`
`Mod( 16, 4 )`	`0`
`Mod( 100, 8.2 )`	`1.600`

19.5.8 Pow

Pow is an internal scripting function that calculates the power of a base.

Function	Group	Execution	Windows	Embedded	Thin Client
Pow	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Pow( numBase, numExponent )
```

Inputs

numBase

Integer or Real tag containing the value of the base of the function.

numExponent

Returns the result of the exponential function.

Returned value Returns the power of the base with the exponent.

Examples

Expression	Returned value
`Pow( 2, 3 )`	8
`Pow( 10, 4 )`	10000

19.5.9 ResetBit

Resets a single bit in an Integer tag to 0.

Function	Group	Execution	Windows	Embedded	Thin Client
ResetBit	Arithmetic	synchronous	supported	supported	supported

Syntax

```
ResetBit( tagName, numBitNumber )
```

Inputs

tagName

The name of an Integer tag in which the bit value is to be reset.

☞ To directly specify the name of a tag instead of using the value of the tag, the tag name has to be enclosed in quotation marks.

For example, `ResetBit( "Second", 1 )`.

numBitNumber

A numeric tag or value specifying the position (0…31) of the bit to reset.

Returned value

Value	Description
0	No error
1	Invalid parameter
2	Tag does not exist

☞ Individual bits in Integer tags can also be read and written by means of the "Bit field" (variable attribute).

For example, enter **Second->b0** to access the LSB (Least Significant Bit of the Second tag) and **Second->b31** to access the MSB (Most Significant Bit of the Second tag).

Examples

Expression	Returned value
`ResetBit( "numSource", 4 )`	If the tag numSource held a value of 16 (2#000**1** 0000), then the function returns `0` and numSource holds a new value of 0 (2#000**0** 0000
`ResetBit( "numSource", 1 )`	If the tag numSource held the value 19 (2#0001 00**1**1), then the function returns `0` and numSource holds a new value of 17 (2#0001 00**0**1)

19.5.10 Round

Rounds "numValue" to the nearest integer.

Function	Group	Execution	Windows	Embedded	Thin Client
Round	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Round( numValue )
```

Inputs

numValue

A Real tag that holds the value to be rounded.

Returned value Returns the integer result of the round function.

Examples

Expression	Returned value
`Round( "345.87" )`	`346`
`Round( "65.323" )`	`65`

19.5.11 SetBit

Sets a single bit in an Integer tag to 1.

Function	Group	Execution	Windows	Embedded	Thin Client
SetBit	Arithmetic	synchronous	supported	supported	supported

Syntax

```
SetBit( tagName, numBitNumber )
```

Inputs

tagName

The name of an Integer tag where the bit value is set.



To directly specify the name of a tag instead of using the value of the tag, the tag name has to be enclosed in quotation marks.

For example, `SetBit( "Second", 1 )`.

numBitNumber

A numeric tag or value specifying the position (0…31) of the bit to set.

Returned value

Value	Description
0	No error
1	Invalid parameter
2	Tag does not exist

Notes

Individual bits in Integer tags can also be read and written by means of the "Bit field" (variable attribute).

For example, enter "Second->b0" to access the LSB (Least Significant Bit of the Second tag) and "Second->b31" to access the MSB (Most Significant Bit of the Second tag).

Examples

Expression	Returned value
`SetBit( "numSource", 4 )`	If the tag numSource held a value of 0, then this function returns 0 and numSource holds a new value of 16.
`SetBit( "numSource", 1 )`	If the tag numSource held the value 17, then this function returns 0 and numSource holds a new value of 19

19.5.12 Sqrt

Calculates the square root of "numValue".

Function	Group	Execution	Windows	Embedded	Thin Client
Sqrt	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Sqrt( numValue )
```

Inputs

numValue
Integer or Real tag to be square rooted.

Returned value

Returns the square root of the value in the "numValue" tag.



If "numValue" has a negative value, this function returns the value 0 and sets the quality of the returned tag to "BAD" (quality display in the Database Spy).

Examples

Expression	Returned value
`SQRT( 25 )`	5
`SQRT( 67 )`	8.185353

19.5.13 Swap16

Swaps the two lower bytes of a tag.

Function	Group	Execution	Windows	Embedded	Thin Client
Swap16	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Swap16( numValue )
```

Inputs

numValue
Integer tag that holds the numeric value of the bytes to be swapped.

Returned value Returns the numeric value after swapping the bytes.

Examples

Expression	Returned value
`Swap16( 16 ) // 2#0000 0000 0001 0000`	`4096 (2#0001 0000 0000 0000)`
`Swap16( 43760 ) // 2#1010 0101 1111 0000`	`61610 (2#1111 0000 1010 0101)`

19.5.14 Swap32

Swaps two data words of a tag.

Function	Group	Execution	Windows	Embedded	Thin Client
Swap32	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Swap32( numValue )
```

Inputs

numValue
Integer tag that holds the numeric value of the data words to be swapped.

Returned value Returns the numeric value after swapping the data words.

Examples

Expression	Returned value
`Swap32( 16 )` 2#0000 0000 0000 0000 0000 0000 0001 0000	`1048576` 2#0000 0000 0001 0000 0000 0000 0000 0000
`Swap32( 246333120 )` 2#1010 1010 1010 1010 1111 1111 0000 0000	`-1094709586` 2#1111 1111 0000 0000 1010 1010 1010 1010

19.5.15 Trunc

Returns the whole-number portion of a real numeric value.

Function	Group	Execution	Windows	Embedded	Thin Client
Trunc	Arithmetic	synchronous	supported	supported	supported

Syntax

```
Trunc( numValue )
```

Inputs

numValue
Real tag of which the whole-number portion is to be calculated.

Returned value Returns the whole-number portion of the real numeric value of "numValue", i.e., all decimal places are truncated without rounding.

Examples

Expression	Returned value
`Trunc( 234.987 )`	`234`
`Trunc( -3465.9 )`	`-3465`

19.6 Statistical Functions

19.6.1 Statistical Functions – General Information

These functions are used to calculate specific statistics – such as average, maximum, and minimum – from two or more numeric values.

Function	Description
Avg, page 622	Calculates the average value of a set of numbers.
Max, page 623	Returns the maximum value of a set of numbers.
Min, page 624	Returns the minimum value of a set of numbers.
Rand, page 625	Generates a random number between 0 and 1.

19.6.2 Avg

Returns the average value of a set of numbers.

Function	Group	Execution	Windows	Embedded	Thin Client
Avg	Statistical	synchronous	supported	supported	supported

Syntax

```
Avg( numValue1, numValue2, … , numValueN )
Avg( "tagArray", numSample, optNumIgnore )
```

This function has two formats:

- If the first parameter is a numeric tag or value, the

  `Avg(numValue1, numValue2,…, numValueN)` format has to be used.
- If the first parameter is an array tag in quotation marks or a String tag, the

  `Avg("tagArray", numSample, optNumIgnore)` format has to be used.

Inputs

numValue (1…N)

Integer or Real tags containing the numbers to be averaged.

tagArray

Name of an array tag (Real or Integer) containing the values to be averaged.

numSample

The number of array elements to be averaged.

optNumIgnore

Optional Integer or Real tag containing the value to be ignored when calculating the average.

Returned value Returns the average of the values.

Examples

Tag name	Expression
Tag	Avg(1, 2.34, 5, 7, 4, 8, 9.4) // Returned value = 5.248571
Tag	Avg(1, 5, -9, 0, 5, 3) // Returned value = 0.833333

Tag name	Expression
Tag	`Avg( "tagArray[1]", 3 )` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 30
Tag	`Avg( "tagArray[1]", 3, 10 )` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 40

19.6.3 Max

Returns the maximum value of a set of numbers.

Function	Group	Execution	Windows	Embedded	Thin Client
Max	Statistical	synchronous	supported	supported	supported

Syntax

```
Max( numValue1, numValue2, … , numValueN )
Max( "tagArray", numSample, optNumIgnore )
```


This function has two formats:

- If the first parameter is a numeric tag or value, the `Max(numValue1, numValue2,…, numValueN)` format is used.
- If the first parameter is an array tag in quotation marks or a string tag, the `Max("tagArray", numSample, optNumIgnore)` format is used.

Inputs

numValue (1…N)

Integer or Real tags containing the numbers to be analyzed.

tagArray

Name of an array tag (Real or Integer) containing the values to be analyzed.

numSample

The number of array elements to be analyzed.

optNumIgnore

Integer or Real tags containing the value to be ignored in the analysis.

Returned value Returns the maximum value of the set.

Examples

Tag name	Expression
Tag	`Max( 1, 2.34, 5, 7, 4, 8, 9.4)` // Returned value = 9.4
Tag	`Max( 1, 5, -9, 0, 5, 3)` // Returned value = 5
Tag	`Max( "tagArray[1]", 3)` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 60
Tag	`Max( "tagArray[1]", 3, 10)` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 60

19.6.4 Min

Returns the minimum value of a set of numbers.

Function	Group	Execution	Windows	Embedded	Thin Client
Min	Statistical	synchronous	supported	supported	supported

Syntax

```
Min( numValue1, numValue2, … , numValueN )
Min( "tagArray", numSample, optNumIgnore )
```



This function has two formats:

- If the first parameter is a numeric tag or value, the `Min( numValue1, numValue2, … , numValueN)` format is used.
- If the first parameter is an array tag in quotation marks or a String tag, the `Min( "tagArray", numSample, optNumIgnore )` format is used.

Inputs

numValue (1…N)

Integer or Real tags containing the numbers to be analyzed.

tagArray

Name of an array tag (Real or Integer) containing the values to be analyzed.

numSample

The number of array elements to be analyzed.

optNumIgnore

Integer or Real tags containing a value to be ignored in the analysis.

Returned value

Returns the minimum value of the set.

Examples

Tag name	Expression
Tag	`Min( 1, 2.34, 5, 7, 4, 8, 9.4 )` // Returned value = 1
Tag	`Min( 1, 5, -9, 0, 5, 3 )` // Returned value = -9

Tag name	Expression
Tag	`Min( "tagArray[1]", 3 )` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 10
Tag	`Min( "tagArray[1]", 3, 10 )` // If tagArray[1]=10, tagArray[2]=20 and tagArray[3]=60, then the Returned Value = 20

19.6.5 Rand

Generates a random number between 0 and 1.

Function	Group	Execution	Windows	Embedded	Thin Client
Rand	Statistical	synchronous	supported	supported	supported

Syntax

```
Rand()
```

This function does not have any parameters.

Returned value Returns a real number between 0 and 1.

Examples

Tag name	Expression
Tag	Rand() // Returned value = ?, Where: 0<?<1

19.7 Logarithmic Functions

19.7.1 Logarithmic Functions – General Information

These functions are used to perform logarithmic operations on numeric values.

Function	Description
Exp, page 625	Calculates the value of e (2.718282) raised to the power of "numValue".
Log, page 626	Calculates the natural log of "numValue".
Log10, page 626	Calculates the log base 10 of "numValue".

19.7.2 Exp

Calculates the value of Euler's number "e" (2.718282) raised to the power of "numValue".

Function	Group	Execution	Windows	Embedded	Thin Client
Exp	Logarithmic	synchronous	supported	supported	supported

Syntax

```
Exp(numValue)
```

Inputs

numValue
Integer or Real tag containing the value of the exponent of this function.

Returned value Returns the value of "e^(numValue)".

Examples

Tag name	Expression
Tag	Exp(1) // Returned value = 2.718282
Tag	Exp(5.25896) // Returned value = 192.281415

19.7.3 Log

Calculates the natural logarithm of "numValue".

Function	Group	Execution	Windows	Embedded	Thin Client
Log	Logarithmic	synchronous	supported	supported	supported

Syntax

```
Log(numValue)
```

Inputs

numValue

Integer or Real tag from which the natural logarithm is taken.

Returned value

Returns the value of "ln(numValue)".

☞ If "numValue" has a negative value, this function returns the value 0 and sets the quality of the returned tag to "BAD" (quality display in the Database Spy).

Examples

Tag name	Expression
Tag	Log(2.718282) // Returned value = 1
Tag	Log(100) // Returned value = 4.605170

19.7.4 Log10

Calculates the logarithm "numValue" to the base of 10.

Function	Group	Execution	Windows	Embedded	Thin Client
Log10	Logarithmic	synchronous	supported	supported	supported

Syntax

```
Log10(numValue)
```

Inputs

numValue

Integer or Real tag, from which the logarithm to the base of 10 is taken.

Returned value

Returns the value of "ln(numValue)".

☞ If "numValue" has a negative value, this function returns the value 0 and sets the quality of the returned tag to "BAD" (quality display in the Database Spy).

Examples

Tag name	Expression
Tag	Log10(1000) // Returned value = 3
Tag	Log10(43.05) // Returned value = 1.633973

19.8 Logical functions

19.8.1 Logical functions – General Information

These functions are used to perform logical operations (e.g., if/then, true/false) on tags and expressions.

Function	Description
False, page 628	Determines whether the specified tag or expression is logically false.
If, page 628	Evaluates a specified tag/expression to determine whether it is logically true or false, and then it returns a corresponding value.
Toggle, page 629	Returns the toggled value from the contents of "numValue" tag.
True, page 630	Determines whether the specified tag or expression is logically true.

19.8.2 Logical Operators

The internal scripting language supports the following logic operators.

Logical operators

Operator	Usage	Description
AND	A AND B	TRUE if A and B are both TRUE
OR	A OR B	TRUE if A is TRUE, or B is TRUE, or both
XOR	A XOR B	TRUE if A is TRUE, or B is TRUE, but not both
NOT	NOT A	TRUE if A is FALSE
=	X = Y	TRUE if X is equal to Y
>	X > Y	TRUE if X is greater than Y
>=	X >= Y	TRUE if X is greater than or equal to Y
<	X > Y	TRUE if X is less than Y
<=	X <= Y	TRUE if X is less than or equal to Y
<>	X <> Y	TRUE if X is not equal to Y

&	X & Y	Bitwise AND:
		`0101 (decimal 5)`
		`AND 0011 (decimal 3)`
		`= 0001 (decimal 1)`

|	X | Y	Bitwise OR:
		`0101 (decimal 5)`
		`OR 0011 (decimal 3)`
		`= 0111 (decimal 7)`

^	X ^ Y	Bitwise XOR:
		`0101 (decimal 5)`
		`XOR 0011 (decimal 3)`
		`= 0110 (decimal 6)`

~	~ X	Bitwise NOT:
		`NOT 0101 (decimal 5)`
		`= 1010 (decimal 10)`

Operator	Usage	Description
>> n	X >> Y	Rotate n bits to right: 0110 (decimal 6) ROTATE RIGHT = 0011 (decimal 3)
<< n	X << Y	Rotate n bits to left: 0110 (decimal 6) ROTATE LEFT = 1100 (decimal 12)



For more complex logical operators, the Logical, page 627 and Loop, page 846 functions are to be used.

19.8.3 False

Determines whether the specified tag or expression is logically false.

Function	Group	Execution	Windows	Embedded	Thin Client
False	Logical	synchronous	supported	supported	supported

Syntax

```
False(TagOrExpression)
```

Inputs

TagOrExpression
Tag or expression to be used in the function.

Returned value

0	If the tag or expression is "not" logically false
1	If the tag or expression is logically false



This function is useful if an actual value of 0 needs to be returned in case the expression returns a value other than 0.

Examples

Tag name	Expression
Tag	False(1) // Returned value = 0
Tag	False(5 < 2) // Returned value = 1

19.8.4 If

Determines whether the contents of "numExpression" are logically true and returns the value of "numThen" or "optNumElse" accordingly.

Function	Group	Execution	Windows	Embedded	Thin Client
If	Statistical	synchronous	supported	supported	supported

Syntax

```
If( numExpression, numThen, optNumElse )
```

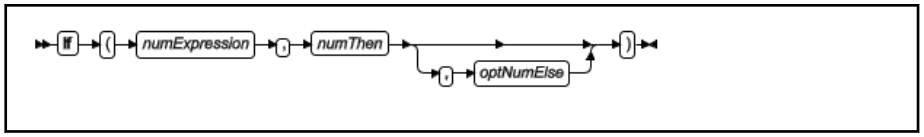


Fig. 19-2: Syntax

Inputs

numExpression

Tag or expression used as the condition in the function.

numThen

Tag or expression used if the condition is logically true (i.e., not 0).

optNumElse

Optional tag or expression used if the condition is logically false (i.e., 0).

This parameter is optional; see "Returned value" below.

Returned value

numThen	If the numExpression is logically true
optNumElse	If the numExpression is logically false
No value returned	If the numExpression is logically false and there is no optNumElse in the function

Notes

"numExpression" can be a combination of logic statements (e.g., AND, OR, NOT). For example:

```
If(TagA>TagB AND TagA=10,1,0)
```

"numThen" can be another function, including the function If. Therefore, you can use the functions in cascade. For example:

```
If(TagA>TagB,If(TagA<TagC,1,2),3)
```

Examples

Evaluate the expression "5 > 4", and then return the corresponding value (i.e., return 10, because the expression is logically true):

```
If(5>4,10,6)
```

Evaluate the expression "5 < 2", and then return the corresponding value (i.e., return 2, because the expression is logically false):

```
If(5<2,0,2)
```

Evaluate the expression "3 = 9", and then return the corresponding value (i.e., return no value, because the expression is logically false and optNumElse is not specified):

```
If(3=9,67)
```

19.8.5 Toggle

Returns the toggled value from the contents of "numValue" tag.

Function	Group	Execution	Windows	Embedded	Thin Client
Toggle	Logical	synchronous	supported	supported	supported

Syntax

```
Toggle( numValue )
```

Inputs

numValue
Boolean tag containing the value to be toggled.

Returned value Numeric result (0 or 1) of the value to be toggled.



This function does not actually change the value of the tag. However, the function can be used in an expression or command changing the value of the tag.

Examples

Tag name	Expression
Tag	Toggle(MyBoolTag) // Returned value = 1 if MyBoolTag value equals 0, or 0 if MyBoolTag value equals 1
Tag	Toggle(numValue) // Returned value = toggled value of the number in the numValue tag

19.8.6 True

Determines whether the specified tag or expression is logically true.

Function	Group	Execution	Windows	Embedded	Thin Client
True	Logical	synchronous	supported	supported	supported

Syntax

```
True(TagOrExpression)
```

Inputs

TagOrExpression
Tag or expression to be used in the function.

Returned value

0	If the tag or expression is not logically true
1	If the tag or expression is logically true



This function is useful if an actual value of 1 needs to be returned in case the expression returns a value other than 0.

Examples

Tag name	Expression
Tag	True(1) // Returned value = 1
Tag	True(5 < 2) // Returned value = 0

19.9 String Functions

19.9.1 String Functions – General Information

Diese Funktionen dienen zum Verändern von Zeichenketten oder deren Konvertierung in numerische Werte.

Function	Description
Asc2Str, page 631	Converts one or more Unicode character codes to a string.
CharToValue, page 632	Converts a string to Unicode character codes and then stores those values in an integer array.
CharToValueW, page 633	Converts a string to Unicode character codes, combines each two codes into a double-byte word, and then stores those values in an integer array.
ClassMembersToStrVector, page 633	Transfers values from a Class tag to an Array tag.
NCopy, page 634	Copies a defined section of a larger string.
Num, page 635	Converts a string into a float.
Str, page 636	Converts a number into a string.
Str2Asc, page 636	Converts a character to its corresponding Unicode character code.
StrCompare, page 636	Compares two strings to see if they are identitical.
StrCompareNoCase, page 637	Compares two strings to see if they are identitical, ignoring the case of letters (i.e., the lower-case "a" is considered to have the same value as the upper-case "A").
StrFromInt, page 638	Converts an integer into its string representation in another base number system, such as binary (base-2) or octal (base-8).
StrFromReal, page 638	Converts a real numerical value to a string value, in either floating-point or exponential notation.
StrFromTime, page 639	Converts a timestamp from UTC standard notation into a formatted string, adjusted to reflect the Time Zone setting in the Control Panel of the local computer.
StrGetElement, page 640	Gets a specific element from a string source.
StrLeft, page 641	Copies the first characters of a larger string.
StrLen, page 641	Determines the length of a string.
StrLower, page 642	Converts a string to all lower case characters.
StrRChr, page 642	Isolates the final occurrence of a character sequence within a string.
StrRight, page 642	Copies the last characters in a larger string.
StrSetElement, page 643	Sets a specific element in a string source.
StrStr, page 643	Isolates the first occurrence of a character sequence within a string.
StrStrPos, page 644	Finds the first occurrence of a character within a string.
StrTrim, page 644	Removes unwanted spaces from a string.
StrTrimAll, page 645	Eliminates a specific char from the whole string.
StrUpper, page 646	Converts a string to all uppercase characters.
ValueToChar, page 646	Converts an integer array of Unicode character codes to a string.
ValueWToChar, page 647	Converts an integer array of Unicode character codes to a string, where each value in the array is a double-byte word.

19.9.2 Asc2Str

This function converts one or more Unicode character codes to a string.

Function	Group	Execution	Windows	Embedded	Thin Client
Asc2Str	Character string	synchronous	supported	supported	supported

Syntax

```
Asc2Str( numChar1, numChar2, … , numCharN )
```

Inputs

numChar (1-N)

A Unicode character code (in decimal).

Returned value

Returns a string with the converted codes.



Although the name of the function indicates that it only supports ASCII characters, it can be used for the entire Unicode character set.

Examples

Tag name	Expression
Tag	Asc2Str(65) // Returned value = "A"
Tag	Asc2Str(65, 66, 67) // Returned value = "ABC"
Tag	Asc2Str(Array[0], Array[1], Array[2]) // Returned value = "ABC"

19.9.3 CharToValue

This function converts a string to Unicode character codes and stores those values in an Integer array.

Function	Group	Execution	Windows	Embedded	Thin Client
CharToValue	Character string	synchronous	supported	supported	supported

Syntax

```
CharToValue("tagString","tagArray")
```

Inputs

tagString

The name of the String tag whose value is to be converted.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

tagArray

The name of the Integer array that is to receive the converted values. If no array is specified, the default value is 0.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns the number of array elements used, which has to be equal to the number of characters in the string.

Examples

If StrTag= "ABC", then Array[0]= 65, Array[1]= 66, and Array[2]= 67:

```
CharToValue( "StrTag", "Array" )
```

If StrTag= "ABC", then Array[10]= 65, Array[11]= 66, and Array[12]= 67:

```
CharToValue( "StrTag", "Array[10]" )
```

19.9.4 CharToValueW

This function converts a string to Unicode character codes, combines two codes into a double-byte word and stores those values in an Integer array.

Function	Group	Execution	Windows	Embedded	Thin Client
CharToValueW	Character string	synchronous	supported	supported	supported

Syntax

```
CharToValueW("tagString","tagArray")
```

Inputs

tagString

The name of the String tag whose value is to be converted.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

tagArray

The name of the Integer array that is to receive the converted values. If no array is specified, the default value is 0.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns the number of array elements used, which is equal to "half" the number of characters in the string.

☞ Because of how each two character codes are combined into single value, this function only supports Unicode character codes 0 through 255. For character codes greater than 255 or if double-byte words are not needed, the CharToValue page 632 function is to be used.

Examples

If StrTag= "Studio", then Array[0]= 29779 ("St"), Array[1]= 25717 ("ud"), and Array[2]= 28521 ("io"):

```
CharToValue( "StrTag", "Array" )
```

If StrTag= "Studio", then Array[10]= 29779 ("St"), Array[11]= 25717 ("ud"), and Array[12]= 28521 ("io"):

```
CharToValue( "StrTag", "Array[10]" )
```

19.9.5 ClassMembersToStrVector

Transfers values from a class tag to an array tag.

Function	Group	Execution	Windows	Embedded	Thin Client
ClassMember-sToStrVector	Character string	synchronous	supported	supported	supported

Syntax

```
ClassMembersToStrVector( "strClassTag" , numStartPos, numNumPos, "strArrayTag", optBooStartPosTarget )
```

Inputs

strClassTag

String value containing the class tag name.

numStartPos

Start position (array index) of "strClassTag" .

numNumPos

Number of positions (array indexes) to be transferred from "strClassTag" .

strArrayTag

String value containing the array tag that is to receive the values from "strClassTag" .

optBooStartPosTarget

Start position (array index) of "strArrayTag". If omitted, the default value 1 is used.

Returned value

−6	Array size of "strClassTag" is not big enough for "numStartPos"
−5	"strClassTag" is not a Class tag
−4	"strClassTag" is not found
−3	"strArrayTag" is not found
−2	Invalid data type of the parameters
−1	Invalid number of parameters
0	Transferred successfully

☞ If "strClassTag" has more than one element, the value of each class element is transferred to "strArrayTag". Therefore, it is important to make sure that the array size of "strArrayTag" is big enough to receive all values from "strClassTag".

Examples

Tag name	Expression
Tag	ClassMembersToStrVector("Classtag", 5, 3, "Arraytag")
Tag	ClassMembersToStrVector("Classtag", 5, 3, "Arraytag", 0)
Tag	ClassMembersToStrVector (TagName, 0, 1, ArrayName)

19.9.6 NCopy

Copies a defined section of a larger string.

Function	Group	Execution	Windows	Embedded	Thin Client
NCopy	Character string	synchronous	supported	supported	supported

Syntax

```
NCopy( strSource, numStartChar, numQtdChar )
```

Inputs

strSource

The source string.

numStartChar

Integer tag containing a number corresponding to the first character being copied.

numQtdChar

The number of characters to be copied.

Returned value Returns a string that is part of the source string (as defined by the function).

Examples

Tag name	Expression
Tag	NCopy("Studio version 7.0", 7, 7) // Returned value = "version"
Tag	NCopy("Technical Reference", 0, 9) // Returned value = "Technical"

☞ The value 0 is assigned to the first character in the string.

19.9.7 Num

Converts a string into a floating point number.

Function	Group	Execution	Windows	Embedded	Thin Client
Num	Character string	synchronous	supported	supported	supported

Syntax

```
Num( strValue )
```

Inputs

strValue

The number of characters to be converted into the floating point number format.

Returned value Returns the number (formerly in a string format) as floating point number.

Examples

Tag name	Expression
Tag	Num("321654.987") // Returned value = 321654.987
Tag	Num("5.6589626246") // Returned value = 5.6589626246

☞ The string for the floating point number cannot use characters other than the numbers (0..9) and a decimal point (.). Otherwise, the function returns the value 0.0.

19.9.8 Str

Converts a number into a string.

Function	Group	Execution	Windows	Embedded	Thin Client
Str	Character string	synchronous	supported	supported	supported

Syntax

```
Str( numValue )
```

Inputs

numValue

Integer or floating point number tag containing a number to be converted to a string.

Returned value Returns the string in a floating point number format.

Examples

Tag name	Expression
Tag	Str(321654.987) // Returned value = "321654.987"
Tag	Str(5.65896246) // Returned value = "5.658962"

19.9.9 Str2Asc

This function converts a character to its original Unicode character code.

Function	Group	Execution	Windows	Embedded	Thin Client
Str2Asc	Character string	synchronous	supported	supported	supported

Syntax

```
Str2Asc(strChar)
```

Inputs

strChar

The character to be converted.

Returned value Returns the Unicode character code (in decimal) for the specified character.



Although the name of the function indicates that it only supports ASCII characters, it can be used for the entire Unicode character set.

Examples

Tag name	Expression
Tag	Str2Asc("C") // Returned value = 67
Tag	Str2Asc("o") // Returned value = 111

19.9.10 StrCompare

Compares two strings to check if they are identical.

Function	Group	Execution	Windows	Embedded	Thin Client
StrCompare	Character string	synchronous	supported	supported	supported

Syntax

```
StrCompare( strValue1, strValue2 )
```

Eingaben

strValue1

A string or a tag of type String, page 111. This is the first string to be compared.

strValue2

A string or a tag of type String, page 111. This is the second string to be compared.

Returned value

−1	The value of `strValue1` is smaller than the value of `strValue2`.
0	`strValue1` and `strValue2` are identical.
1	The value of `strValue1` is greater than the value of `strValue2`.

Examples

Tag name	Expression
Tag	StrCompare("Text1", "Text2") // Returned value = −1
Tag1="Text" Tag2="Text"	StrCompare(Tag1, Tag2) // Returned value = 0
Tag1="Text1" Tag2="Text2"	StrCompare(Tag1, Tag2) // Returned value = −1

19.9.11 StrCompareNoCase

Compares two strings to check if they are identical. The case of characters is ignored (i.e., a lower-case "a" has the same value as an upper-case "A").

Function	Group	Execution	Windows	Embedded	Thin Client
StrCompareNo-Case	Character string	synchronous	supported	supported	supported

Syntax

```
StrCompareNoCase( strValue1, strValue2 )
```

Inputs

strValue1

A string or a tag of type String, page 111. This is the first string to be compared.

strValue2

A string or a tag of type String, page 111. This is the second string to be compared.

Returned value

−1	The value of `strValue1` is less than the value of `strValue2`.
0	`strValue1` and `strValue2` are identical
1	The value of `strValue1` is greater than the value of `strValue2`

Examples

Tag name	Expression
Tag	StrCompareNoCase("Text1", "TEXT1") // Returned value = 0
Tag1="Text1" Tag2="TEXT1"	StrCompareNoCase(Tag1, Tag2) // Returned value = 0

19.9.12 StrFromInt

Converts an integer into its respective string in another base number system, such as binary (base 2) or octal (base 8).

Function	Group	Execution	Windows	Embedded	Thin Client
StrFromInt	Character string	synchronous	supported	supported	supported

Syntax

```
StrFromInt( numValue, numBase )
```

Inputs

numValue

The numeric value to be converted into a string.

numBase

The base number system to convert into.

Returned value

This function returns a string of the given integer in the specified base number system. The returned value can be stored in any String-type tag.



A real number can be specified instead of an integer. However, only the whole part of the number is converted. To convert the entire real number, use the StrFromReal, page 638 function instead.

If the base does not need to be changed, use the Str, page 636 function instead.

Examples

Tag name	Expression
Tag	StrFromInt(26, 2) // Returned value = "11010"
Tag	StrFromInt(26, 8) // Returned value = "32"

19.9.13 StrFromReal

Converts a real numeric value to a string, either as floating point number or in exponential notation.

Function	Group	Execution	Windows	Embedded	Thin Client
StrFromReal	Character string	synchronous	supported	supported	supported

Syntax

```
StrFromReal(numValue,numPrecision,{ strType | f | e | E })
```

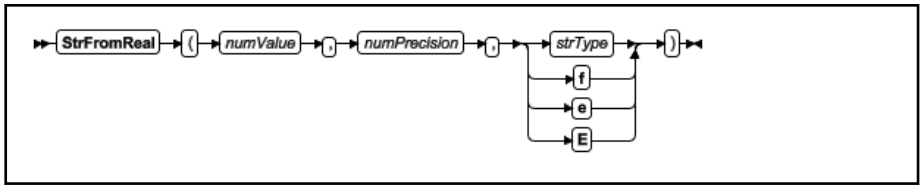


Fig. 19-3: Syntax

Inputs

numValue

The numeric value to be converted.

numPrecision

The number of decimal places to be shown in the resulting string. Please note that the value is rounded instead of truncated.

strType

A single-character code that specifies how the resulting string is to be formatted, as described in the following table:

Value of strType	Description
f	Formatted in floating-point notation
e	Formatted in exponential notation with a lower-case "e"
E	Formatted in exponential notation with an upper-case "E"

Returned value

This function returns a string representation of the given numeric value with the specified precision and notation.

Examples

StrFromReal(263.355, 2, "f")

…returns a string value of "263.36".

StrFromReal(263.355, 2, "e")

…returns a string value of "2.63e+002".

StrFromReal(263.355, 2, "E")

…returns a string value of "2.63E+002".

19.9.14 StrFromTime

Converts a time stamp from UTC standard notation into a formatted string, adjusted to reflect the Time Zone setting in the Control Panel of the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
StrFromTime	Character string	synchronous	supported	supported	supported

Syntax

```
StrFromTime( numUTCTime, numType )
```

Inputs

numUTCTime

An integer or a tag of type Integer, page 111. A time stamp in UTC standard notation.

numType

An integer or a tag of type Integer, page 111. Specifies the format of the resulting string, as described in the following table:

Value of numType	Description
1	Displays the "date" in the same format that is selected in the Control Panel on the local computer.
2	Displays the "time" in the same format selected in the Control Panel of the local computer
3	Displays a standard 24-character string that shows both date and time
4	Displays the abbreviated name of the day of the week
5	Displays the full name of the day of the week

Returned value

This function returns a string representation of the given time stamp with the specified formatting. The returned value can be stored in any tag of type String, page 111,.



The Coordinated Universal Time (UTC) standard counts the number of seconds elapsed since 12:00 AM GMT on January 1, 1970. Each day consists of 86,400 seconds.

Examples

The examples below refer to a computer set to Eastern Standard Time (or UTC −05:00).

Tag name	Expression
Tag	StrFromTime(86400, 1) // Returned value = "1/1/70"
Tag	StrFromTime(86400, 2) // Returned value = "07:00:00 PM"
Tag	StrFromTime(86400, 3) // Returned value = "Thu Jan 01 19:00:00 1970"
Tag	StrFromTime(86400, 4) // Returned value = "Thu"
Tag	StrFromTime(86400, 5) // Returned value = "Thursday"

19.9.15 StrGetElement

Gets a specific element from a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrGetElement	Character string	synchronous	supported	supported	supported

Syntax

```
StrGetElement( strSource, strDelimiter, numElementNumber )
```

Inputs

strSource

The source string.

strDelimiter

Character used as delimiter between the elements.

numElementNumber

Number of the element returned by the function. The first element has the number 1. The second element has the number 2 and so forth.

Returned value Returns the element (string value) retrieved from "strSource".

Examples

Tag name	Expression
Tag	StrGetElement("a|b|c", "|", 2) // returned value = "b"
Tag	StrGetElement("a,b,c", ",", 3) // returned value = "c"

19.9.16 StrLeft

Copies the first characters of a larger string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrLeft	Character string	synchronous	supported	supported	supported

Syntax

```
StrLeft( strSource, numQtdChar )
```

Inputs

strSource

The source string.

numQtdChar

The number of characters to be copied.

Returned value Returns a string containing the left-most characters in the source string.

Examples

Tag name	Expression
Tag	StrLeft("Studio version 7.0", 8) // Returned value = Studio v
Tag	StrLeft ("Technical Reference", 9) // Returned value = Technical

19.9.17 StrLen

Determines the length of a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrLen	Character string	synchronous	supported	supported	supported

Syntax

```
StrLen( strSource )
```

Inputs

strSource

The string.

Returned value Returns an integer, i.e., the number of characters in the string.

Examples

Tag name	Expression
Tag	StrLen("Studio version 7.0") // Returned value = 18
Tag	StrLen("Technical Reference") // Returned value = 19

19.9.18 StrLower

Converts a string to all lower case characters.

Function	Group	Execution	Windows	Embedded	Thin Client
StrLower	Character string	synchronous	supported	supported	supported

Syntax

```
StrLower( strSource )
```

Inputs

strSource

The string to be converted.

Returned value Returns the string, where all the characters are in lower case.

Examples

Tag name	Expression
Tag	StrLower("Studio version 7.0") // Returned value = "studio version 7.0"
Tag	StrLower("Technical Reference") // Returned value = "technical reference"

19.9.19 StrRChr

Isolates the final occurrence of a character sequence within a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrRChr	Character string	synchronous	supported	supported	supported

Syntax

```
StrRChr( strSource, strChrSequence )
```

Inputs

strSource

The source string.

strCharSequence

The reference string.

Returned value Returns a string following the last occurrence of a character within the string.

Examples

Tag name	Expression
Tag	StrRChr("Studio version 7.0", "i") // Returned value = "ion 7.0"
Tag	StrRChr("Technical Reference", "n") // Returned value = "nce"

19.9.20 StrRight

Copies the last characters in a larger string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrRight	Character string	synchronous	supported	supported	supported

Syntax

```
StrRight( strSource, numQtdChar )
```

Inputs

strSource

The source string.

numQtdChar

The number of characters to be copied.

Returned value Returns a string containing the right-most characters in a source string.

Examples

Tag name	Expression
Tag	StrRight("Studio version 7.0", 8) // Returned value = "sion 7.0"
Tag	StrRight("Technical Reference", 9) // Returned value = "Reference"

19.9.21 StrSetElement

Sets a specific element in a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrSetElement	Character string	synchronous	supported	supported	supported

Syntax

```
StrSetElement( strSource, strDelimiter, numElementNumber, strValue )
```

Inputs

strSource

The source string.

strDelimiter

Character used as delimiter between the elements.

numElementNumber

Number of the element where the string value is to be written by the function. Numbering starts at index 1.

strValue

String value to be written to the numElementNumber of the strSource String tag.

Returned value Returns the string value updated with the "strValue".

Examples

Tag name	Expression
StringTag	StrSetElement(strSource, "|", 2, "abcd")
StringTag	StrSetElement(strSource, ",", 3, "defg")

19.9.22 StrStr

Isolates the first occurrence of a character sequence within a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrStr	Character string	synchronous	supported	supported	supported

Syntax

```
StrStr( strSource, strCharSequence )
```

Inputs

strSource

The source string.

strCharSequence

The reference string.

Returned value Returns the string of characters following the first occurrence of a character within the source string.

Examples

Tag name	Expression
Tag	StrStr("Studio version 7.0", "i") // Returned value = "io version 7.0"
Tag	StrStr("Technical Reference", "n") // Returned value ="nical Reference"

19.9.23 StrStrPos

Finds the first occurrence of a character within a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrStrPos	Character string	synchronous	supported	supported	supported

Syntax

```
StrStrPos( strSource, strCharSequence )
```

Inputs

strSource

The source string.

strCharSequence

The reference string.

Returned value Returns an integer corresponding to the first occurrence of a character within the string.

Examples

Tag name	Expression
Tag	StrStrPos("Studio version 7.0", "i") // Returned value = 4
Tag	StrStrPos("Technical Reference", "a") // Returned value = 7

☞ The value 0 is assigned to the first character in the string.

19.9.24 StrTrim

Removes spaces from a string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrTrim	Character string	synchronous	supported	supported	supported

Syntax

```
StrTrim( strReference, optNumFlag )
```

Inputs

strReference

A string or tag of type String, page 111 containing the source string.

optNumFlag

An "optional" integer or tag of type Integer, page 111:

Value of optNum-Flag	Description
0	Removes all spaces from both the beginning and the end of the string
1	Removes all spaces only from the beginning of the string
2	Removes all spaces only from the end of the string
3	Removes all spaces except for single spaces between words

If no value is specified for "optNumFlag", then "0" is the default.

Returned value

This function returns a string equal to "strReference" minus the specified space characters. The returned value can be stored in any tag of type String, page 111.

Examples

Expression
StrTrim(" Studio version 7.0 ") // Returned value = "Studio version 7.0"
StrTrim(" Studio version 7.0 ", 0) // Returned value = "Studio version 7.0"
StrTrim(" Studio version 7.0 ", 1) // Returned value = "Studio version 7.0 "
StrTrim(" Studio version 7.0 ", 2) // Returned value = " Studio version 7.0"
StrTrim(" Studio version 7.0 ", 3) // Returned value = "Studio version 7.0"

19.9.25 StrTrimAll

Eliminates a specific character from the whole string.

Function	Group	Execution	Windows	Embedded	Thin Client
StrTrimAll	Character string	synchronous	supported	supported	supported

Syntax

```
StrTrimAll( strReference, optStrTrimChar )
```

Inputs

strReference

The source string.

optStrTrimChar

Character that is to be removed from the string. If this parameter is omitted, the space character is removed from the string by default.

Returned value

Returns a string similar to "strReference" minus the characters removed by the function.

Examples

Tag name	Expression
Tag	StrTrimAll("Studio version 7.0 ", " ") // Returned value = "Studioversion7.0"

19.9.26 StrUpper

Converts a string to all upper case characters.

Function	Group	Execution	Windows	Embedded	Thin Client
StrUpper	Character string	synchronous	supported	supported	supported

Syntax

```
StrUpper( strSource )
```

Inputs

strSource

The string.

Returned value Returns the string with all characters in uppercase.

Examples

Tag name	Expression
Tag	StrUpper("Studio version 7.0") // Returned value = "STUDIO VERSION 7.0"
Tag	StrUpper("Technical Reference") // Returned value = "TECHNICAL REFERENCE"

19.9.27 ValueToChar

This function converts an integer array of Unicode characters to a string.

Function	Group	Execution	Windows	Embedded	Thin Client
ValueToChar	Character string	synchronous	supported	supported	supported

Syntax

```
ValueToChar("tagArray",numQtdChars)
```

Inputs

tagArray

The name of the integer array that is to receive the converted values. If no array index is specified, the default is 0.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

numQtdChars

The number of values to be converted (minimum of 1), starting with the specified array index.

Returned value Returns a string with the converted values.

Examples If Array[0] = 65, Array[1] = 66 and Array[2] = 67, the returned value is "ABC":

ValueToChar("Array", 3)

If Array[10] = 65, Array[11] = 66 and Array[12] = 67, the returned value is "ABC":

ValueToChar("Array[10]", 3)

19.9.28 ValueWToChar

This function converts an integer array of Unicode characters to a string where each value in the array is a double-byte word.

Function	Group	Execution	Windows	Embedded	Thin Client
ValueWToChar	Character string	synchronous	supported	supported	supported

Syntax

```
ValueWToChar("tagArray",numQtdChars)
```

Inputs

tagArray

The name of the integer array containing the double-byte values to be converted. If no array index is specified, the default value is 0.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

numQtdChars

The number of values to be converted (minimum of 1), starting with the specified array index.

Returned value

Returns a string with the converted values.

☞ Each value in the array has to be a double-byte word and each word is split into two bytes for conversion. As such, this function only supports Unicode characters 0 through 255. For characters greater than 255 or if double-byte words are not needed, use the ValueToChar page 646 function.

Examples

If Array[0] = 29779, Array[1] = 25717 and Array[2] = 28521, the returned value is "Studio":

ValueWToChar("Array", 3)

If Array[10] = 29779, Array[11] = 25717 and Array[12] = 28521, the returned value is "Studio":

ValueWToChar("Array[10]", 3)

19.10 Date and Time Functions

19.10.1 Date & Time functions – General Information

These functions are used to interact with the system clock or manipulate time stamps.

Function	Description
ClockGetDate, page 648	Calculates the date that corresponds to a specified number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970.
ClockGetDayOfWeek, page 649	Calculates the day of the week that corresponds to a specified number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970.
ClockGetTime, page 651	Calculates the time that corresponds to a specified number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970.
DateTime2Clock, page 652	Calculates how many seconds have elapsed since 12:00 AM GMT on January 1, 1970 (taking into account the current time zone specified on the local computer.)
DateTime2UTC, page 653	Converts a date and time from the specified time zone to Coordinated Universal Time (UTC).
GetClock, page 655	Calculates how many seconds have elapsed since 12:00 AM GMT on January 1, 1970, at the moment the function was run (taking into account the current time zone, as specified on the local computer).
GetTimeZone, page 655	Gets a specified time zone name or index.
GetTimeZoneCount, page 657	Gets the number of available time zones on the local computer.
GetUTC, page 657	Gets the current Coordinated Universal Time (UTC) on the local computer.
Hour2Clock, page 657	Converts time in the HH:MM:SS format into seconds.
SetSystemDate, page 658	Sets the date in the operating system's clock.
SetSystemTime, page 658	Sets the time in the operating system's clock.
SetTimeZone, page 659	Sets the time zone for the Viewer module / thin client, separate from the local computer's system settings.
UTC2DateTime, page 660	Converts a date and time from Coordinated Universal Time (UTC) to the specified time zone.

19.10.2 ClockGetDate

Calculates the date, based on how many seconds have elapsed since 12:00 AM GMT on January 1, 1970 (taking into account the current time zone of the computer).

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetDate	Date & Time	synchronous	supported	supported	supported

Syntax

```
ClockGetDate( numSeconds, optTimeZone )
```

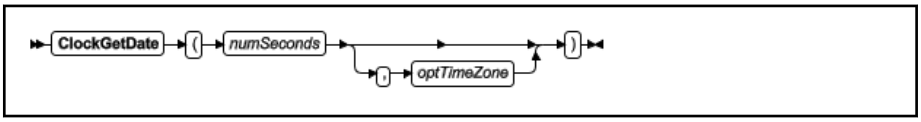


Fig. 19-4: Syntax

Inputs

numSeconds

The number of seconds elapsed since 12:00 AM GMT on January 1, 1970.

This number is typically provided by the function GetClock, page 655, but it can be any number.

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

If this function is executed successfully, it returns a string that contains the date that corresponds to the specified number of seconds. If a time zone is specified, the date is adjusted to reflect the change from the current time zone to the specified time zone. The date is formatted as `MM/DD/YYYY`.

If this function is not executed successfully - for example, if the specified time zone is invalid - it returns a value of `0`.

Notes

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Get the date when the system clock started counting (i.e., 01/01/1970):

```
ClockGetDate(0,"Greenwich Mean Time")
```

Get a returned value of 04/15/2002:

```
ClockGetDate(1018886359)
```

Get the current number of seconds elapsed since 12:00 AM GMT on January 1, 1970, then format that as a date, then adjust it from the current time zone to "Central Standard Time":

```
ClockGetDate(GetClock(),"Central Standard Time")
```

19.10.3 ClockGetDayOfWeek

Calculates the day of the week that corresponds to a specified number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetDayOf-Week	Date & Time	synchronous	supported	supported	supported

Syntax

```
ClockGetDayOfWeek( numSeconds, optTimeZone )
```

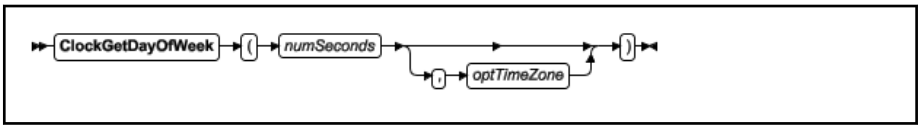


Fig. 19-5: Syntax

Inputs

numSeconds

The number of seconds elapsed since 12:00 AM GMT on January 1, 1970.

This number is typically provided by the function GetClock, page 655, but it can be any number.

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

Returns the day of the week (calculated in integer format) as follows:

0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

If a time zone is specified, the day is adjusted to reflect the change from the current time zone to the specified time zone.

Notes If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples *Get the day of the week when the system clock started counting (i.e., Wednesday):*

```
ClockGetTime(0)
```

Get a returned value of 1 (Monday):

```
ClockGetDayOfWeek(1018886359)
```

Get the current number of seconds elapsed since 12:00 AM GMT on January 1, 1970, and then calculate the day of the week:

```
ClockGetDayOfWeek(GetClock())
```

19.10.4 ClockGetTime

This function calculates the time that corresponds to a specified number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetTime	Date & Time	synchronous	supported	supported	supported

Syntax

```
ClockGetTime( numSeconds, optTimeZone )
```

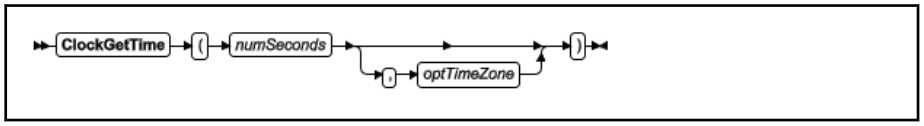


Fig. 19-6: Syntax

Inputs

numSeconds

The number of seconds elapsed since 12:00 AM GMT on January 1, 1970.

This number is typically provided by the function GetClock, page 655, but it can be any number.

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).

Value	Description
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

If this function is executed successfully, it returns a string that contains the time that corresponds to the specified number of seconds. If a time zone is specified, the time is adjusted to reflect the change from the current time zone to the specified time zone. The time is formatted as `HH:MM:SS`.

If this function is not executed successfully - for example, if the specified time zone is invalid - it returns a value of `0`.

Notes

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Get the time when the system clock started counting (i.e., 00:00:00):

```
ClockGetTime(0,"Greenwich Mean Time")
```

Get a returned value of 10:59:19:

```
ClockGetTime(1018886359)
```

Get the current number of seconds elapsed since 12:00 AM GMT on January 1, 1970, then format that as a time, then adjust it from the current time zone to "Central Standard Time":

```
ClockGetTime(GetClock(),"Central Standard Time")
```

19.10.5 DateTime2Clock

Calculates how many seconds have elapsed since 12:00 AM GMT on January 1, 1970 (taking into account the current time zone specified on the local computer.)

Function	Group	Execution	Windows	Embedded	Thin Client
DateTime2Clock	Date & Time	synchronous	supported	supported	supported

Syntax

```
DateTime2Clock( strDate, strTime, optTimeZone )
```

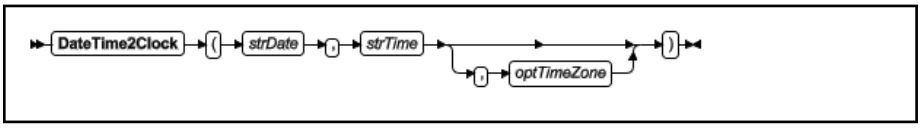


Fig. 19-7: Syntax

Inputs

strDate

The date to be used in the calculation, formatted as MM/DD/YYYY.

strTime

The time to be used in the calculation, formatted as HH:MM:SS.

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value Returns the number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970. If a time zone is specified, then the number of seconds is adjusted to reflect the change from the current time zone to the specified time zone.

Notes If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples *Get the number of seconds elapsed when the system clock started counting:*

```
DateTime2Clock("01/01/1970","00:00:00","Greenwich Mean Time")
```

Get the number of seconds elapsed at 10:59:19 AM on April 15, 2002, in the current time zone:

```
DateTime2Clock("04/15/2002","10:59:19")
```

19.10.6 DateTime2UTC

This function converts a date and time from the specified time zone to Coordinated Universal Time (UTC).

Function	Group	Execution	Windows	Embedded	Thin Client
DateTime2Clock	Date & Time	synchronous	supported	supported	supported

Syntax

```
DateTime2UTC( strDateTime, optTimeZone )
```

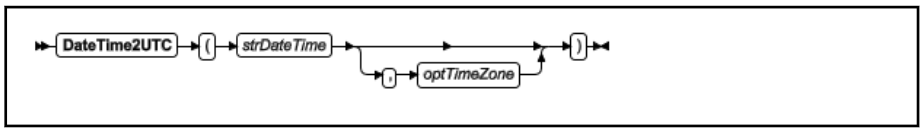


Fig. 19-8: Syntax

Inputs

strDateTime

The date and time to be converted, in the format "MM/DD/YYYY HH:MM:SS".

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

If this function is executed successfully, it returns a string that contains the specified date and time converted from the specified time zone to UTC. Otherwise, it returns one of the following possible values:

Value	Description
-5	Invalid time zone name or index.
-4	Invalid or improperly formatted date/time.
-3	The specified value is not a string.
-2	Invalid number of parameters.

Notes

The list of available time zones varies by operating system version and configuration. If necessary, you can use the functions GetTimeZoneCount, page 657, and GetTimeZone, page 655 to generate a list that is specific to your computer.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Convert the specified date and time from the current time zone to UTC:

```
DateTime2UTC("07/15/2013 19:54:46")
```

Convert the specified date and time from Central Standard Time to UTC:

```
DateTime2UTC("07/15/2013 19:54:46","Central Standard Time")
```

19.10.7 GetClock

Calculates how many seconds have elapsed since 12:00 AM GMT on January 1, 1970, at the moment the function was run (taking into account the current time zone, as specified on the local computer).

Function	Group	Execution	Windows	Embedded	Thin Client
GetClock	Date & Time	synchronous	supported	supported	supported

Syntax

```
GetClock( optTimeZone )
```



Fig. 19-9: Syntax

Inputs

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

Returns the number of seconds that have elapsed since 12:00 AM GMT on January 1, 1970, at the moment the function was run. If a time zone is specified, the number of seconds is adjusted to reflect the change from the current time zone to the specified time zone.

Notes

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

19.10.8 GetTimeZone

This function gets a specified time zone name or index.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetTime	Date & Time	synchronous	supported	supported	supported

Syntax

```
GetTimeZone( optTimeZone )
```

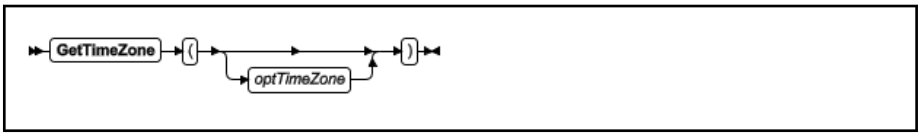


Fig. 19-10: Syntax

Inputs

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

If "optTimeZone" is a time zone name, this function returns the corresponding index.

If "optTimeZone" is a time zone index, this function returns the corresponding name.

Otherwise, this function returns one of the following possible values:

Value	Description
2	Invalid number of parameters
3	Invalid time zone name or index
4	Windows Embedded Compact only supports the current time zone

Notes

The list of available time zones varies by operating system version and configuration. If necessary, you can use this function in combination with the function GetTimeZoneCount, page 657, to generate a list that is specific to your computer.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Get the current time zone:

```
GetTimeZone()
```

Get the time zone name that corresponds to time zone index 24:

```
GetTimeZone(-24)
```

Get the time zone index that corresponds to "Central Standard Time":

```
GetTimeZone("Central Standard Time")
```

19.10.9 GetTimeZoneCount

Gets the number of available time zones on the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetTime	Date & Time	synchronous	supported	supported	supported

Syntax

```
GetTimeZoneCount()
```

This function has no parameters.

Returned value

This function returns the number of available time zones on the local computer, depending on where the function is called from (i.e., the project runtime or the Viewer module / thin client).

On Windows Embedded Compact, this function always returns `1` because the operating system only supports the current time zone.

Notes

The list of available time zones varies by operating system version and configuration. If necessary, you can use this function in combination with the function GetTimeZone, page 655 to generate a list that is specific to your computer.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

19.10.10 GetUTC

Gets the current Coordinated Universal Time (UTC) on the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetTime	Date & Time	synchronous	supported	supported	supported

Syntax

```
GetUTC()
```

This function has no parameters.

Returned value

This function returns a string that contains the current Coordinated Universal Time (UTC) on the local computer, depending on where the function is called from (i.e., the project runtime or the Viewer module / thin client). The time is formatted as `MM/DD/YYYY HH:MM:SS`.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

19.10.11 Hour2Clock

Converts time in the `HH:MM:SS` format into seconds.

Function	Group	Execution	Windows	Embedded	Thin Client
Hour2Clock	Date & Time	synchronous	supported	supported	supported

Syntax

```
Hour2Clock( strTime )
```

Inputs

strTime
The number of hours, minutes and seconds in `HH:MM:SS` format.

Returned value Returns the number of seconds corresponding to the total number of hours, minutes and seconds specified.

Examples

Tag name	Expression
Tag	Hour2Clock("01:00:00") // Returned value = 3600
Tag	Hour2Clock("10:01:01") // Returned value = 36061

19.10.12 SetSystemDate

Sets the date in the clock of the operating system.

Function	Group	Execution	Windows	Embedded	Thin Client
SetSystemDate	Date & Time	synchronous	supported	supported	supported

Syntax

```
SetsystemDate( strDate )
```

Inputs

strDate

The date in `MM/DD/YYYY` format to set the clock.

Returned value No returned values.

Notes For this function to execute successfully and change the system settings, you must run Rexroth WinStudio with Administrator privileges. To run as an administrator once, right-click the program icon and then click **Run as administrator** on the shortcut menu. To always run as an administrator, open the program's Properties dialog box, click **Compatibility**, and then select **Run this program as an administrator**.

Examples *Set the system clock to 15 Apr 2002:*

```
SetSystemDate("04/15/2002")
```

19.10.13 SetSystemTime

Sets the time in the clock of the operating system.

Function	Group	Execution	Windows	Embedded	Thin Client
SetSystemTime	Date & Time	synchronous	supported	supported	supported

Syntax

```
SetSystemTime( strTime )
```

Inputs

strTime

The time in "HH:MM:SS" format to set the clock.

Returned value No returned values.

Notes For this function to execute successfully and change the system settings, you must run Rexroth WinStudio with Administrator privileges. To run as an administrator once, right-click the program icon and then click **Run as administrator** on the shortcut menu. To always run as an administrator, open the program's Properties dialog box, click **Compatibility**, and then select **Run this program as an administrator**.

Examples *Set the system clock to 3:45:18 PM:*

```
SetSystemTime("15:45:18")
```

19.10.14 SetTimeZone

This function sets the time zone for the Viewer module / thin client, separate from the local computer's system settings.

Function	Group	Execution	Windows	Embedded	Thin Client
ClockGetTime	Date & Time	synchronous	supported	supported (see "Notes" below)	supported

Syntax

```
SetTimeZone( optTimeZone )
```

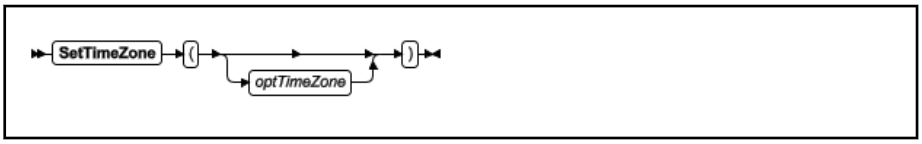


Fig. 19-11: Syntax

Inputs

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is `2`.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657.
0	The current time zone in the Viewer module / thin client.
1	Coordinated Universal Time (UTC)
2	The current time zone in the local computer's system settings
"timezone"	The full name of a time zone.

Returned value

This function returns one of the following possible values:

Value	Description
-3	Invalid time zone name or index
-2	Invalid number of parameters
-1	Invalid function call; this function can only be called from the Viewer module / thin client. The project runtime will always use the local computer's system settings.
0	Function executed successfully

Notes

The list of available time zones varies by operating system version and configuration. If necessary, you can use the functions GetTimeZoneCount, page 657, and GetTimeZone, page 655 to generate a list that is specific to your computer.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Reset the time zone to the local computer's system settings:

```
SetTimeZone()
```

Set the time zone to UTC:

```
SetTimeZone(1)
```

Set the time zone to index 24:

```
SetTimeZone(-24)
```

19.10.15 UTC2DateTime

Converts a date and time from Coordinated Universal Time (UTC) to the specified time zone.

Function	Group	Execution	Windows	Embedded	Thin Client
DateTime2Clock	Date & Time	synchronous	supported	supported	supported

Syntax

```
UTC2DateTime( strUTC, optTimeZone )
```

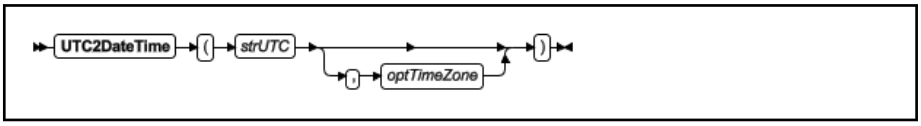


Fig. 19-12: Syntax

Inputs

strUTC

The date and time to be converted, in the format "MM/DD/YYYY HH:MM:SS".

optTimeZone

A time zone name (string) or index (integer).

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

Value	Description
-i	A time zone index (`i`), up to the maximum index returned by the function GetTimeZoneCount, page 657. On Windows Embedded Compact, the only valid index is `-1` (i.e., the current time zone).
0	The current time zone in either the project runtime or the Viewer module / thin client, depending on where the function is called from. This may be different from the current time zone in the local computer's system settings, if the function SetTimeZone, page 659 was executed previously.
1	Coordinated Universal Time (UTC).
2	The current time zone in the local computer's system settings. On Windows Embedded Compact, this value is invalid.
"timezone"	The full name of a time zone (e.g., "Central Standard Time"). On Windows Embedded Compact, this value is invalid.

Returned value

If this function is executed successfully, it returns a string that contains the specified date and time converted from the specified time zone to UTC. Otherwise, it returns one of the following possible values:

Value	Description
-5	Invalid time zone name or index.
-4	Invalid or improperly formatted date/time.
-3	The specified value is not a string.
-2	Invalid number of parameters.

Notes

The list of available time zones varies by operating system version and configuration. If necessary, you can use the functions GetTimeZoneCount, page 657, and GetTimeZone, page 655 to generate a list that is specific to your computer.

If this function is called on Mobile Access, it is executed using the system time on the project runtime server.

Examples

Convert the specified date/time from UTC to the current time zone:

```
UTC2DateTime("07/15/2013 19:54:46")
```

Convert the specified date and time from UTC to Central Standard Time:

```
UTC2DateTime("07/15/2013 19:54:46","Central Standard Time")
```

Convert the current date and time from UTC to Central Standard Time:

```
UTC2DateTime(GetUTC(),"Central Standard Time")
```

19.11 Trigonometric Functions

19.11.1 Trigonometric Functions – General Information

These functions are used to calculate trigonometric values (e.g., sine, cosine, tangent).

Function	Descripon
ACos, page 661	Calculates the Arc Cosine of a value.
ASin, page 662	Calculates the Arc Sine of a value.
ATan, page 662	Calculates the Arc Tangent of a value.
Cos, page 662	Calculates the Cosine of a value.
Cot, page 663	Calculates the cotangent of an angle.
Pi, page 663	Calculates the value of pi.
Sin, page 664	Calculates the sine of an angle.
Tan, page 664	Calculates the tangent of an angle.

19.11.2 ACos

Calculates the Arc Cosine of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
ACos	Trigonometric	synchronous	supported	supported	supported

Syntax

```
ACos( numValue )
```

Inputs

numValue
Numeric tag for the value of which the Arc Cosine is calculated.

Returned value Returns the Arc Cosine of "numValue" in radians.

Examples

Tag name	Expression
Tag	ACos(1) // Returned value = 0.000000
Tag	ACos(0) // Returned value = 1.570796

19.11.3 ASin

Calculates the Arc Sine of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
ASin	Trigonometric	synchronous	supported	supported	supported

Syntax

```
ASin( numValue )
```

Inputs

numValue
Numeric tag for the value of which the Arc Sine is calculated.

Returned value Returns the Arc Sine of "numValue" in radians.

Examples

Tag name	Expression
Tag	ASin(1) // Returned value = 1.570796
Tag	ASin(0) // Returned value = 0.000000

19.11.4 ATan

Calculates the Arc Tangent of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
ATan	Trigonometric	synchronous	supported	supported	supported

Syntax

```
ATan( numValue )
```

Inputs

numValue
Numeric tag for the value of which the Arc Tangent is calculated.

Returned value Returns the Arc Tangent of "numValue" in radians.

Examples

Tag name	Expression
Tag	ATan(1) // Returned value = 0.785398
Tag	ATan(0) // Returned value = 1.570796

19.11.5 Cos

Calculates the Cosine of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
Cos	Trigonometric	synchronous	supported	supported	supported

Syntax

```
Cos( numAngle )
```

Inputs

numAngle
The angle (in radians) the Cosine of which is calculated.

Returned value Returns the Cosine of "numAngle".

Examples

Tag name	Expression
Tag	Cos(1.570796) // Returned value = 0.000000
Tag	Cos(0) // Returned value = 1.000000

19.11.6 Cot

Calculates the Cotangent of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
Cot	Trigonometric	synchronous	supported	supported	supported

Syntax

```
Cot( numAngle )
```

Inputs

numAngle
The angle (in radians) the Cotangent of which is calculated.



2π radians is the same as 360 degrees.

Returned value Returns the Cotangent of "numAngle".

The function cannot return the actual Cotangent of π (i.e., Tan(Pi()) or equivalent), because that is infinite.

Instead, the function returns the largest number possible, given the limited precision of the value returned by the function Pi(), page 663.

Examples

Calculate the cotangent of 1 radian:

```
Cot(1)
```

Calculate the cotangent of π/2 radians:

```
Cot(Pi()/2)
```

Calculate the cotangent of the value of the project tag MyAngle:

```
Cot(MyAngle)
```

19.11.7 Pi

Calculates the value of "pi".

Function	Group	Execution	Windows	Embedded	Thin Client
Pi	Trigonometric	synchronous	supported	supported	supported

Syntax

```
Pi()
```

This function does not have any parameters. However, the parentheses are still to be included as the function is otherwise recognized as tag instead as a function.

Returned value Returns the value of "Pi".

Examples

Tag name	Expression
Tag	Pi() // Returned value = 3.141593

19.11.8 Sin

Calculates the Sine of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
Sin	Trigonometric	synchronous	supported	supported	supported

Syntax

```
Sin( numAngle )
```

Inputs

numAngle

The angle (in radians) the Sine of which is calculated.

Returned value Returns the Sine of "numAngle".

Examples

Tag name	Expression
Tag	Sin(0) // Returned value = 0.000000
Tag	Sin(1.570796) // Returned value = 1.000000

19.11.9 Tan

Calculates the Tangent of a value.

Function	Group	Execution	Windows	Embedded	Thin Client
Tan	Trigonometric	synchronous	supported	supported	supported

Syntax

```
Tan( numAngle )
```

Inputs

numAngle

The angle (in radians) the Tangent of which is calculated.



2π radians is the same as 360 degrees.

Returned value Returns the Tangent of "numAngle".

The function cannot return the actual tangent of π/2 (i.e., Tan(Pi()/2) or equivalent), because that is infinite.

Instead, the function returns the largest number possible, given the limited precision of the value returned by the function Pi(), page 663.

Examples

Calculate the tangent of 1 radian:

```
Tan(1)
```

Calculate the tangent of π radians:

```
Tan(Pi())
```

Calculate the tangent of the value of the project tag MyAngle:

```
Tan(MyAngle)
```

19.12 Screen Functions

19.12.1 Screen Functions – General Information

These functions are used to open and close WinStudio screens.

Function	Description
Close, page 665	Is a built-in scripting function that closes an open project screen.
Open, page 666	Opens a project screen on the client with some specified settings.
OpenPrevious, page 670	Is a built-in scripting function that re-opens the last screen to be closed.
ShowInplaceInput, page 671	Shows a simple text input dialog at a specified location in the project client/viewer.
ShowMessageBox, page 673	Is a built-in function that shows a message box with one or more buttons to capture the user's response.

19.12.2 Close

"Close" is an internal scripting function that closes an open WinStudio screen.

Function	Group	Execution	Windows	Embedded	Thin Client
Close	Screens	asynchronous	supported	supported	supported

Syntax

```
Close(strScreen{ | ,optNumID })
```

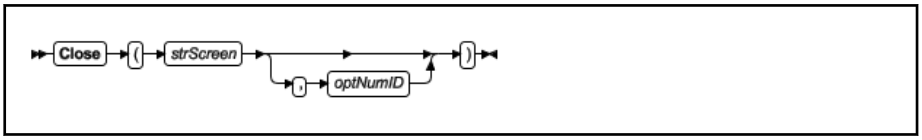


Fig. 19-13: Syntax

Inputs

strScreen

The name of the screen (not including the .scr extension) to be closed.

Some web servers are case-sensitive. If the project is to be deployed as a web application, only lowercase letters are to be used for the screen name.

optNumID
The specific ID or instance number of the screen to be closed, if there is more than one open screen with the same name. (The ID is assigned when the screen is opened with the Open, page 666 function.) This parameter is optional. If no value is specified, the default ID is 0.

Returned value This function does not return any value.

Notes If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

The following limitations also apply to how the function can be used in project screens in Mobile Access:

- The function cannot be called from the sub-routine `Screen_OnClose` in the Screen Script, page 932; and
- You should avoid creating a script that both shows a dialog box (e.g., a message box that is shown using the VBScript function MsgBox, page 956 or the WinStudio built-in function ShowMessageBox, page 673) and opens or closes a screen. The Mobile Access web interface can display only one screen at a time, and opening a new screen will automatically close the previous one. Depending on the order of execution, this can lead to unexpected behavior such as a dialog box being orphaned or not being shown at all.

Finally, in some cases, you do not need to call this function to close a screen because the screen will be closed automatically when another screen replaces it. For more information, see Screen attributes dialog, page 170.

Examples *Closing the WinStudio screen named "main":*

```
Close( "main" )
```

Closing the WinStudio screen named "alarms":

```
Close( "alarms" )
```

Close the WinStudio screen named "main" with ID 10:

```
Close( "main", 10 )
```

19.12.3 Open

"Open" is an internal scripting function that opens a WinStudio screen.

Function	Group	Execution	Windows	Embedded	Thin Client
Open	Screens	asynchronous	supported	supported	supported

Syntax

```
Open(strScreenAndProperties,optNumX1,optNumY1,optNumX2,optNumY2,optNumResizeFlag,optNumID,optStrMnemonicList)
```

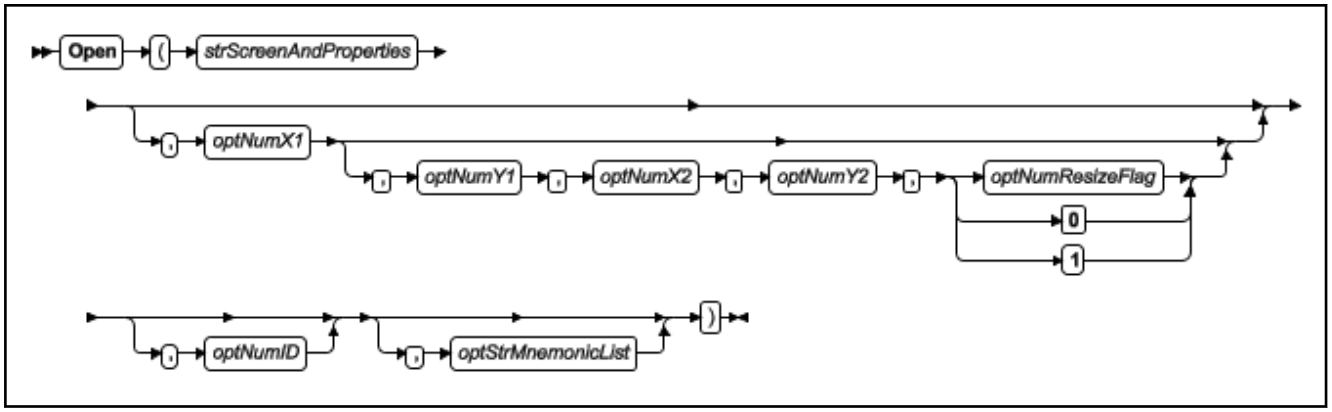


Fig. 19-14: Syntax

Inputs

strScreenAndProperties

The name of the screen to be opened. (The screen file extension *.SCR is assumed, so you do not need to include it.)

If you specify only the screen name, then it will be opened with its default properties (as specified in Screen attributes dialog, page 170). You can, however, customize the properties as follows:

Property	Syntax
Title Bar	`EnableTitleBar: Enable | Disable; TitleBar: title`
System Menu	`SystemMenu: Enable | Disable`
Maximize Box	`Maximize: Enable | Disable`
Minimize Box	`Minimize: Enable | Disable`
Style	`Style: Overlapped | Popup | Replace(Partial) | Dialog | Replace(Complete)`
Border	`Border: None | Thin | Resizing`

optNumX1, optNumY1, optNumX2, optNumY2

The coordinates, in pixels, for the top-left (X1,Y1) and bottom-right (X2,Y2) corners of the screen.

These are optional parameters. If no values are specified, then the default screen size and location are used.

Please note the following special circumstances:

- You can open the screen at the mouse's current position by using `Open("screen",1)`, or `Open("screen",1,-1,-1,-1,…)` if the parameters at the end are needed.
- If optNumX1 equals optNumX2 and optNumY1 equals optNumY2, then the default screen size is used but the screen is centered at `(X1,Y1)`.
- If optNumX2 is less than optNumX1 and/or optNumY2 is less than optNumY1, or if all four parameters are set to `-1`, then the parameters are ignored and the default screen size and location are used.

optNumResizeFlag

Specifies whether the size of the screen objects is adjusted when opening the screen:

Value	Description
0	The size of screen objects is not adjusted.
1	Screen objects are automatically adjusted to fit the new dimensions of the screen, as specified by the coordinates described above. The adjustment is made when opening the screen. If the screen size is changed after opening, the objects are not readjusted.

This parameter is required if all four coordinates are specified.

optNumID

An ID or instance number is assigned to the screen as multiple instances of the same screen file can be opened. (This ID is required if a screen is closed using the Close, page 665 function.)

This parameter is optional. If the value is missing, the default value of `0` is used.

optStrMnemonicList

A string that describes how the custom properties of generic objects or Saving own project symbols, page 223 in the screen are completed if the screen is opened. This string has the following syntax:

```
# Label: Value
```

"Label" is the name of the property. "Value" is the tag, expression or direct value assigned to the property. More than one mnemonic can be declared as long as the mnemonics are separated by spaces.

You can also specify a external text file that contains the mnemonics separated by either spaces or line returns. The file must have the extension `*.MNE`, and it must be saved in the Web sub-folder of your project folder. Please note that the file extension is assumed, so you do not need to include it in parameter, and if you want to save the file in a deeper sub-folder, then you need to specify a file path relative to the Web sub-folder.



Mnemonics are not supported when opening project screens in Mobile Access.

Returned value

This function returns one of the following values:

Value	Description
0	Failure
1	Success

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

The following limitations also apply to how the function can be used in project screens in Mobile Access:

- The function cannot be called from the sub-routine `Screen_OnClose` in the Screen Script, page 932; and
- You should avoid creating a script that both shows a dialog box (e.g., a message box that is shown using the VBScript function MsgBox, page 956 or the WinStudio built-in function ShowMessageBox, page 673) and opens or closes a screen. The Mobile Access web interface can display only one screen at a time, and opening a new screen will automatically close the previous one. Depending on the order of execution, this can lead to unexpected behavior such as a dialog box being orphaned or not being shown at all.

Examples

Open the screen using the default screen size, location, and ID:

```
Open("main")
```

Open the screen at the mouse's current position:

```
Open("main",1)
```

Open the screen at the mouse's current position and assign it an ID of 10:

```
Open("main",1,-1,-1,-1,0,10)
```

Open the screen using the default screen size but centered at the coordinates (500,250), do not resize the screen objects, and assign the screen an ID of 10:

```
Open("main",500,250,500,250,0,10)
```

Open the screen using the default screen size and location, use the default ID, and replace the custom properties Mne1 and Mne2 with Tag1 and Tag2, respectively:

```
Open("main",-1,-1,-1,-1,0,0,"#Mne1:Tag1 #Mne2:Tag2")
```

Open the screen using the default screen size and location, use the default ID, and replace the custom properties with values defined in the mnemonics file located at: projectname\Web\mnemonic.mne

```
Open("main",-1,-1,-1,-1,0,0,"mnemonic")
```

Open the screen using the default screen size but centered at the coordinates (500,250), do not resize the screen objects, assign the screen an ID of 10, and replace the custom properties with values defined in the mnemonics file located at: projectname\Web\mnemonic.mne

```
Open("main",500,250,500,250,0,10,"mnemonic")
```

Open the screen using the default screen size and location, assign it an ID of 2, and replace the custom properties with values defined in the mnemonics file located at: projectname\Web\Mnemonics\mnemonic.mne

```
Open("main",-1,-1,-1,-1,0,2,"Mnemonics\mnemonic")
```

Open the screen using the default screen size and location, do not resize the screen objects, use the default ID, and customize the screen properties:

```
Open("main; EnableTitleBar: Enable; TitleBar: Main Screen; SystemMenu: Enable; Maximize: Disable;
                            Minimize: Disable; Style: Popup; Border: Thin",-1,-1,-1,-1,0,0,"")
```

19.12.4 OpenPrevious

"OpenPrevious" is an integrated scripting function that re-opens the WinStudio screen that was closed last.

Function	Group	Execution	Windows	Embedded	Thin Client
OpenPrevious	Screens	asynchronous	supported	supported	supported

Syntax

```
OpenPrevious({ | optNumX1,optNumY1,optNumX2,optNumY2 })
```

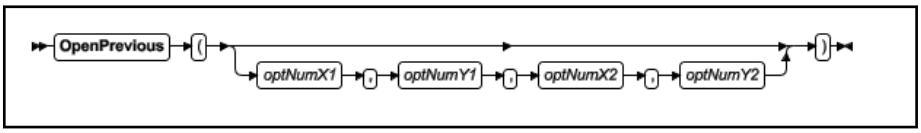


Fig. 19-15: Syntax

Inputs

optNumX1, optNumY1, optNumX2, optNumY2

The coordinates in pixels for the upper-left (X1,Y1) and lower-right (X2,Y2) corners of the screen.

These are optional parameters. If no values are specified, the default screen size and location are used. For further information, refer to Screen attributes dialog, page 170.

Returned value

This function returns one of the following values:

Value	Description
0	Failure
1	Success

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

The following limitations also apply to how the function can be used in project screens in Mobile Access:

- The function cannot be called from the sub-routine `Screen_OnClose` in the Screen Script, page 932; and
- You should avoid creating a script that both shows a dialog box (e.g., a message box that is shown using the VBScript function MsgBox, page 956 or the WinStudio built-in function ShowMessageBox, page 673) and opens or closes a screen. The Mobile Access web interface can display only one screen at a time, and opening a new screen will automatically close the previous one. Depending on the order of execution,

this can lead to unexpected behavior such as a dialog box being orphaned or not being shown at all.

Examples

Opening the previous screen using its default size and location:

```
OpenPrevious()
```

Opening the previous screen in the top-left corner of the display and sized to 800x600:

```
OpenPrevious( 0, 0, 800, 600 )
```

19.12.5 ShowInplaceInput

This function shows a simple text input dialog at a specified location in the project client/viewer.

Function	Group	Execution	Windows	Embedded	Thin Client
ShowInplaceInput	Screens	asynchronous	supported	supported	supported

Syntax

```
ShowInplaceInput("tagOutput",numStartXPos,numStartYPos,optNumMin,optNumMax,optNumEnablePasswordMode,
                                                                                    optNumShowOSVK)
```

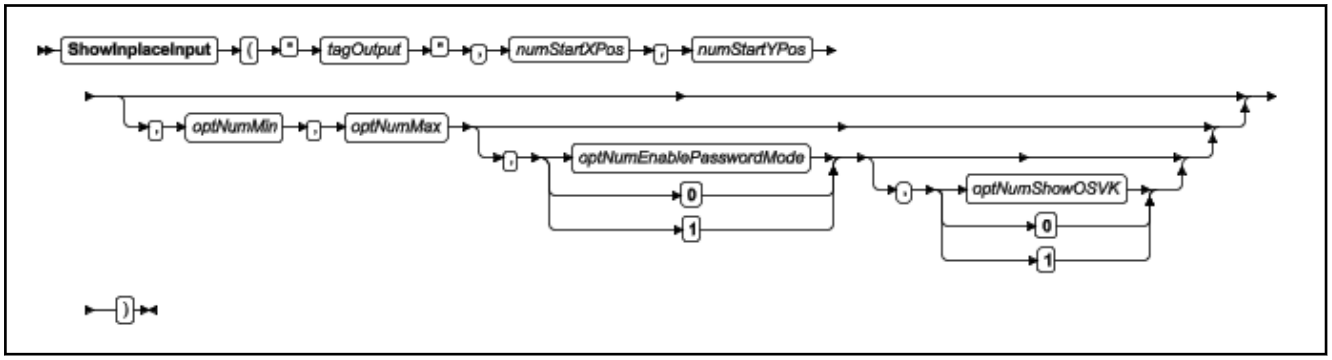


Fig. 19-16: Syntax

Inputs

tagOutput

The name of a tag that receives the input.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

numStartXPos

The starting X position in the top-left corner of the input dialog – i.e., the number of pixels between this corner and the left side of the display.

numStartYPos

The starting Y position of the top-left corner of the input dialog – i.e., the number of pixels between this corner and the top of the display.

optNumMin

The minimum numeric value that is accepted by the input dialog.

This parameter is optional. If no value is specified, the dialog accepts any value.

optNumMax

The maximum numeric value that is accepted by the input dialog.

This parameter is optional. If no value is specified, the dialog accepts any value.

optNumEnablePasswordMode

Option to enable Password mode, which hides the operator input as if it is a password:

Value	Description
0	Show input as plain text
1	Hide input

This parameter is optional. If no value is specified, the default value is 0.

optNumShowOSVK

Option to show the default virtual keyboard Viewer tab, page 61:

Value	Description
0	Do not show the virtual keyboard
1	Show virtual keyboard

This parameter is optional. If no value is specified, the default value is 0.

Returned value

This function returns the following possible values:

Value	Description
0	Successful
-1	Invalid tag specified for "tagOutput"
-2	Invalid number of parameters
-3	Viewer is not running

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples

ShowInplaceInput - Examples

```
ShowInplaceInput( "OperatorInput", 50, 50 )
ShowInplaceInput( "OperatorInput", 50, 50, 1, 100 )
ShowInplaceInput( "OperatorInput", 50, 50, 1, 100, 0, 1 )
```

19.12.6 ShowMessageBox

This function shows a simple message box with either an <OK> button or <Yes> / <No> buttons.

Function	Group	Execution	Windows	Embedded	Thin Client
ShowMessage-Box	Screens	synchronous	supported	supported	supported

Syntax

```
ShowMessageBox(strMessage,optNumButtons,optStrTitle)
```

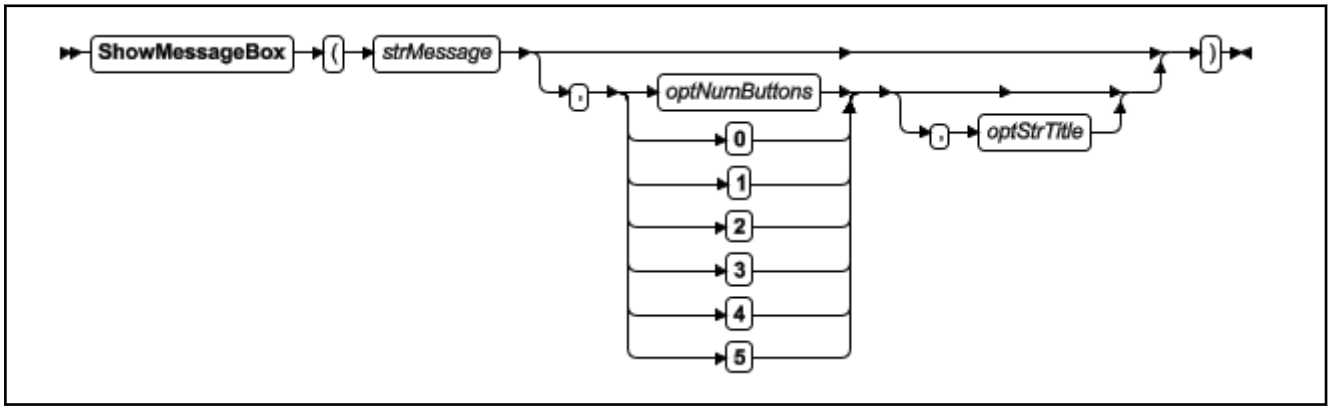


Fig. 19-17: Syntax

Inputs

strMessage

The message body to be displayed in the box.

optNumButtons

A numeric flag that specifies which kind of buttons to display in the message box:

Value	Description
0	<OK> button (default)
1	<OK> / <Cancel> buttons
2	<Abort> / <Retry> / <Ignore> buttons
3	<Yes> / <No> / <Cancel> buttons
4	<Yes> / <No> buttons
5	<Retry> / <Cancel> buttons

This parameter is optional. If no value is specified, the default value is 0.

optStrTitle

The title of the message.

This parameter is optional. If no value is specified, no title is displayed.

Returned value

This function returns the following possible values:

Value	Description
-1	Bad parameter
0	Message box not displayed because the Viewer is not open; see "Notes" below
1	Operator clicked on **OK**
2	User clicked **Cancel**
3	User clicked **Abort**
4	User clicked **Retry**
5	User clicked **Ignore**
6	Operator clicked **Yes**
7	Operator clicked **No**

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

The following limitations also apply to how the function can be used in project screens in Mobile Access:

- The function cannot be called from the sub-routine `Screen_OnClose` in the Screen Script, page 932; and
- You should avoid creating a script that both shows a dialog box (e.g., a message box that is shown using the VBScript function MsgBox, page 956 or the WinStudio built-in function ShowMessageBox, page 673) and opens or closes a screen. The Mobile Access web interface can display only one screen at a time, and opening a new screen will automatically close the previous one. Depending on the order of execution, this can lead to unexpected behavior such as a dialog box being orphaned or not being shown at all.



The message, button labels, and title (if any) can all be displayed in other languages during run time. For more information, see Project localization, page 487.

Examples

Display a plain message with an <OK> button:

```
ShowMessageBox("The action could not be completed.")
```

Display a question with <Yes> / <No> buttons:

```
ShowMessageBox("Continue with action?",4)
```

Display an alert with an <OK> button and a title:

```
ShowMessageBox("The action could not be completed.",0+48,"Alert")
```

19.13 Security Functions

19.13.1 Security Functions – General Information

These functions are used to manage users and groups in the security system of projects.

Function	Description
BlockUser, page 675	Is a built-in scripting function that blocks an existing user from logging onto a project. This allows you to disable a user account without deleting it.
CheckESign, page 676	Is a built-in function that prompts the run-time user to electronically sign an event by entering their username and password. You can call this function to secure scripts and expressions, just as you can select the E-Sign option in object properties to secure screen objects and animations.
CheckSecurityLevel, page 677	Checks whether the current user has access to the specified security level.
CreateUser, page 678	Creates a new user in your project's security system.
ExportSecuritySystem, page 680	Exports the security system configuration to an encrypted file.
GetLastESignUser, page 681	Is a built-in function that gets the last user who electronically signed an event during run time.
GetSecuritySystemStatus, page 681	Gets the status of the security system and its connection to the authentication server, when the security mode is either Distributed–Client or Domain (LDAP).
GetUserFullName, page 683	Gets the full name (if any) of a specified user in the project security system.
GetUserNames, page 683	Gets the name of all users; stores the names in the UsersArray tag
GetUserPwdAging, page 684	Gets the age of the password for a specified user - that is, the time remaining until the password expires, or if it has expired, the time since it expired.
GetUserState, page 685	Use to see the current status of a selected user.
ImportSecuritySystem, page 685	Imports a security system configuration from an external file.
RemoveUser, page 687	Removes a specified user from your project's security system.
SetPassword, page 688	Sets a new password for a specified user in your project's security system.
UnblockUser, page 689	Unblocks a blocked user in the security system.

19.13.2 BlockUser

"BlockUser" is a built-in scripting function that blocks an existing user from logging onto a project. This allows you to disable a user account without deleting it.

Function	Group	Execution	Windows	Embedded	Thin Client
BlockUser	Security	synchronous	supported	supported	supported

Syntax

```
BlockUser(strUserName)
```

Inputs

strUserName
The name of the user to block.

Returned value

This function returns the following possible values:

Value	Description
0	User blocked successfully
1	Invalid number of parameters
2	Incorrect parameter type
3	Specified user does not exist
4	User currently logged on does not have the rights to block (i.e., user does not have **Edit Security System**)
5	The operation on the distributed security system failed
6	User cannot be blocked
7	The current Security mode does not allow users to be blocked/ unblocked
8	Internal error

Notes

If you use this function to block a user, then the option User is blocked is selected in the User Account dialog. For more information, see Creating and configuring users, page 478.

Examples

Blocking the user named Bob:

```
BlockUser( "Bob" )
```

Block the user named in position 3 of the array badUsers:

```
BlockUser( badUsers[3] )
```

Block the user that is currently logged on:

```
BlockUser( UserName )
```

19.13.3 CheckESign

"CheckESign" is an internal scripting function that prompts the runtime user to electronically sign an event by entering their user name and password. It can be called to secure specific expressions and scripts, just as the **E-Sign** option secures screen objects and animations.

Function	Group	Execution	Windows	Embedded	Thin Client
CheckESign	Security	synchronous	supported	supported	supported

Syntax

```
CheckESign(optNumLevel)
```

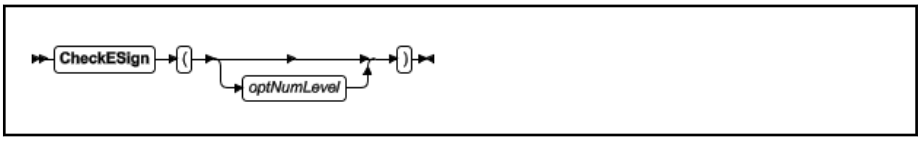


Fig. 19-18: Syntax

Inputs

optNumLevel

The security level (from 0 to 255, Runtime only) to which the user must have access in order to sign.

For more information, see Security access levels, page 462.

This parameter is optional; if no value is specified, the default value is 0.

Returned value

This function returns the following possible values:

Value	Description
0	Username and/or password not accepted, or the user does not have access to the specified level.
1	Username and password accepted. The event is saved in the event history with the user's signature.

Notes

Calling this function displays a dialog box on the client, where the user must enter their username and password.

Users and groups are managed in the project security system.

For more information, see Project security, page 461.

Examples

Prompt the user for their username and password — the user must have access to security level 20…

```
CheckESign(20)
```

19.13.4 CheckSecurityLevel

The function "CheckSecurityLevel" checks whether the current user has access to the specified security level.

Function	Group	Execution	Windows	Embedded	Thin Client
CheckSecurityLevel	Security	synchronous	supported	supported	supported

Syntax

```
CheckSecurityLevel(numLevel,optNumType)
```

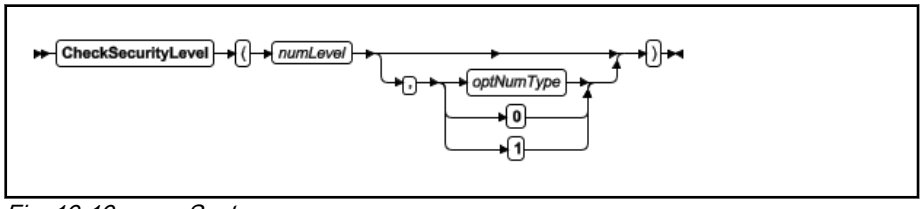


Fig. 19-19: Syntax

Inputs

numLevel

The security level to be checked.

optNumType

The type of security level to check: `0` is Runtime, `1` is Development.

This is an optional parameter; if no value is specified, the default value is `0`.

Returned value

This function returns the following possible values:

Value	Description
0	The current user does not have access to the specified security level.
1	The current user has access to the specified security level.

Notes

Each user can belong to multiple groups, and those groups typically have access to different security levels. Rather than check the groups to which a user belongs, and from that try to determine the security levels to which the user has access, you can use this function to check the security levels directly.

For more information about groups and security levels, see Creating and configuring groups, page 474.

Examples

The user "Bob" belongs to the groups "GroupA" and "GroupB".

GroupA has access to runtime levels 10 to 20, and GroupB has access to development levels 30 to 40. That means Bob has access to runtime security levels 10 to 20 and development security levels 30 to 40.

Given is this, if Bob is the current user (i.e., the user who is currently logged onto the client where the function is executed) and this function is called to check for access to runtime security level 17…

```
CheckSecurityLevel(17)
```

…it returns a value of 1, meaning that Bob has access.

Similary, if this function is called to check for access to development security level 33…

```
CheckSecurityLevel(33,1)
```

…it again returns a value of 1.

However, if this function is called to check for access to runtime security level 25…

```
CheckSecurityLevel(25,0)
```

…it returns a value of 0, meaning that Bob does not have access.

19.13.5 CreateUser

This function creates a new user in the project security system.

Function	Group	Execution	Windows	Embedded	Thin Client
CreateUser	Security	synchronous	supported	supported	supported

Syntax

```
CreateUser(optStrUserName,optStrGroup,optStrPassw,optStrUserFullName)
```

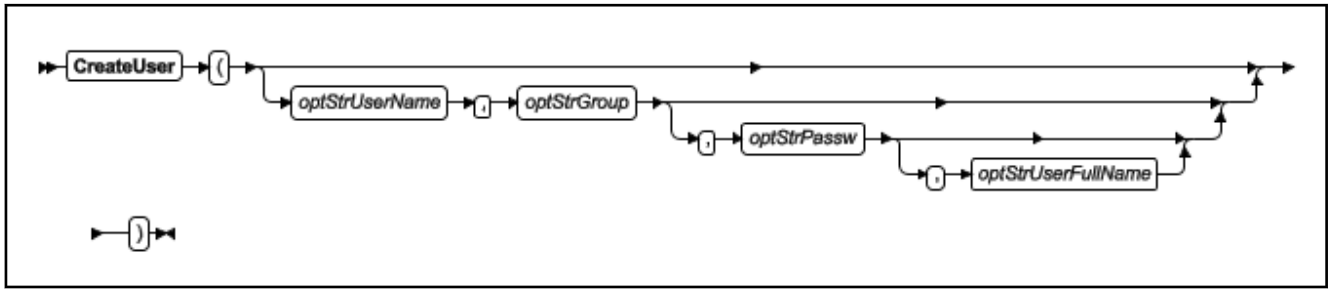


Fig. 19-20: Syntax

Inputs

optStrUserName

The name of the user to be created

optStrGroup

The name of the user to be created, and the group(s) to which the user will belong. You can specify multiple groups by separating them with a comma.

These are optional parameters; if no values are specified, a dialog box will be displayed on the client so that the current user can provide the information.

optStrPassw

The user's password.

This is an optional parameter; if no value is specified, the user will not have a password. (To specify one later, either call the function SetPassword, page 688, or edit the user's settings in the project security system.)

optStrUserFullName

The full name of the user.

This is an optional parameter; if no value is specified, the user will not have a full name. (To specify one later, edit the user's settings in the project security system.)

Returned value

This function returns the following possible values:

Value	Description
-1000	Could not display dialog box. The function should executed on the client.
-1	Internal error; contact technical support
0	New user created successfully
1	Invalid number of parameters
2	Incorrect parameter type
3	User name already exists
4	Group does not exist
5	Failed to save to configuration file
6	Invalid user
7	Full user name already exists
8	Reentrant function call not allowed
9	User clicked <Cancel> on the Create User dialog box
10	Invalid password, check the minimum password size specified for the group.
11	Invalid group. (Group may not have "Runtime group" option selected.)
12	Create User dialog box is already displayed, cannot display another dialog box. (For example, if the user clicked OK without providing all of the required information.)

Value	Description
13	Current user does not have sufficient privileges to create users (i.e., the option "Edit Security System" is not selected in the Security System settings).
14	The current Security Mode does not allow a user to be created.

Hinweise Users created with this function have the **Runtime user** option selected in the User Account dialog. For more information, see Creating and configuring users, page 478.

Examples *Display the Create User dialog box on the client, to get the information from the current user:*

```
CreateUser()
```

Create a user named "Bob" in the group "Admin", with the password "Chocolate":

```
CreateUser("Bob","Admin","Chocolate")
```

Create a user named "Albert" (full name "Albert Jones") in the group "Engineering", with the password "EMC2":

```
CreateUser("Albert","Engineering","EMC2","Albert Jones")
```

19.13.6 ExportSecuritySystem

This function exports the security system configuration to an encoded file.

Function	Group	Execution	Windows	Embedded	Thin Client
ExportSecurity-System	Security	synchronous	supported	supported	supported

Syntax

```
ExportSecuritySystem(strFileName,strPassword)
```

Eingaben

strFileName

The complete file path and name to save the configuration file.

strPassword

The main password for the security system. This same password is used to protect the exported file.

Returned value This function returns the following possible values:

Value	Description
-2	Incorrect parameter type
-1	Invalid number of parameters
0	Could not write security data
1	File exported successfully

Notes This function can be called only from scripts executed on the project server. None of the connected clients - not even the client running on the same computer as the project server, because it runs as a separate process on that

computer - have the necessary access to the security system. Therefore, generally speaking:

- It can be called from the Startup Script (which is executed when the project itself is run), Script Groups (which are continuously executed by the Background Task), and any Global Procedures called by them; and
- It cannot be called from the Graphics Script (which is executed separately by each client), Screen Scripts (which are attached to individual screens), and Command animations.

For more information, see VBScript interfaces in the software, page 925.

To work around this limitation, create a Script Group that actually calls the function, configure an appropriate tag/expression trigger in the Execution box of the Script worksheet, and then create a project screen that somehow changes the value of that tag/expression. Therefore, when a user on a connected client uses the screen to change the value, the Script Group will be executed on the project server.

Examples

```
ExportSecuritySystem( "C:\security.txt" ) ExportSecuritySystem( "C:\security.txt", "mypa55w0rd" )
```

19.13.7 GetLastESignUser

"GetLastESignUser" is a built-in function that gets the last user who electronically signed an event during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
GetLastESignUser	Security	synchronous	supported	supported	supported

Syntax

```
GetLastESignUser()
```

This function has no parameters.

Returned value

This function returns (as a string) the name of the last user who electronically signed an event during run time. (Such events are generated when the user uses a screen object that has the E-Sign option selected or triggers a script that calls the function CheckESign, page 676.

For more information, see Events and event logs, page 326.) If the user failed to provide a valid username and password, or if the user clicked "Cancel" to exit the E-Sign dialog box, this function returns an empty string (" ").

Notes

This function gets the last user on the client where the function is executed. If the function is executed on the project runtime server, it gets the last user of the server's local Viewer.

19.13.8 GetSecuritySystemStatus

This function gets the status of the security system and its connection to the authentication server if the security mode is either Distributed–Client or Domain (LDAP).

Function	Group	Execution	Windows	Embedded	Thin Client
GetSecuritySystemStatus	Security	synchronous	supported	supported	supported

Syntax

```
GetSecuritySystemStatus()
GetSecuritySystemStatus( optNumType )
```

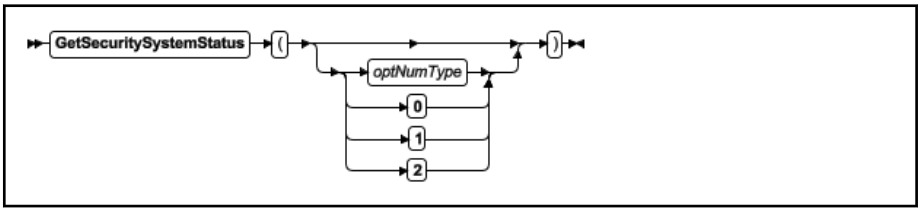


Fig. 19-21: Syntax

Inputs

optNumType

Specifies the actions to be carried out in case of a status update.

Value	Description
0	Execute a fast check using either Ping or Bind (depending on the server settings), however, take no other actions
1	Force reload of users and groups from the authentication server
2	Clear cached users and groups

This is an optional parameter. If no value is specified, the default value is "0".

Returned value

This function returns the following possible values:

Value	Security Mode is Distributed–Client	Security Mode is Domain (LDAP)
0	No cache	Connection timeout
1	Updated cache	Bind timeout
2	Outdated local cache	Query timeout
3	Outdated server cache	Disconnected
4	Disconnected from server	Connected
5	N/A	Query returned no users or groups
6	N/A	Invalid user or group

Notes

This function returns the same value that is sent to the project tag configured in the **Status Tag** field, in the Configuring server settings for security modes, page 466.

Additionally, there are other results besides calling this function that update the status:

- If a user logs on to the project. Specifically, if the user logs on via the built-in "LogOn" dialog (opened by either calling the "LogOn" function or by selecting the **LogOn** menu command in the Viewer), the status is updated before the dialog is displayed.
- If the authentication server is offline and the retry interval (configured in the security system server settings) has elapsed.
- If the security system settings are opened in the development environment.

Whenever the status is updated, the new value is immediately sent to the project tag configured in the **Status Tag** field.

Examples

Getting the status of the security system:

```
GetSecuritySystemStatus()
```

Force the security system to reload all users and groups from the authentication server:

```
GetSecuritySystemStatus( 1 )
```

19.13.9 GetUserFullName

This function gets the full name (if specified) of a specified user in the project security system.

Function	Group	Execution	Windows	Embedded	Thin Client
GetUserFullName	Security	synchronous	supported	supported	supported

Syntax

```
GetUserFullName(strUserName,"tagUserFullName")
```

Inputs

strUserName

Name of a user in the project security system.

tagUserFullName

The name of a tag (String type) that is to receive the full name of the specified user after the function is called. If no full name is defined for the specified user, the tag receives an empty string ("").

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

This function returns the following possible values:

Value	Description
0	Specified user does not exist
1	Successful; the specified user exists

Examples

Getting the full name of the currently logged-on user (via the system tag "UserName"):

```
GetUserFullName( UserName, "UserFullName" )
```

Getting the full name of the user "engineer1":

```
GetUserFullName( "engineer1", "UserFullName" )
```

19.13.10 GetUserNames

Gets the name of all users; stores the names in the UsersArray tag

Function	Group	Execution	Windows	Embedded	Thin Client
GetUserNames	Security	synchronous	supported	supported	executed on the server

Syntax

```
GetUserNames( "tagUsers", optNumUserType, "opttagGroups" )
```

Inputs

tagUsers

Name of the array tag that is to receive the list of user names

optNumUserType

Type of the user to get

0	Returns all users
1	Only users created during runtime
2	Only users created using the development environment

opttagGroups

Name of the array tag that is to receive the list of group names for each user

Returned value

The number of users or a negative number from the following list:

−1	Invalid number of parameters
−2	tagUsers is invalid
−3	optNumUserType is invalid
−4	opttagGroups is invalid
−5	Error, function cannot be called in the Thin Client

Examples

Retrieves all users, stores the names in the UsersArray tag:

```
GetUserNames("UsersArray")
```

Retrieves all users created during runtime, stores the names in the UsersArray tag:

```
GetUserNames("UsersArray", 1)
```

Retrieves all users created in the development environment, stores the names in the UsersArray tag:

```
GetUserNames("UsersArray", 2)
```

Retrieves all users created in the development environment, stores the names in the UsersArray tag and the group names per each user in the Groups tag:

```
GetUserNames("UsersArray", 2, "Groups")
```

19.13.11 GetUserPwdAging

The function GetUserPwdAging gets the age of the password for a specified user - that is, the time remaining until the password expires, or if it has expired, the time since it expired.

Function	Group	Execution	Windows	Embedded	Thin Client
GetUserPwdAging	Security	synchronous	supported	supported	supported

Syntax

```
GetUserPwdAging(strUser)
```

Inputs

strUser
User name of which the password aging is checked.

Returned value

≤0	Number of hours since the password expired
0	Specified user is not logged on
>0	Number of hours remaining until the password expires

☞ If the function is not executed properly (e.g., User Name is invalid) or if the specified user is not logged on, the function returns BAD quality.

Examples

Get the age of the password for the user "John":

```
GetUserPwdAging("John")
```

Get the age of the password for the user who is currently logged on:

```
GetUserPwdAging(UserName)
```

19.13.12 GetUserState

Gets the current status of a selected user.

Function	Group	Execution	Windows	Embedded	Thin Client
GetUserState	Security	synchronous	supported	supported	supported

Syntax

```
GetUserState(strUserName)
```

Inputs

strUserName
The name of the user.

Returned value

This function returns the following possible values:

Value	Description
-3	Specified user does not exist
-2	Incorrect parameter type
-1	Invalid number of parameters
0	Specified user is unblocked
1	Specified user is blocked

Examples

Tag name	Expression
Tag	GetUserState("Bob")
Tag	GetUserState ("Albert")

19.13.13 ImportSecuritySystem

This function imports a security system configuration from an external file.

Function	Group	Execution	Windows	Embedded	Thin Client
ImportSecurity-System	Security	synchronous	supported	supported	supported

Syntax

```
ImportSecuritySystem(strSecuritySystemPassword, strFileName, strFilePassword, optNumMode)
```

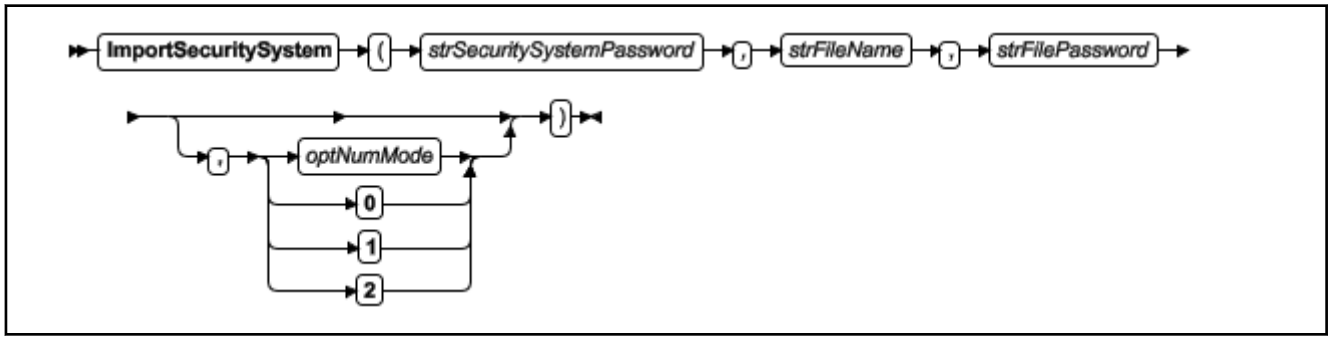


Fig. 19-22: Syntax

Inputs

strSecuritySystemPassword

The main password for the current security system configuration of the project. (The security system has to be enabled.)

strFileName

The complete file path and name of the configuration file be imported. (Before, the file has to be exported from a "WinStudio" project using either the Backing up the security system configuration, page 481 or the ExportSecuritySystem, page 680 function.)

strFilePassword

The password for the specified configuration file.

optNumMode

A numeric value specifying how the imported settings are to be handled:

Value	Description
0	Attaching the imported settings to the current settings. In the event of a conflict, the current settings are replaced by the imported settings.
1	Attaching the imported settings to the current settings. In the event of a conflict, the current settings are kept.
2	Replace the current settings completely by the imported settings.

This is an optional parameter. If no value is specified, the default value is "0".

Returned value

This function returns the following possible values:

Value	Description
-2	Incorrect parameter type
-1	Invalid number of parameters

Value	Description
0	Could not read security data
1	File imported successfully

Examples

```
ExportSecuritySystem( "curr3ntPa55w0rd", "C:\security.txt", "1mp0rtPa55w0rd" )
ExportSecuritySystem( "curr3ntPa55w0rd", "C:\security.txt", "1mp0rtPa55w0rd", 2 )
```

19.13.14 RemoveUser

Removes a user from the WinStudio security system.

Function	Group	Execution	Windows	Embedded	Thin Client
RemoveUser	Security	synchronous	supported	supported	supported

Syntax

```
RemoveUser( strUserName )
```

Inputs

strUserName

The name of the user to be removed.

Returned value

This function returns the following values:

Value	Description
0	User removed successfully
1	Invalid number of parameters
2	Incorrect parameter type
3	User currently logged on does not have the rights to remove (i.e., user does not have Edit Security System)
4	User cannot be removed
5	Specified user does not exist
6	Component-level failure
7	Failed to save to configuration file
8	The current Security mode does not allow user to be removed

Examples

Remove the user named "Bob":

```
RemoveUser("Bob")
```

Remove the user specified by the tag InvalidUser:

```
RemoveUser(InvalidUser)
```



This function can only be used to remove users created using the function CreateUser( ), page 678.

You cannot use this function to remove users who were created using the Security System configuration tools in the project development environment.

For more information, see Project security, page 461.

19.13.15 SetPassword

Specifies a new password for a user in your project's security system.

Function	Group	Execution	Windows	Embedded	Thin Client
SetPassword	Security	synchronous	supported	supported	supported

Syntax

```
SetPassword(optStrUserName,optStrNewPassword,optStrConfirmPassword,optStrCurrentPassword)
```

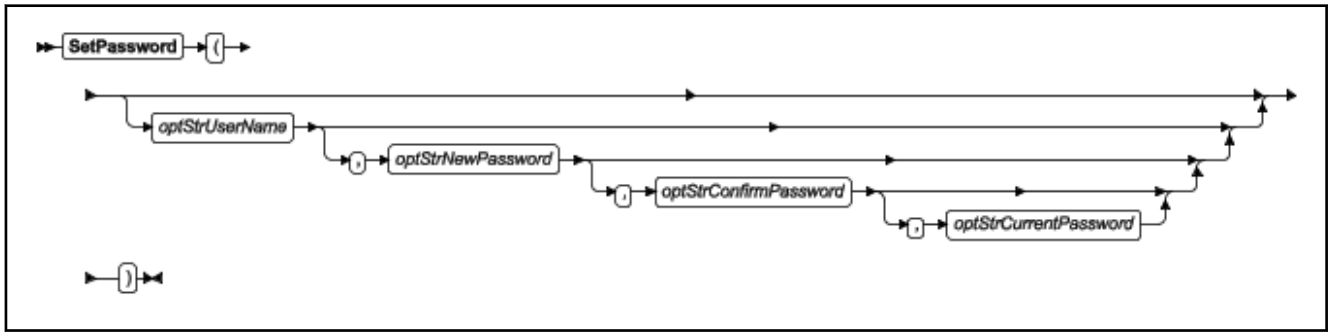


Fig. 19-23: Syntax

Inputs

optStrUserName

The name of the user.

This is an optional parameter; if no value is specified, a dialog box will be displayed on the client so that the current user can select the name.

optStrNewPassword

The new password for the specified user.

This is an optional parameter; if no value is specified, a dialog box will be displayed on the client so that the current user can type the password.

optStrConfirmPassword

The new password again, to confirm that it has been typed correctly.

This is an optional parameter; if no value is specified, a dialog box will be displayed on the client so that the current user can type the password.

optStrCurrentPassword

The current password for the specified user, to authorize the change.

This is an optional parameter; if no value is specified, a dialog box will be displayed on the client so that the current user can type the password.

Returned value

This function returns a numeric value with the following meaning:

Value	Description
-1000	Could not display dialog box, because the function was called on the project runtime server. The function should be called on the client.
-1	Internal error; contact technical support
0	Password set successfully

Value	Description
1	Invalid number of parameters
2	Incorrect parameter type
3	Specified user does not exist
4	Reentrant call not allowed
5	User clicked <Cancel>
6	Group does not exist
7	Password too weak
8	Invalid password
9	Invalid user
10	The current user does not have sufficient privileges to set user passwords (i.e., the current user does not have the option **Edit Security System** selected in the Security System settings).
11	Server offline.
12	Communication error (e.g., the function was called on remote client that cannot communicate with the LDAP server).
13	Confirmation does not match new password.
14	Set Password dialog box is already displayed, cannot display another dialog box. (For example, if the user clicked <OK> without providing all of the required information.)
15	The current security mode does not allow user passwords to be changed.

Notes If you do not want the dialog box to be displayed on the client, you must provide all four paramaters to the function. Otherwise, the dialog box will be displayed in order to get the remaining information from the current user.

Examples *Display the Set Password dialog box on the client, to get the information from the current user:*

```
SetPassword()
```

Change the password for user "Admin", but display the dialog box in order to get the new password and authorization:

```
SetPassword("Admin")
```

Change the password for user "Admin" to the value of the tag newPassword, but display the dialog box in order to get authorization:

```
SetPassword("Admin",newPassword,newPassword)
```

Change the password for user "Admin" to the value of the tag newPassword, without displaying the dialog box:

```
SetPassword("Admin",newPassword,newPassword,"DLfVU89Y")
```

19.13.16 UnblockUser

This function unblocks a blocked user in the security system.

Function	Group	Execution	Windows	Embedded	Thin Client
UnblockUser	Security	synchronous	supported	supported	supported

Syntax

```
UnblockUser( strUserName )
```

Inputs

strUserName

The name of the user to be unblocked.

Returned value

This function returns the following possible values:

Value	Description
0	User successfully unblocked
1	Invalid number of parameters
2	Incorrect parameter type
3	Specified user does not exist
4	User currently logged on does not have the rights to unblock (i.e., user does not have Edit Security System)
5	The operation on the distributed security system failed
6	Specified user cannot be blocked
7	The current Security mode does not allow users to be blocked/ unblocked
8	Internal error

Notes

If you use this function to unblock a user, then the option **User is blocked** is cleared in the User Account dialog.

For more information, see Creating and configuring users, page 478.

Examples

Unblock the user named "Bob":

```
UnblockUser("Bob")
```

Unblock the user specified by the tag BlockedUser:

```
UnblockUser (BlockedUser)
```

19.14 Module Activity Functions

19.14.1 Module Activity Functions – General Information

These functions are used to manage the various runtime modules of a project, e.g., background tasks, the data server, and the project viewer, as well as module interaction with the operating system.

Funktion	Description
AppActivate, page 692	Activates (i.e., brings to the front) another application window that is already open.
AppIsRunning, page 693	Verifies another application window is open and running.
AppPostMessage, page 694	Is a built-in scripting function that sends a Windows system message to another application window.
AppSendKeys, page 695	Sends keyboard commands to the active application.

Funktion	Description
CleanReadQueue, page 696	Removes all read messages from the communications module.
CloseSplashWindow, page 696	Closes the WinStudio splash screen.
DisableMath, page 697	Is a built-in function that pauses the execution of all Math worksheets.
EnableMath, page 697	Is a built-in function that resumes the execution of all Math worksheets.
EndTask, page 697	Stops a specified run-time module that is currently running.
ExitWindows, page 698	Exits the Windows operating system in a specified manner.
IsScreenOpen, page 699	Checks if a WinStudio screen is open on a client.
IsTaskRunning, page 699	Verifies that a background task is running.
IsViewerInFocus, page 700	Verifies that the project viewer (Viewer.exe) is in focus on the display.
KeyPad, page 700	Displays a Virtual Keyboard to prompt the runtime user to enter a tag value.
LogOff, page 702	Logs off the current user and then logs on the default user (typically "guest").
LogOn, page 702	This function either logs on a specified user or displays a Log On dialog.
Math, page 703	Executes the specified Math worksheet.
PostKey, page 703	Posts virtual-key codes to the currently displayed project screen. In other words, it simulates keyboard input.
Recipe, page 704	Executes the specified Recipe worksheet.
Report, page 705	Executes the specified Report worksheet and sends the output to hard disk, printer, or PDF.
RunGlobalProcedureAsync, page 706	Executes a global procedure asynchronously, in its own thread, so that it does not slow down or interfere with other running scripts. The procedure is run on the project server, but it can be called by any local or remote client.
RunGlobalProcedureAsyncGetStatus, page 708	Gets the status of one or more global procedures that were run asynchronously by calling the RunGlobalProcedureAsync function. Each procedure is run in its own thread, so that it does not slow down or interfere with other threads.
RunGlobalProcedureOnFalse, page 710	Runs a global procedure when the value of a specified project tag or expression becomes FALSE.
RunGlobalProcedureOnServer, page 711	Runs a specified VBScript procedure, as defined in the Procedures folder in the Project Explorer. The procedure is run on the project runtime server, but it can be triggered by any client that calls this function.
RunGlobalProcedureOnTrigger, page 712	Runs a global procedure when the value or quality of a specified tag changes.
RunGlobalProcedureOnTrue, page 714	Runs a global procedure when the value of a specified project tag or expression becomes TRUE.
RunVBScript, page 715	Executes a statement in VBScript language.
SecureViewerReload, page 716	Closes the Secure Viewer program and then reloads it with a new configuration file.
SendKeyObject, page 716	Sends a key event code to objects in the currently displayed project screen. You can use this function to trigger Command animations on those objects.
SetAppPath, page 719	Sets the new file path for the project folder. After this function is executed, WinStudio will look for all of the project files (i.e.,screens, alarms, trends, database, events) in this folder.

Funktion	Description
SetViewerInFocus, page 720	Moves the Viewer program window in front of all other open windows and then maximizes it to fill the display.
SetViewerPos, page 720	Sets the height, width, and position of the project viewer or thin client.
ShutDown, page 721	Stops all execution tasks and runtime modules, effectively shutting down the project.
StartTask, page 721	Starts a specified run-time module that is not currently running.
ViewerPostMessage, page 722	Posts a Windows System Message to the specified project screen.
Wait, page 723	Pauses the project for a specified number of milliseconds.
WinExec, page 724	Executes a Windows command as if it was entered at the command prompt.
WinExecIsRunning, page 726	Indicates whether a Windows program previously executed by the function WinExec is still running.

19.14.2 AppActivate

"AppActivate" is an internal scripting function that activates (i.e., brings to the front) another application window that is already open.

Function	Group	Execution	Windows	Embedded	Thin Client
AppActivate	Module activity	asynchronous	supported	supported	supported

Syntax

```
AppActivate(strAppTitle{ | ,{ optNumActiv | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }{ |,optNumTimeout } })
```

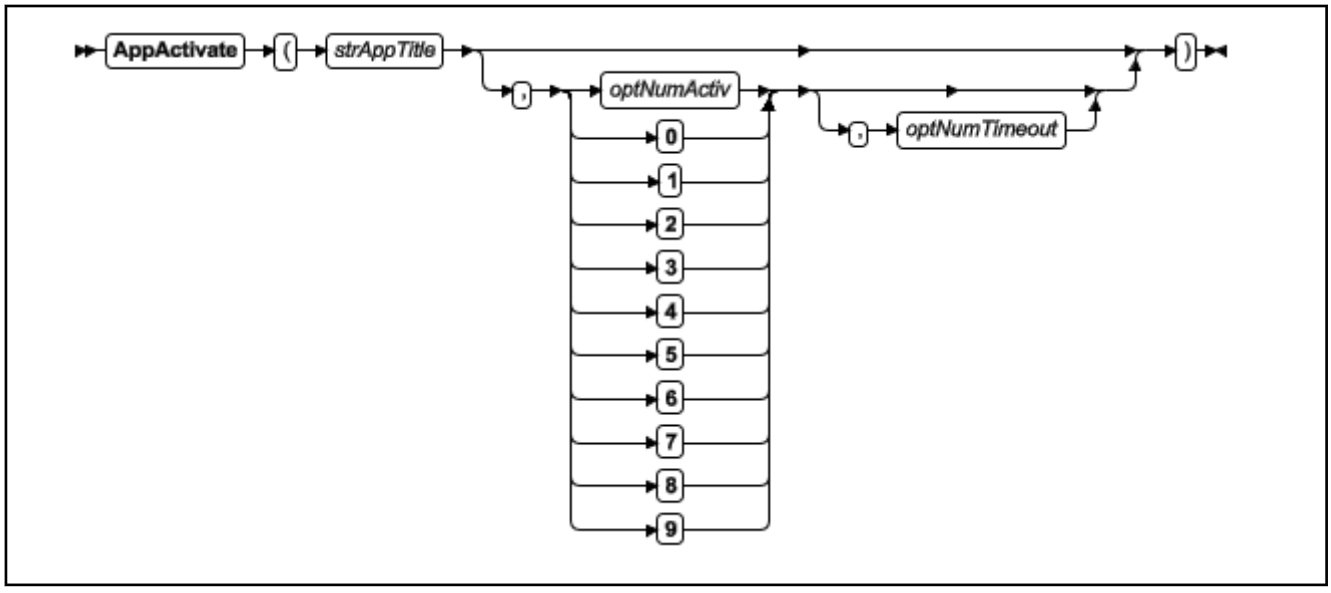


Fig. 19-24: Syntax

Inputs

strAppTitle

The full title (as shown in the title bar) of the application window.

optNumActiv

Controls how the specified window is to be activated:

Value	Command	Description
0	SW_HIDE	Hides the currently active window and then activates the specified window.
1	SW_SHOWNORMAL	Activates and displays the specified window. If the window is minimized or maximized, it is restored to its original size and position. This command is to be used if a window is displayed for the first time.
2	SW_SHOWMINIMIZED	Activates the specified window and then minimizes it.
3	SW_SHOWMAXIMIZED	Activates the specified window and then maximizes it.
4	SW_SHOWNOACTIVATE	Displays the specified window but does not activate it. If the window is minimized or maximized, it is restored to its original size and position.
5	SW_SHOW	Activates and displays the specified window in its current size and position. This function is similar to **SW_SHOWNORMAL** except that if the window is minimized or maximized, it remains in that state.
6	SW_MINIMIZE	Minimizes the specified window and then activates the next open window.
7	SW_SHOWMINNOACTIVATE	Displays the specified window as a minimized window but does not activate it.
8	SW_SHOWNA	Displays the specified window in its current size and position but does not activate it. This function is similar to SW_SHOWNOACTIVATE except that if the window is minimized or maximized, it remains in that state.
9	SW_RESTORE	Activates and displays the specified window. If the window is minimized or maximized, it is restored to its original size and position. This command is to be used when restoring a minimized window.

This parameter is optional. If no value is specified, the default command is "SW_RESTORE".

optNumTimeout

The timeout period (in milliseconds) for the function to be successfully executed. If the function is not executed in this period for any reason, it is aborted.

This parameter is optional. If no value is specified, the default timeout is five seconds (or 5000 milliseconds).

Returned value

This function returns one of the following values:

Value	Description
0	**ERROR**: The specified application window was not activated or did not respond within the specified timeout period.
1	**SUCCESS**: The specified application window was successfully activated.

☞ "AppActivate" is similar to the function ShowWindow in the Microsoft Windows API and enables many of the same options. For further information, please refer to the Windows API documentation.

Examples

Displaying the Microsoft Word document named "test.doc":

AppActivate("test.doc — Microsoft Word", 5)

19.14.3 AppIsRunning

"AppIsRunning" is an internal scripting function that checks if another application window (e.g., a Microsoft Word document) is open and running.

Function	Group	Execution	Windows	Embedded	Thin Client
AppIsRunning	Module activity	synchronous	supported	supported	supported

Syntax

```
AppIsRunning(strAppTitle{ | ,optNumTimeout })
```

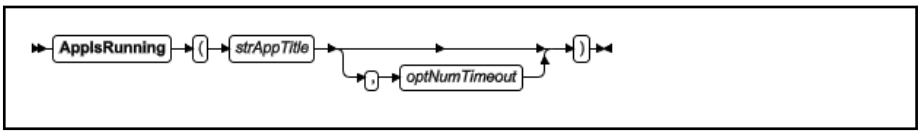


Fig. 19-25: Syntax

Inputs

strAppTitle

The full title (as shown in the title bar) of the application window.

optNumTimeout

The timeout period (in milliseconds) for the function to be successfully executed. If the function is not executed in this period for any reason, it is aborted.

This parameter is optional. If no value is specified, the default timeout is five seconds (or 5000 milliseconds).

Returned value

This function returns one of the following values:

Value	Description
0	**ERROR**: The specified application window is not open or did not respond within the specified timeout period.
1	**SUCCESS**: The specified application window is open and running.

☞ "AppIsRunning" is similar to the function IsWindow in the Microsoft Windows API. For further information, please refer to the Windows API documentation.

Examples

Verifying that the Microsoft Word document named "test.doc" is open and running:

AppIsRunning("test.doc — Microsoft Word")

19.14.4 AppPostMessage

"AppPostMessage" is an internal scripting function that sends a Windows system message to another application window.

Function	Group	Execution	Windows	Embedded	Thin Client
AppPostMessage	Module activity	synchronous	supported	supported	supported

Syntax

```
AppPostMessage(strAppTitle,strMessage,numWParam,numLParam{ | ,optNumTimeout })
```



Fig. 19-26: Syntax

Inputs

strAppTitle

The full title (as shown in the title bar) of the application window.

strMessage

The name or code of the system message.

The "CLOSE", "MINIMIZE", "MAXIMIZE" and "RESTORE" messages can be entered as string values enclosed in quotation marks. All other message codes have to be given as numeric values.

numWParam

Additional message-specific information.

numLParam

Additional message-specific information.

optNumTimeout

The timeout period (in milliseconds) for the function to be successfully executed. If the function is not executed in this period for any reason, it is aborted.

This parameter is optional. If no value is specified, the default timeout is five seconds (or 5000 milliseconds).

Returned value

This function returns one of the following values:

Value	Description
0	**ERROR**: The specified application window is not open or did not respond within the specified timeout period.
1	**SUCCESS**: The specified application window is open and running.

"AppPostMessage" is similar to the function PostMessage in the Microsoft Windows API and enables many of the same options. For further information including a list of available system messages, please refer to the Windows API documentation.

Examples

Closing the Calculator application:

AppPostMessage("Calculator", "CLOSE", 3, 1)

19.14.5 AppSendKeys

Sends keyboard commands to the active application.

Function	Group	Execution	Windows	Embedded	Thin Client
AppSendKeys	Module activity	synchronous	supported	not supported	supported

Syntax

```
AppSendKeys( strKeys1, strKeys2, … , strKeysN )
```

Inputs

strKeys (1-N)
String tags containing the keyboard commands to be used.

Returned value No returned values.

Examples

Tag name	Expression
	AppSendKeys("S", "t", "u", "d", "i", "o", "<ENTER>")
	AppSendKeys("<Alt>F")

☞ <ALT>, <CTRL>, or <SHIFT> can be specified in the text to send a code equal to the "Alt", "Ctrl" or "Shift" keyboard commands. To send the "<" character, specify "<<" in the text.

19.14.6 CleanReadQueue

Removes all reading messages from the communications module.

Function	Group	Execution	Windows	Embedded	Thin Client
CleanReadQueue	Module activity	synchronous	supported	not supported	executed on the server

Syntax

```
CleanReadQueue()
```

This function does not have any parameters.

Returned value No returned values.

Examples

Tag name	Expression
	CleanReadQueue()

☞ This function is not to be used in new projects. For compatibility reasons, however, it is still available for "WinStudio" projects created with previous versions.

19.14.7 CloseSplashWindow

Closes the "WinStudio" splash screen.

Function	Group	Execution	Windows	Embedded	Thin Client
CloseSplash-Window	Module activity	synchronous	supported	not supported	executed on the server

Syntax

```
CloseSplashWindow()
```

This function does not have any parameters.

Returned value No returned values.

Examples

Tag name	Expression
	CloseSplashWindow()

19.14.8 DisableMath

"DisableMath" is an internal scripting function that stops the execution of all math worksheets.

Function	Group	Execution	Windows	Embedded	Thin Client
DisableMath	Module activity	asynchronous	supported	supported	executed on the server

Syntax

```
DisableMath()
```

This function does not have any parameters.

Returned value

There are no returned values for this function.



To resume the execution of math worksheets, page 400, call the EnableMath, page 697, function.

Examples

Tag name	Expression
	DisableMath()

19.14.9 EnableMath

"EnableMath" is an internal scripting function that reactivates the execution of all math worksheets.

Function	Group	Execution	Windows	Embedded	Thin Client
EnableMath	Module activity	asynchronous	supported	supported	executed on the server

Syntax

```
EnableMath()
```

This function does not have any parameters.

Returned value

There are no returned values for this function.



In most cases, execution was paused by calling the DisableMath, page 697 function.

Examples

Tag name	Expression
	EnableMath()

19.14.10 EndTask

Terminates the "WinStudio" module that is currently running.

Function	Group	Execution	Windows	Embedded	Thin Client
EndTask	Module activity	asynchronous	supported	supported	executed on the server

Syntax

```
EndTask(strTask)
```

Inputs

strTask

The name of the task to be stopped (from the following list):

BGTASK	Background tasks
VIEWER	Viewer
DBSPY	Database Spy
LOGWIN	LogWin
DRIVERDriver-Name	Driver
UNIDDECL	DDE client
UNINDDE	DDE server
UNIODBC	ODBC
TCPSERVER	TCP/IP server
TCPCLIENT	TCP/IP client
OPCCLIENT	OPC

Returned value No returned values.

Examples

Tag name	Expression
	EndTask("Viewer")

☞ To close a driver, the following syntax is to be used:

EndTask("DriverDriverName")

Where DriverName is the name of the ***.dll** file of the driver. For example:

EndTask("DriverMODBU")

19.14.11 ExitWindows

Terminates the Windows operating system in a specified manner.

Function	Group	Execution	Windows	Embedded	Thin Client
ExitWindows	Module activity	asynchronous	supported	not supported	supported

Syntax

```
ExitWindows(numExitCode)
```

Inputs

numExitCode

A numeric code specifying how Windows is to be terminated:

Value	Description
0	Restart
1	Log off
2	Shut down

Returned value No returned values.

Examples

Tag name	Expression
	ExitWindows(1)

19.14.12 IsScreenOpen

Checks if a WinStudio screen is open on a client.

Function	Group	Execution	Windows	Embedded	Thin Client
IsScreenOpen	Module activity	asynchronous	supported	supported	not supported

Syntax

```
IsScreenOpen(strScreen{ | ,optNumID })
```

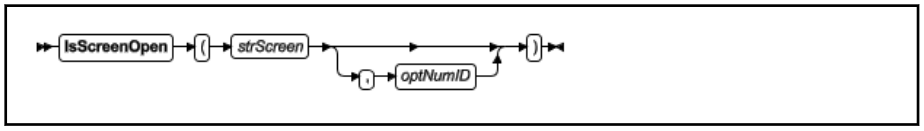


Fig. 19-27: Syntax

Inputs

strScreen

The name of the WinStudio screen to be checked.

optNumID

The unique instance number of the screen. (The number is assigned when the screen is opened with the Open, page 666 function.)

This parameter is optional. If no value is specified, the default ID is 0.

Returned value

0	The screen is not open.
1	The screen is open.

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples

Is the screen named "main" open?

```
IsScreenOpen("main")
```

Is the screen named "main" with ID 10 open?

```
IsScreenOpen("main",10)
```

19.14.13 IsTaskRunning

Checks whether a background task is running.

Function	Group	Execution	Windows	Embedded	Thin Client
IsTaskRunning	Module activity	synchronous	supported	supported	executed on the server

Syntax

```
IsTaskRunning(strTask)
```

Inputs

strTask

The name of the task to be checked.

Returned value

0	Task is not running.
1	Task is running.

Examples

Is the project viewer (**Viewer.exe**) running?

IsTaskRunning("Viewer")

19.14.14 IsViewerInFocus

Checks whether the project viewer ("Viewer.exe") is in focus on the screen.

Function	Group	Execution	Windows	Embedded	Thin Client
IsViewerInFocus	Module activity	synchronous	supported	not supported	not supported

Syntax

```
IsViewerInFocus()
```

This function does not have any parameters.

Returned value

0	Viewer is not in focus.
1	Viewer is in focus.

Examples

IsViewerInFocus - Examples

```
IsViewerInFocus()
```

19.14.15 KeyPad

Displays a virtual keyboard to prompt the runtime user to enter a tag value.

Function	Group	Execution	Windows	Embedded	Thin Client
KeyPad	Module activity	asynchronous	supported	supported	executed on the server

Syntax

```
KeyPad("strTagName",optStrKeyboardName,optNumIsPassword,optStrHint,optNumMin,
                                optNumMax,optNumESign,optStrConfirmation)
```

Inputs

strTagName

The name of the tag that is to be written to by means of the "Virtual Keyboard".

This name has to be enclosed in quotation marks. Otherwise, the project tries to get the contents of the tag.

optStrKeyboardName

Type of "Virtual Keyboard" (e.g., `AlphaNumeric`, `Enhanced Keypad` or `Keypad`). If this parameter is omitted, the default Virtual Keyboard is launched.

optNumIsPassword

If this parameter is set at any value different from 0 (zero), the characters typed in the Virtual Keyboard appear as asterisks ("*"). This option is useful if the user is entering a password.

optStrHint

The value specified for this parameter is displayed in the title bar of the Virtual Keyboard window if the **ShowHint** option is selected in the project settings, page 61.

optNumMin, optNumMax

Numeric minimum and maximum values for the tag when using the keyboard type "Keypad" if the option **Show MIN/MAX fields** is enabled in the project settings, page 61. (These values are ignored for all other keyboard types.) These are optional parameters. For implementation, however, both have to be specified. If only one parameter is specified – e.g., Min but not Max – it is ignored.

optNumESign

If this parameter is set with any value different from 0 (zero), the user is prompted to log on to complete the input and the action is noted in the log.

optStrConfirmation

The value specified for this parameter is displayed in a confirmation dialog that has to be confirmed by the user to complete the input.



Confirmation cannot be automated or bypassed. The dialog can only be confirmed by the user actually pressing a key or mouse button.

This parameter is optional. If no value is specified, no dialog is displayed.

Returned value

Error	Description
0	Successful
1	Error
2	Tag does not exist
3	Reentrant error, function is already running
4	Invalid number of parameters
5	Internal error, contact the technical support for more information

Examples

Tag name	Expression
Tag	KeyPad("tagA")
Tag	KeyPad("tagA", "EnhKeypad")
Tag	KeyPad("tagA", "EnhKeypad", 1)
Tag	KeyPad("tagA", "EnhKeypad", 1, "My Input", 0, 100)

19.14.16 LogOff

This function logs off the current user and logs on the default user (typically "guest").

Function	Group	Execution	Windows	Embedded	Thin Client
LogOff	Module activity	asynchronous	supported	supported	supported

Syntax

```
LogOff()
```

This function does not have any parameters.

Returned value

This function returns the following possible values:

Value	Description
0	Error
1	Successful

Examples

LogOff - Examples

```
LogOff()
```

19.14.17 LogOn

This function either logs on a specified user or displays a "Log On" dialog.

Function	Group	Execution	Windows	Embedded	Thin Client
LogOn	Module activity	asynchronous	supported	supported	supported

Syntax

```
LogOn({ | optStrUsername,optStrPassword })
```

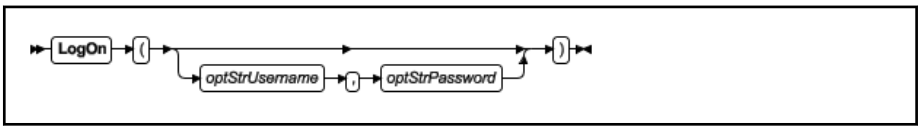


Fig. 19-28: Syntax

Inputs

optStrUsername

Name of the user to be logged on.

optStrPassword

Password of the specified user.

"optStrUsername" and "optStrPassword" are optional parameters. If they are not specified, the project displays a Log On dialog, page 482 instead to prompt the current operator of the station to log on.

Returned value

This function returns the following possible values:

Value	Description
0	Error (e.g., user name or password is invalid) or cancellation
1	Successful

Notes

The following limitations also apply to how the function can be used in project screens in Mobile Access:

The function cannot be called from the sub-routine `Screen_OnClose` in the `Screen_OnClose` im Screen Script, page 932 aus aufgerufen werden; and

You should avoid creating a script that both shows a dialog box (e.g., a message box that is shown using the VBScript function MsgBox, page 956 or the WinStudio built-in function ShowMessageBox, page 673) and opens or closes a screen.

The Mobile Access web interface can display only one screen at a time, and opening a new screen will automatically close the previous one. Depending on the order of execution, this can lead to unexpected behavior such as a dialog box being orphaned or not being shown at all.

Examples

Displaying a Log On dialog:

```
LogOn()
```

Log on username Albert with password EMC2:

```
LogOn( "Albert", "EMC2" )
```

19.14.18 Math

Executes the specified math worksheet.

Function	Group	Execution	Windows	Embedded	Thin Client
Math	Module activity	synchronous	supported	supported	supported

Syntax

```
Math( numWorksheet )
```

Inputs

numWorksheet
Number of the math worksheet to be executed.

Returned value

No returned values.

Examples

Tag name	Expression
	Math(6)

NOTICE

Running a math worksheet from inside another module pauses that module until the math worksheet finishes.

For this reason, this function is only to be used if it is absolutely necessary to avoid decreasing the performance of the other modules.

19.14.19 PostKey

Sends a keyboard code to the currently displayed WinStudio screen.

Function	Group	Execution	Windows	Embedded	Thin Client
PostKey	Module activity	synchronous	supported	supported	supported

Syntax

```
PostKey(numKeydownOrKeyup, numwParam, numlParam)
```

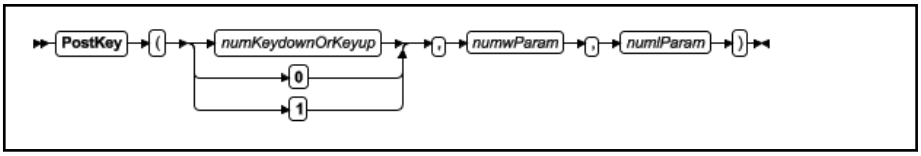


Fig. 19-29: Syntax

Inputs

numKeydownOrKeyup

A numerical flag with the following possible values:

- 0: KeyDown event
- 1: KeyUp event

In most cases, this value should be 0 to indicate that the key is being pressed.

numwParam

The virtual-key code to be posted, in hexadecimal format. For a list of codes, go to: msdn.microsoft.com/library/dd375731.aspx

numlParam

Additional message information.

In most cases, this value should be `0` to indicate that there is no additional information.

Returned value

No returned values.

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples

Post the virtual-key code for the Home key:

```
PostKey(0,0x24,0)
```

19.14.20 Recipe

Executes the specified recipe worksheet.

Function	Group	Execution	Windows	Embedded	Thin Client
Recipe	Module activity	synchronous	supported	supported	supported

Syntax

```
Recipe( strFunction )
```

Inputs

strFunction

String tag specifing the operation to be performed and the Recipe worksheet to be used in the

`[Operation]:[Recipe sheet]` format.

Operations:

Save	Saves data to a data file.
Load	Loads data from a data file
Delete	Deletes a data file
Init	Initializes a data file with a value of 0 in all tags

Returned value

0	No error
1	If the tag is numeric
2	Expression does not contain ":"
3	Previous command to the invalid ":"
4	Task not found by the system
5	Disk error

Examples

Tag name	Expression
Tag	Recipe("Save:Recipe1")
Tag	Recipe("Load:Recipe5")



- The Background task has to be running to execute the recipe functions.
- If this function is called on a Thin Client, the command is sent to the server (via TCP/IP) and the Recipe task, page 395 on the server executes the command.

  For this reason, please note that tags configured with a Scope of **Local** rather than **Server** are still updated on the server.

19.14.21 Report

Executes the specified report worksheet and sends the output to the hard disk, printer or a PDF file.

Function	Group	Execution	Windows	Embedded	Thin Client
Report	Module activity	synchronous	supported	supported	supported

Syntax

```
Report( "strFunction" , optNumOrientation )
```

Inputs

strFunction

String specifying the operation to perform and the Report worksheet to output, using the syntax "`Operation:Report worksheet`", where…

`Operation` is either

- `Disk` (saves data file to hard disk),
- `Prn` (sends report to default printer), or
- `Pdf` (generates a PDF file of the report); and

`Report worksheet` is the name of the Report worksheet file (`*.rep`) to output.

optNumOrientation

Set the page orientation as follows:

- 0: (default) for Portrait,
- 1: for Landscape.

This parameter is ignored if the `Operation` is other than `Prn`.



Some features of this function are not supported when running the project on a Windows CE device: it cannot generate PDFs; it cannot change paper orientation using the **optNumOrientation** parameter; and it does not support report worksheets in RTF format.

Returned value

Value	Description
0	Successful
1	strFunction is configured with a numeric value (invalid)
2	strFunction does not contain ":" (invalid)
3	strFunction contains an invalid output type before the ":"
4	Background task is not running (see hint below)
5	Disk error (e.g., disk full, read-only file cannot be overwritten or invalid path)
6	Specified report worksheet does not exist



The Background task has to be running in order to execute this function. Otherwise, the function returns the error code "4".

For further information, refer to Execution Tasks, page 91.

Examples

Tag name	Expression
	Report("Disk:Report1.rep")
	Report("Prn:Report2.rep", 0)
	Report("Prn:Report3.rep", 1)
	Report("Pdf:Report1.rep")

19.14.22 RunGlobalProcedureAsync

This function executes a global procedure asynchronously in its own thread to ensure that other running scripts are not affected. The procedure is run on the project server, however, it can be called by any local or remote client.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureAsync	Module activity	asynchronous	supported	supported	not supported; see note

Syntax

```
RunGlobalProcedureAsync(strProcedureName{ | ,optStrArgument1,…,optStrArgumentN })
```

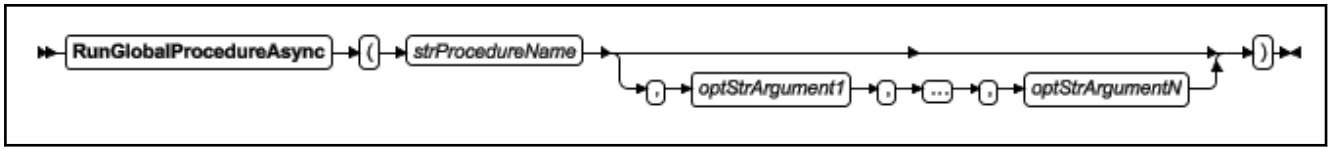


Fig. 19-30: Syntax

Inputs

strProcedureName

The name of the procedure (i.e., a VBScript function or sub-routine defined in the Procedures folder, page 927) to run asynchronously.

optStrArgument1, …, optStrArgumentN

Values that are transmitted to the parameters of the procedure. Arguments have to be transmitted as strings.

Returned value

If the procedure is successfully executed, this function returns a thread ID that can be used with the RunGlobalProcedureAsyncGetStatus, page 708 function. Otherwise, this function returns an error code:

Value	Description
-1	Function is not supported on Viewer / Web Thin Client
-2	Invalid number of parameters At least the procedure name has to be specified
-3	Maximum number of threads exceeded See note
-4	Failed to compile VBScript parameters for execution
-5	Failed to start the thread execution
-100	Internal error Please contact technical support

Notes

It is very important to note that this function can only be called by background tasks (e.g., Math, Script, Scheduler) on the project server. It cannot be directly called from any project client, even if the project client is running on the same workstation or device as the project server, because the client process is single-threaded. To indirectly call the function from a project client, configure a math, page 400 or script, page 401 worksheet to execute on a tag/expression trigger, and then configure a project screen to activate the trigger when needed.

For example, configure the worksheet to execute when the value of `MyTag` is 1, and then configure a Button screen object to toggle the value of `MyTag` when pressed.

Also, the maximum number of VBScript threads that can be executed asynchronously is configured by manually editing the project file (i.e., `projectname.APP`) to change the following setting:

```
[Script]
MaxAsyncThreads=8
```

The default number of threads is 8, but the only real limit is determined by the available system resources. Increasing the number of threads may decrease runtime performance.

Examples

Given the following procedure that is defined in the Procedures folder…

RunGlobalProcedureAsync - Examples

```
Function AddMe(intNumber)
  If intNumber >= 6 Then
    AddMe = 0
  Else
    AddMe = intNumber + 2
  End If
End Function
```

…the procedure is run by calling the **RunGlobalProcedureAsync** function…

```
RunGlobalProcedureAsync( "AddMe", "2" )
```

…and the function returns a thread ID that can be used with the RunGlobalProcedureAsyncGetStatus, page 708 function.

19.14.23 RunGlobalProcedureAsyncGetStatus

This function gets the status of one or more global procedures that were executed asynchronously by calling the "RunGlobalProcedureAsync" function. Each procedure is executed in its own thread so that other threads are not affected.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureAsyncGetStatus	Module activity	synchronous	supported	supported	not supported; see note

Syntax

```
RunGlobalProcedureAsyncGetStatus( )
RunGlobalProcedureAsyncGetStatus( optNumThreadID )
RunGlobalProcedureAsyncGetStatus("optTagThreadIDs", "optTagStatus", "optTagParameters")
```

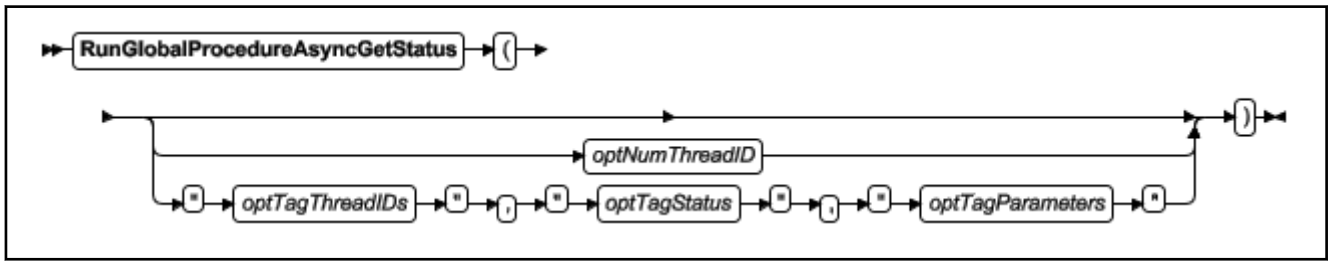


Fig. 19-31: Syntax

Inputs

optNumThreadID

The thread ID returned by the RunGlobalProcedureAsync, page 706 function if the procedure was successfully executed.

optTagThreadIDs

The name of an array tag that is to receive the thread IDs of all currently running and recently terminated threads.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

optTagStatus

The name of an array tag that is to receive the status of all currently running and recently terminated threads.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

optTagParameters

The name of an array tag that is to receive the parameters of all currently running and recently terminated threads.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

If the function succeeds, the possible returned values depend on how the function was called:

If the function was called with no parameters…

```
RunGlobalProcedureAsyncGetStatus( )
```

…then the returned value is the total number of threads that are currently running.

If the function was called with only the optNumThreadID parameter…

```
RunGlobalProcedureAsyncGetStatus( optNumThreadID )
```

…then the returned value is either 0, indicating that the thread is still running, or the value that was returned by the procedure.

If the function was called with the Array tags…

```
RunGlobalProcedureAsyncGetStatus( "optTagThreadIDs", "optTagStatus",
                                                     "optTagParameters" )
```

…then the tags will receive the appropriate values for all currently running and recently completed threads.

If the function fails, it returns one of the following error codes:

Value	Description
-1	Function is not supported on Viewer / Web Thin Client
-2	Invalid thread ID
-3	Invalid "optTagThreadIDs"
-4	Invalid "optTagStatus"
-5	Invalid "optTagParameters"
-100	Internal error Please contact technical support

Notes

It is very important to note that this function can only be called by background tasks (e.g., Math, Script, Scheduler) on the project server. It cannot be direct-

ly called from any project client, even if the project client is running on the same workstation or device as the project server, because the client process is single-threaded. To indirectly call the function from a project client, configure a math, page 400 or script, page 401 worksheet to execute on a tag/expression trigger, and then configure a project screen to activate the trigger when needed. For example, configure the worksheet to execute when the value of `MyTag` is 1, and then configure a Button screen object to toggle the value of `MyTag` when pressed.

Also, when the call to RunGlobalProcedureAsync, page 706 succeeds, it returns an ID for the thread created and starts running the procedure in that thread. The status of the thread is stored in an internal buffer and can be retrieved using the "RunGlobalProcedureAsyncGetStatus" function.

The buffer gets cleared when:

- The "RunGlobalProcedureAsyncGetStatus" function has been called and the thread status is different from 0 (thread is running); or
- The maximum buffer size has been exceeded, the thread is no longer running, and a call to start a new thread has been made.

19.14.24 RunGlobalProcedureOnFalse

This function directly executes a global procedure if the value of a specified tag/expression becomes FALSE.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureOnFalse	Module activity	synchronous	supported	supported	supported

Syntax

```
RunGlobalProcedureOnFalse("strCondition",strProcedureOnFalse)
```

Inputs

strCondition

A project tag or expression



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

strProcedureOnFalse

The name of the procedure (i.e., a VBScript function or sub-routine defined in the Procedures, page 927 folder) to run if the value of **strCondition** becomes FALSE (zero).

Returned value

This function returns the following possible values:

Value	Description
0	Error
1	Successful

Notes

If this function is called, it is continuously executed by the project runtime client (i.e., the Viewer) until either it or the project runtime server is stopped.

That means every time the value of the condition becomes FALSE, the procedure is run. However, the procedure is run only once when the value of the condition becomes FALSE; it is not repeatedly run while the value of the condition is FALSE.

For the procedure to run again, the value of the condition must become TRUE and then FALSE again.

Also, the function can be called more than once, so that the same procedure can be run by different triggers. The project runtime client manages the execution of all instances of the function.

The procedure is run on the client where this function was called. To run a procedure on the project server, use the function RunGlobalProcedureOnServer, page 711.

The value of strCondition is passed to the procedure as an argument, so the procedure should be written to receive it.

For example:

```
Sub UsingOnFalse(strCondition)
  .
  .
  .
End Sub
```

Please note that you do not actually have to use the argument in your procedure, only that you should write the procedure to receive it.

No other arguments can be passed to the procedure.

To ensure compatibility with previous versions of Rexroth WinStudio, passing the argument is disabled by default in existing projects and enabled by default in new projects. To change this for your project, open your project file (`projectname.APP`) in a text editor and then edit the following property:

```
[Options]
EnableTagNameOnRunGlobalProcedureOnTag=<0 (disabled) / 1 (enabled)>
```

Examples

When the value of TagOnFalse becomes FALSE, run the procedure UsingOnFalse:

```
RunGlobalProcedureOnFalse("TagOnFalse","UsingOnFalse")
```

19.14.25 RunGlobalProcedureOnServer

The function "RunGlobalProcedureOnServer" runs a specified VBScript procedure, as defined in the **Procedures** folder in the Project Explorer.

The procedure is run on the project runtime server, but it can be triggered by any client that calls this function.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureOnServer	Module activity	synchronous	supported	supported	supported

Syntax

```
RunGlobalProcedureOnServer(strProcedureName{ | ,optStrArgument1,…,optStrArgumentN })
```

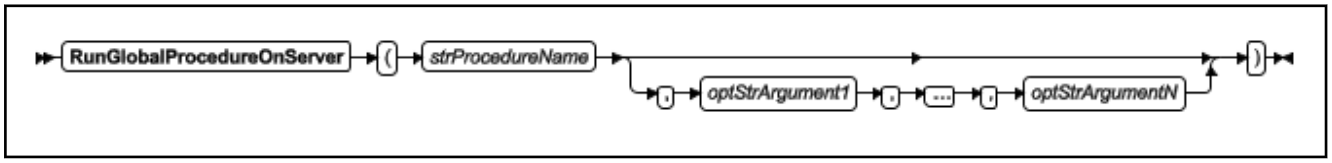


Fig. 19-32: Syntax

Inputs

strProcedureName

The name of the procedure (i.e., a VBScript function or sub-routine defined in the Procedures, page 927, folder) to be executed on the server.

optStrArgument1, …, optStrArgumentN

Values that are passed to the procedure's parameters. Arguments must be passed as strings, but the procedure will interpret them as the correct data types.

For more information, see "Examples" below.

Returned value

This function returns the value that is returned by the called procedure.

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Also, you cannot call this function in a procedure that was itself run by calling the function "RunGlobalProcedureOnServer". If you attempt to do so, then the function will return an error. This is to prevent a possible memory leak caused by nested or recursive function calls. You can still call other procedures directly, as you normally would in VBScript, or you can use the function Eval in VBScript to dynamically determine the procedure you are calling.

Examples

Given the following procedure that is defined in the **Procedures** folder…

RunGlobalProcedureOnServer - Examples

```
Function AddMe(intNumber)
  If intNumber >= 6 Then
    AddMe = 0
  Else
    AddMe = intNumber + 2
  End If
End Function
```

…the procedure is run by calling the "RunGlobalProcedureOnServer" function…

```
RunGlobalProcedureOnServer("AddMe","2")
```

…and it returns a value of 4.

19.14.26 RunGlobalProcedureOnTrigger

This function directly executes a global procedure if the value or quality of a specified tag changes.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureOnTrigger	Module activity	synchronous	supported	supported	supported

Syntax

```
RunGlobalProcedureOnTrigger("strTagName",strProcedureOnTrigger)
```

Inputs

strTagName

Name of a project tag.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

strProcedureOnTrigger

The name of the procedure (i.e., a VBScript function or sub-routine defined in the Procedures, page 927 folder) to run if the value of the specified tag changes.

Returned value

This function returns the following possible values:

Value	Description
0	Error
1	Successful

Notes

If this function is called, it is continuously executed by the project runtime client (i.e., the Viewer) until either it or the project runtime server is stopped. That means every time the value or quality of the specified tag changes, the procedure is run. Also, the function can be called more than once, so that the same procedure can be run by different triggers. The project runtime client manages the execution of all instances of the function.



The procedure is executed on the client. To execute a procedure on the server, use the function RunGlobalProcedureOnServer, page 711.

The value or quality of the specified tag is passed to the procedure as an argument, so the procedure should be written to receive it.

For example:

```
Sub MyProcedure(strTrigger)
   .
   .
   .
End Sub
```

In practice, this means either…

```
strTrigger = $tagname->Value
```

...or...

```
strTrigger = $tagname->Quality
```

…depending on which one changed. You can then use the value or quality in your procedure.

Please note that you do not actually have to use the argument, only that you should write the procedure to receive it.

No other arguments can be passed to the procedure.

To ensure compatibility with previous versions of Rexroth WinStudio, passing the argument is disabled by default in existing projects and enabled by de-

fault in new projects. To change this for your project, open your project file (`projectname.APP`) in a text editor and then edit the following property:

```
[Options]
EnableTagNameOnRunGlobalProcedureOnTag=<0 (disabled) / 1 (enabled)>
```

Examples When the value or quality of the tag `MyInteger` changes, run the procedure `MyProcedure`:

```
RunGlobalProcedureOnTrigger("MyInteger","MyProcedure")
```

The equivalent of `MyInteger->Value` or `MyInteger->Quality`, depending on which one changed, is passed to the procedure as an argument.

19.14.27 RunGlobalProcedureOnTrue

This function directly executes a global procedure if the value of a specified tag/expression becomes TRUE.

Function	Group	Execution	Windows	Embedded	Thin Client
RunGlobalProcedureOnTrue	Module activity	synchronous	supported	supported	supported

Syntax

```
RunGlobalProcedureOnTrue("strCondition",strProcedureOnTrue)
```

Inputs

strCondition

A project tag or expression to trigger the function.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

strProcedureOnTrue

The name of the procedure (i.e., a VBScript function or sub-routine defined in the Procedures, page 927 folder) to run if the value if "strCondition" becomes TRUE (not zero).

Returned value This function returns the following possible values:

Value	Description
0	Error
1	Successful

Notes Once this function is called, it is continuously executed by the project runtime client (i.e., the Viewer) until either it or the project runtime server is stopped. That means every time the value of the condition becomes TRUE, the procedure is run. However, the procedure is run only once when the value of the condition becomes TRUE; it is not repeatedly run while the value of the condition is TRUE. For the procedure to run again, the value of the condition must become FALSE and then TRUE again.

Also, the function can be called more than once, so that the same procedure can be run by different triggers. The project runtime client manages the execution of all instances of the function.

The procedure is run on the client where this function was called. To run a procedure on the project server, use the function RunGlobalProcedureOnServer, page 711.

The value of "strCondition" is passed to the procedure as an argument, so the procedure should be written to receive it.

For example:

```
Sub UsingOnTrue(strCondition)
  .
  .
  .
End Sub
```

Please note that you do not actually have to use the argument in your procedure, only that you should write the procedure to receive it.

No other arguments can be passed to the procedure.

To ensure compatibility with previous versions of Rexroth WinStudio, passing the argument is disabled by default in existing projects and enabled by default in new projects. To change this for your project, open your project file (`projectname.APP`) in a text editor and then edit the following property:

```
[Options]
EnableTagNameOnRunGlobalProcedureOnTag=<0 (disabled) / 1 (enabled)>
```

Examples

When the value of "TagOnTrue" becomes TRUE, run the procedure "UsingOnTrue":

```
RunGlobalProcedureOnTrue("TagOnTrue","UsingOnTrue")
```

19.14.28 RunVBScript

Executes a statement in VBScript language.

Function	Group	Execution	Windows	Embedded	Thin Client
RunVBScript	Module activity	synchronous	supported	supported	supported

Syntax

```
RunVBScript ( strScript, "optTagReturnError" )
```

Inputs

strScript

Script statement that is to be executed by the function.

optTagReturnError

Name of the tag that is to receive the error (if any) generated by the statement (e.g., "Division by zero"). String type tags have to be put in quotation marks.

Returned value

Value	Description
0	Error
1	Successful

Examples

Tag name	Expression
TagResult	`RunVBScript("MsgBox(Time)")` `//` Executes the MsgBox function page 956 from VBScript and displays the current time.
	`RunVBScript(TagStatement)` // Executes the statement configured in the value of the string tag `TagStatement`.
	`RunVBScript("$TagC=$TagA/$TagB", "TagError")` // Writes in `TagC` the result of `TagA` divided by `TagB`. The error generated by the operation (if any) is written to the string tag `TagError` (e.g., "Division by zero").



This function is useful to execute VBScript statements in tasks that support the internal language only (e.g., scheduler groups). VBScript functions created in the global procedures can also be called.



The runtime station has to support the VBScript statements configured in this function in order to execute them.

19.14.29 SecureViewerReload

"SecureViewerReload" is an integrated scripting function that closes the "Secure Viewer" program and reloads it with a new configuration file.

Function	Group	Execution	Windows	Embedded	Thin Client
SecureViewerReload	Module activity	synchronous	not supported	not supported	Secure Viewer only

Syntax

```
SecureViewerReload(strFileName)
```

Inputs

strFileName

The file path of an INI file (`*.ini`) that describes the new configuration. (If the file is located in the same folder as `Viewer.exe`, only the file name is needed.) The file is to have the same structure and contain the same settings as the default configuration file (`Viewer.ini`).

This parameter can be a String tag or a constant text in quotation marks.

Returned value

This function does not return any value.

Examples

Reload the Secure Viewer with the configuration file that is specified by the tag `configFile1`:

```
SecureViewerReload(configFile1)
```

Reload the Secure Viewer with the configuration file that is located at the specified file path:

```
SecureViewerReload("C:\Program Files\Secure Viewer\Bin\Config1.ini")
```

19.14.30 SendKeyObject

Sends keyboard codes to the objects in the currently displayed WinStudio screen. This function can be used to trigger actions on dynamic screen objects.

Function	Group	Execution	Windows	Embedded	Thin Client
SendKeyObject	Module activity	synchronous	supported	not supported	not supported

Syntax

```
SendKeyObject( numEvent, strMainKey, numShift, numCtrl, numAlt, strTargetScreen, optNumID)
```

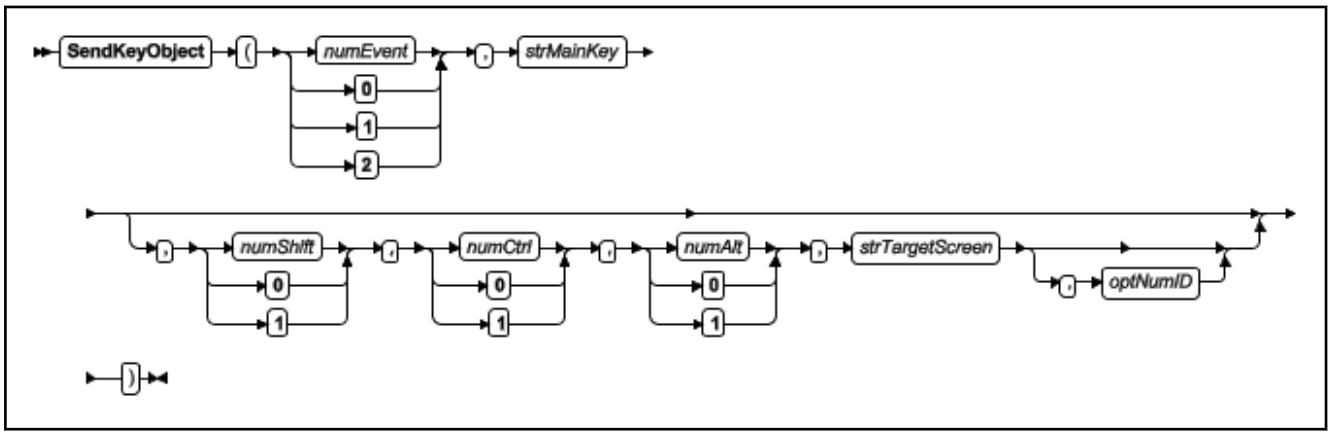


Fig. 19-33: Syntax

Inputs

numEvent

The event code for "On Down", "On While" or "On Up" of the command animation, page 246 is as follows:

0	On Down
1	On While
2	On Up

If the "On While" event is specified, the "On While" script on the Command animation is executed just once for each time this function is executed. It is not continuously executed as if the key is pressed and held down, because this function does not have a parameter for specifying duration.

If you want to cause that sort of behavior, you can include this function in an appropriately configured FOR loop.

strMainKey

Keyboard codes to be sent to WinStudio screens. The following values are permissible:

- "A" … "Z"
- "+"
- "-"
- "/"
- "*"
- "HOME"
- "END"
- "INSERT"
- "DELETE"
- "DOWN", down arrow (↓)
- "UP", up arrow (↑)
- "LEFT", left arrow (←)
- "RIGHT", right arrow (→)
- "PAGEUP"
- "PAGEDOWN"
- "SPACE"
- "RETURN"
- "BACKSPACE"
- "ESCAPE"
- "F1" … "F20"

The key must be enclosed in quotes, as shown.

numShift

A numeric value that indicates whether to include <Shift> with the specified key (e.g., <Shift+R>). The following values are accepted: 0 is no, 1 is yes.

This parameter is optional; if no value is specified, the default value is 0.

numCtrl

A numeric value that indicates whether to include <Ctrl> with the specified key (e.g., <Ctrl+R>). The following values are accepted: 0 is no, 1 is yes.

This parameter is optional; if no value is specified, the default value is 0.

numAlt

A numeric value that indicates whether to include <Alt> with the specified key (e.g., <Alt+R>). The following values are accepted: 0 is no, 1 is yes.

This parameter is optional; if no value is specified, the default value is 0.

strTargetScreen

The name of the screen that will receive the key event code.

This parameter is optional; if no value is specified, the currently active screen is used.

optNumID

The specific instance number of the screen. (The ID is assigned when the screen is opened with the Open(), page 666 function.)

This is an optional command parameter. The default value is 0.

Returned value

This function does not return any values.

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Also, the parameters "numShift", "numCtrl", "numAlt" and "strTargetScreen" are all optional, but if you configure one of them, you must configure the others as well.

Examples

Send <R> to the currently active screen:

```
SendKeyObject(0,"R")
```

Send <Ctrl+Shift+R> to the screen named "main" with ID 10:

```
SendKeyObject(0,"R",1,1,0,"main",10)
```

19.14.31 SetAppPath

Sets the new file path for the project folder. After this function is executed, "WinStudio" searches for all project files (i.e.,screens, alarms, trends, database, events) in this folder.

Function	Group	Execution	Windows	Embedded	Thin Client
SetAppPath	Module activity	synchronous	supported	not supported	executed on Server

Syntax

```
SetAppPath(strPath)
```

Inputs

strPath

String tag or constant text in quotation marks with the file path

Returned value

0	Failed to set path
1	Succeeded in setting path

Examples

Tag name	Expression
	SetAppPath("C:\Studio\")

☞ If the computer is on a network, the `//IP address` or `host name/Path` syntax can be used to define a location on another network drive.

19.14.32 SetViewerInFocus

"SetViewerInFocus" is an internal scripting function that moves the viewer in front of all other open windows and maximizes it to fill the screen.

Function	Group	Execution	Windows	Embedded	Thin Client
SetViewerInFocus	Module activity	synchronous	supported	not supported	supported

Syntax

```
SetViewerInFocus()
```

There are no parameters.

Returned value This function does not return any value.

Notes With the introduction of Windows XP, system security functions prevent program windows from moving themselves in front of other windows without user input. If this function is called, the Windows taskbar indicates the activated window by blinking. (Anti-virus software may flag this mechanism as suspicious behavior.) The program window is not moved to the front before the program is selected by the user.

To work around this, "SetViewerInFocus" has to be called at least once in the Startup Script, page 402 of the project. Allow twenty seconds more for your project to finish starting up, and then after that, any additional calls of this function should work normally.

19.14.33 SetViewerPos

Sets the height, width and position of the project viewer or Thin Client.

Function	Group	Execution	Windows	Embedded	Thin Client
SetViewerPos	Module activity	synchronous	supported	not supported	supported

Syntax

```
SetViewerPos( numLeft, numTop, optNumWidth, optNumHeight )
```

Inputs

numLeft

A numeric tag that specifies the left-side position of the Viewer in pixels.

numTop

A numeric tag that specifies the top-side position of the Viewer in pixels.

optNumWidth

"Optional" numeric tag specifying the Viewer width in pixels.

optNumHeight

"Optional" numeric tag containing the Viewer height in pixels.

Returned value

0	Error
1	Successful

Examples

Tag name	Expression
Tag	SetViewerPos(50, 50, 640, 480)

The Viewer is opened with its default size set in the project settings if the optional parameters (**numWidth** and **numHeight**) are not specified.

19.14.34 ShutDown

Closes all active modules of the runtime environment.

Function	Group	Execution	Windows	Embedded	Thin Client
ShutDown	Module activity	asynchronous	supported	supported	supported (see "Notes" below)

Syntax

```
ShutDown()
```

This function does not have any parameters.

Returned value

This function does not return any values.

Notes

This function only stops the project runtime server. It does not close the development environment, page 39 on the server, if it happens to be open.

If this function is called from a screen object or script on a client, it only stops the Viewer module on that client. On Mobile Access, it returns the user to the logon screen.

Also, when this function is used in project screens in Mobile Access, it cannot be called from the sub-routine "Screen_OnClose" in the Screen Script, page 932.

19.14.35 StartTask

Starts a runtime module that is not currently running.

Function	Group	Execution	Windows	Embedded	Thin Client
StartTask	Module activity	asynchronous	supported	supported	executed on the server

Syntax

```
StartTask(strTask)
```

Inputs

strTask

The name of the task to be started (from the following list):

Value	Description
BGTask	Background tasks
Viewer	Viewer
DBSpy	Database Spy
LogWin	LogWin
Driver	Driver

Value	Description
UniDDECl	DDE client
UniNDDE	DDE server
UniODBC	ODBC
TCPServer	TCP/IP server
TCPClient	TCP/IP client
OPCClient	OPC client

Returned value

This function does not return any values.

Examples

Tag name	Expression
	StartTask("Viewer")

The `StartTask("Driver")` function starts all drivers configured in the project. To start a specific driver, the WinExec(), page 724 function has to be used.

19.14.36 ViewerPostMessage

Sends a Windows system message to the specified WinStudio screen.

Function	Group	Execution	Windows	Embedded	Thin Client
ViewerPostMessage	Module activity	asynchronous	supported	supported	supported

Syntax

```
ViewerPostMessage( strScrTitle, numMessage, numwParam, numlParam, optNumID )
```

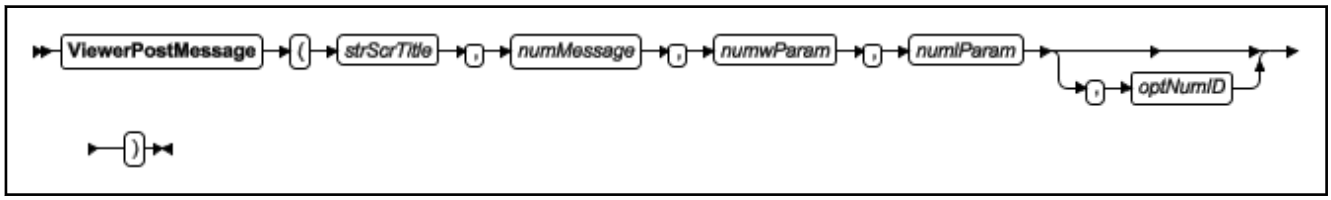


Fig. 19-34: Syntax

Inputs

strScrTitle

The name of the screen to which the message is to be sent.

numMessage

The number of the Windows system message to be sent.

numwParam

A numeric value or tag with a message-specific value, which is passed to the **wParam** parameter of the Windows system message.

numlParam

A numeric value or tag with a message-specific value, which is passed to the **lParam** parameter of the Windows system message.

optNumID

The specific instance number of the WinStudio screen. (The ID is assigned when the screen is opened with the Open(), page 666, function.)

This is an optional command parameter. The default ID is 0.

Returned value This function does not return any values.

Notes

This function emulates the PostMessage function in Microsoft Windows. For further information including a complete list of available Windows system messages, please refer to the Microsoft Developers Network.

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples *Send message 16 to the screen named "main" with ID 10:*

```
ViewerPostMessage("main",16,3,1,10)
```

19.14.37 Wait

Pauses the execution of the runtime system for a specified number of milliseconds.

Function	Group	Execution	Windows	Embedded	Thin Client
Wait	Module activity	synchronous	supported	supported	supported

Syntax

```
Wait(numMillisec)
```

Inputs

numMillisec

A numeric variable or a constant value; the number of milliseconds for the waiting time.

Returned value This function does not return any values.

This function is normally used only in math worksheets, page 400, however, it can also be used in any VBScript interface, page 925.

NOTICE

It is dangerous to use this function anywhere in the project as all incoming information from shared databases and connected devices is ignored as long as the runtime system is paused. Applications with this function have to be carefully tested.

Examples

Tag name	Expression
	Wait(1000)

19.14.38 WinExec

"WinExec" is an internal scripting function that executes a Windows command as if it was entered into the command prompt.

Function	Group	Execution	Windows	Embedded	Thin Client
WinExec	Module activity	asynchronous	supported	supported (see note)	supported

Syntax

```
WinExec(strCommand{ | ,{ optNumState | 0 | 1 | 2 | 3 | 4 | 7 }{ | ,{ optNumSync | 0 | 1 },
                "optTagReturnOrHandle" } })
```

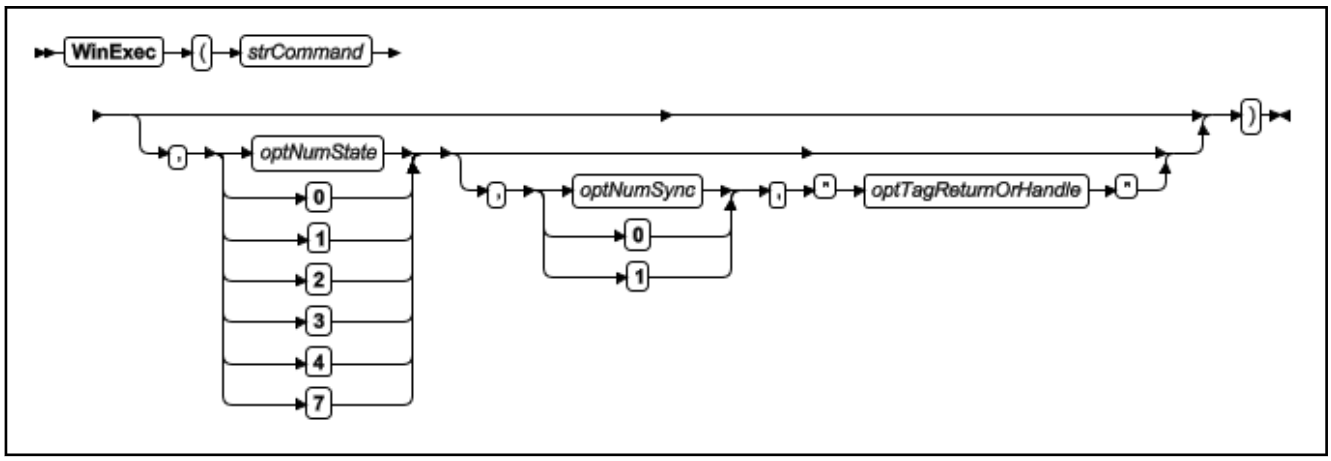


Fig. 19-35: Syntax

Inputs

strCommand

String tag or constant text in quotation marks with the command for command execution

optNumState

The initial state of the program (if any) that is run by the command:

Value	Description
0	Hides the program and transfers the control to another program
1	Activates and displays the program
2	Activates the program and displays it as an icon
3	Activates the program and maximizes it
4	Shows the program at the size it was last used The program is still active
7	Shows the program as an icon The program is still active

This parameter is optional. If no value is specified, the default value is "1".

This parameter is not supported on Windows Embedded target systems. In this case, the transferred parameter value is ignored and the default value is used.

optNumSync

Parameter specifying whether the command is executed synchronously or asynchronously:

Value	Description
0	Asynchronous execution; the function immediately generates a value to be returned
1	Synchronous execution; the function creates a value to be returned as soon as the command has finished executing

This parameter is optional. If no value is specified, the default value is "0".

To verify that an asynchronously executed command has finished, use the "optTagReturnOrHandle" parameter as follows and the WinExecIsRunning, page 726 function.

optTagReturnOrHandle

The name of a project tag that stores the returned value of the command:

- If the command is executed asynchronously, the tag receives a handle that can be used with the WinExecIsRunning, page 726 function to determine whether the command is still running.
- If the command is executed synchronously, the tag receives the output code of the command. (This applies independent of the returned value of the function.)

This is an optional parameter, however, there is no default value.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

This function returns the following possible values:

Value	Description
0	Command was not executed successfully
1	Command was executed successfully

Please note that the returned value indicates only whether the execution of the command has been "started" successfully, particularly if it is executed asynchronously. It does not indicate when or how the command finished its execution.

Examples

Starting Notepad and jumping immediately to the next row of the script:

```
WinExec( "C:\Windows\System32\notepad.exe", 4 )
```

Starting MS Paint and jumping immediately to the next row of the script:

```
WinExec( "C:\Windows\System32\mspaint.exe" )
```

Calling a batch file, executing it in hidden mode, waiting until it is finished and storing the output code in the tag "return":

```
( "CMD /C call C:\Temp\MyBatch.bat", 0, 1, "return" )
```

Calling a VBScript file, executing it in hidden mode, immediately continuing and storing the handle in the tag "handle":

```
WinExec( "CMD /C call C:\Temp\MyScript.vbs", 0, 0, "handle" )
```

Calling VBScript files is not supported on Windows Embedded target systems.

19.14.39 WinExecIsRunning

Checks whether a program started by the "WinExec" function is still running.

Function	Group	Execution	Windows	Embedded	Thin Client
WinExecIsRunning	Module activity	synchronous	supported	supported	supported

Syntax

```
WinExecIsrunning( numHandle, "optTagReturn" )
```

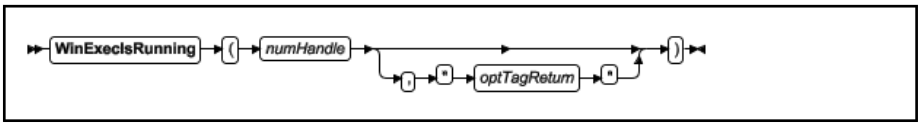


Fig. 19-36: Syntax

Inputs

numHandle

Handle number returned to the "optStrReturnorHandle" tag on calling the function WinExec, page 724.

optTagReturn

Tag with return value of the program that was called by means of the **WinExec** function.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

This function returns the following possible values:

1	Program is still running.
0	Execution has been completed successfully (i.e., program has stopped).
-1	Invalid number of parameters.
-2	Invalid handle number. Check the value of "numHandle".
-3	Execution has been completed, but the return tag is invalid. Check the value of "optTagReturn".

Examples

Tag name	Expression
Tag	WinExecIsRunning(numHandle)
Tag	WinExecIsRunning(numHandle, "return")

19.15 File functions

19.15.1 File functions - General information

These functions are used to read from, write to, print, move, and delete external files.

Function	Description
DeleteOlderFiles, page 728	Deletes files that are older than a date matching the configured mask from the configured path.
DirCreate, page 728	Creates the specified directory.
DirDelete, page 729	Deletes the specified directory.
DirLength, page 730	Returns the size of a specific directory.
DirRename, page 730	Renames directories.
FileCopy, page 731	Copies the file(s) configured in the "strSourceFile" parameter to the path/file configured in the "strTargetFile" parameter.
FileDelete, page 732	Deletes the specified file.
FileLength, page 732	Gets the size of a file.
FileReadFields, page 733	Reads values in the fields of a CSV file and writes these values to multiple project tags or array elements.
FileReadMessage, page 734	Reads a message (e.g. a character string) from a text file and writes this message to a project tag.
FileRename, page 736	Is a built-in scripting function that renames a specified file.
FileWrite, page 736	Is a built-in function writing a specified character string to a specified text file. If the file does not exist, it is created when the function is executed.
FileWriteFields, page 737	Reads values available in multiple project tags or array elements and writes these values to the fields of a CSV file.
FileWriteMessage, page 739	Reads a message (e.g. a character string) from a project tag and writes this message to a text file.
FindFile, page 740	Is a built-in scripting function that searches for files that match a given search string.
FindPath, page 741	Checks whether a directory exists.
GetFileAttributes, page 742	Reads the attributes of a specified file.
GetFileTime, page 742	Reads the time and date the file was last modified.
GetHstInfo, page 743	Returns the start time, end time, and duration of the specified history (*.HST) file.
GetLine, page 744	Gets a specific line or search string from a text file and stores the line in a String tag.
HST2TXT, page 746	Is a built-in function exporting historic trend data from the own binary format of the Rexroth WinStudio to a text file.
HST2TXTIsRunning, page 748	Returns the status of the HST2TXT function.
ImportXML, page 749	Used to import Studio XML files into your project at runtime.

Function	Description
LookupContains, page 751	Checks whether an external file contains the specified keyword in its key column.
LookupGet, page 751	Gets a value from an external file by cross-referencing from a specified keyword.
LookupLoad, page 752	Loads an external file — typically, a delimited text file — that can be used to look up table values. One column of the file is designated as the keywords column, and another column is designated as the values column.
PDFCreate, page 752	Creates a PDF file from the specified source file.
Print, page 753	Prints a text file.
RDFileN, page 754	Launches a file browser window to select a file.
WebGetFile, page 755	Downloads a file from a specified address and then saves it locally.

19.15.2 DeleteOlderFiles

Deletes files from a directory that are older than a specific date and correspond to the configured file mask.

Function	Group	Execution	Windows	Embedded	Thin Client
DeleteOlderFiles	Module activity	synchronous	supported	supported	supported

Syntax

```
DeleteOlderFiles( strPath, strMask, strDate )
```

Inputs

strPath

The folder path to the files that are to be deleted.

strMask

The mask of the files to be deleted.

strDate

The cut-off date. All files older than this date are deleted.



The file format (e.g. **JMT** or **TMJ**) with the respective separator ("/",":","." etc.) is specified in the project settings.

Returned value

Returns the number of files deleted.

Examples

Tag name	Expression
Tag	DeleteOlderFiles("C:\Studio\Project\HST\", "*.hst", "04/12/2002")

19.15.3 DirCreate

Creates the specified directory.

Function	Group	Execution	Windows	Embedded	Thin Client
DirCreate	File	synchronous	supported	supported	executed on the server

Syntax

```
DirCreate( strDirectory, optBooFullPath )
```

Inputs

strDirectory

Path and name of the directory to be created.

optBooEmptyOnly

Optional parameter. If this parameter is not specified or if it is set to 0, the directory is only created if all parent directories exist. If this parameter has a value different from 0, the full path specified in the **strDirectory** parameter is created.

Returned value

Value	Description
-1	Invalid parameters
0	Failed to create the directory (e.g., drive does not exist)
1	Directory created successfully

Examples

Tag name	Expression
Tag	DirCreate("C:\Studio\Temp") // The Temp folder is created in the C:\Studio path (only if the C:\Studio path already exists).
Tag	DirCreate("C:\Studio\Temp",1) // The C:\Studio\Temp full path is created.

19.15.4 DirDelete

Deletes the specified directory.

Function	Group	Execution	Windows	Embedded	Thin Client
DirDelete	File	asynchronous	supported	supported	supported

Syntax

```
DirDelete( strDirectory, optBooEmptyOnly )
```

Inputs

strDirectory

Path and name of the folder to be deleted.



This parameter supports placeholders ("*" and "?").

optBooEmptyOnly

Optional function parameter. If this parameter has a value of 1, the directory is deleted only if it is empty. By default – i.e., if the parameter is not specified or has a value of 0 – the directory is deleted whether it is empty or not.

Returned value

Value	Description
−2	Attempted to delete a non-empty directory when this action is not allowed (i.e., **optBooEmptyOnly** does not equal 0)
−1	Invalid parameters

Value	Description
0	Failed to delete the directory (i.e., directory does not exist)
1	Directory deleted successfully

Examples

Tag name	Expression
Tag	DirDelete("C:\Studio\Temp") // The Temp folder from C:\Studio is deleted.
Tag	DirDelete("C:\Studio\Temp", 1) // The Temp folder from C:\Studio is deleted only if it is empty.

19.15.5 DirLength

Returns the size of a directory.

Function	Group	Execution	Windows	Embedded	Thin Client
DirLength	File	synchronous	supported	supported	executed on the server

Syntax

```
DirLength(strPath)
```

Inputs

strPath

Path of the directory to be checked.

Returned value

−2	Directory does not exist
−1	Invalid parameters
n	Size (in bytes) of the files and subfolders in the directory

NOTICE

This function is executed synchronously. This means, the runtime system pauses while it waits for the function output.

If the specified directory contains large data amounts, this may lead to an interruption in processing of several seconds.

Examples

Tag name	Expression
Tag	DirLength("C:\Studio") // Returns the size (in bytes) of all files and subfolders from C:\Studio.

19.15.6 DirRename

Renames a directory.

Function	Group	Execution	Windows	Embedded	Thin Client
DirRename	File	asynchronous	supported	supported	executed on the server

Syntax

```
DirRename(strPath,strDirectoryFrom,strDirectoryTo)
```

Inputs

strPath

Path of the directory to be renamed

This function supports placeholders ("*" and "?").

strDirectoryFrom

The original name of the directory that is to be renamed.

strDirectoryTo

The new name of the directory.

Returned value

Value	Description
−1	Invalid parameters
0	Failed to rename the directory (e.g., **strDirectoryFrom** does not exist)
1	Directory renamed successfully.

Examples

Tag name	Expression
Tag	DirRename("C:\Studio\","Temp", "New") // C:\Studio\Temp is renamed to C:\Studio\New.

19.15.7 FileCopy

Copies files to a higher directory

Function	Group	Execution	Windows	Embedded	Thin Client
FileCopy	File	Synchronous	Supported	Supported	Supported

Syntax

```
FileCopy(strSourceFile,strTargetFile{ | ,optNumTimeOut})
```

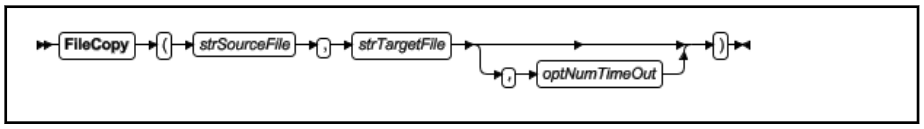


Fig. 19-37: Syntax

Inputs

strSourceFile

Path and name of the file(s) to be copied.

This function supports placeholders ("*" and "?").

strTargetFile

The file path the file(s) are to be copied to.

optNumTimeOut

Numeric tag containing an integer to set the timeout period for the operation in ms.

If this parameter is used, the function returns the value -1 after the specified timeout period.

The copying procedure is not cancelled by this. Instead, it creates an internal process to finish the copying procedure.

Return value

Value	Description
−1	Timeout period expired.
0	Failed to copy file(s).
1	File(s) copied successfully.

NOTICE **This function is executed synchronously.**

This means, the project pauses while it waits for the function to complete. Large amounts of data and slow network connections may lead to longer program interruptions.

Examples

Tag name	Expression
Tag	FileCopy("C:\Studio\Project\HST*.hst", "C:\Temp\Hst\", 1000)
Tag	FileCopy("C:\Studio\Project opert.txt", "C:\Temp\Tuesday_Report.txt", 500)

19.15.8 FileDelete

Deletes the specified file.

Function	Group	Execution	Windows	Embedded	Thin Client
FileDelete	File	synchronous	supported	supported	supported

Syntax

```
FileDelete(strFile)
```

Inputs

strFile

Path and name of the file to be deleted.

Returned value

Value	Description
0	Failed to delete file
Real	Returns the size of the deleted file

Examples

Tag name	Expression
Tag	FileDelete("C:\Studio\Readme.txt")

19.15.9 FileLength

Determines the size of a file.

Function	Group	Execution	Windows	Embedded	Thin Client
FileLength	File	synchronous	supported	supported	supported

Syntax

```
FileLength(strFile)
```

Inputs

strFile

Path and name of the file.

Returned value

Returns the size of the specified file in bytes.

Examples

Tag name	Expression
Tag	FileLength("C:\Readme.txt")

19.15.10 FileReadFields

The function FileReadFields reads values contained in the fields of a CSV file, and then it writes those values to a series of project tags or array elements.

Function	Group	Execution	Windows	Embedded	Thin Client
FileReadFields	File	synchronous	supported	supported	not supported

Syntax

```
FileReadFields(strFilename,numOffset,strStartTagName,numNumberOfTags)
```

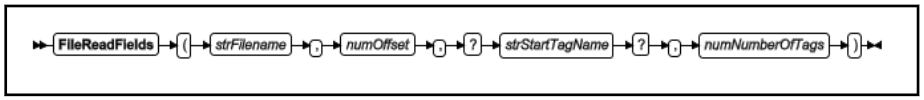


Fig. 19-38: Syntax

Inputs

strFilename

The file path and/or name of the CSV file. If the file is located inside your project folder, you can specify either just the file name or the file path relative to that folder. If the file is located outside your project folder, you must specify the absolute file path.

numOffset

The number of bytes to skip in the CSV file before reading values. To read from the start of the file, numOffset should be `0`.

You can use this parameter to start reading from any position in the file, as long as you know how many bytes to skip. In most cases, you will simply take the value returned by the previous execution of this function (see "Returned value" below) and use it to resume reading where you previously stopped.

However, if you already know the structure of the file and where you want to start in it, you can do that. For example, if you know that each line of the file is exactly 100 bytes and you want to read from the start of the fifth line, "numOffset" should be `400`.

☞ The number of bytes per character in a file depends on the text encoding (i.e., ANSI, UTF-8, UTF-16, or other), the byte order, and the language or character set.

strStartTagName

The name of the first project tag or array element in the series that will receive the values read from the CSV file. Project tags should be sequentially numbered with a numerical suffix, and the series is determined by incrementing that suffix (e.g., `MyTag1`, `MyTag2`, `MyTag3`, and so on). Array elements are handled similarly: the series is determined by incrementing the array index (e.g., `MyArray[1]`, `MyArray[2]`, `MyArray[3]`, and so on). You do not need to begin the numbering with 1.

☞ If the tag name is not enclosed in quotes, the function will try to use the value of specified tag.

numNumberOfTags

The number of project tags or array elements in the series that will receive values read from the CSV file.

For example, if "strStartTagName" is "MyTag4" and "numNumberOfTags" is 5, five values will be read from the file and then written to the tags `MyTag4`, `MyTag5`, `MyTag6`, `MyTag7`, and `MyTag8`.

Returned value If this function is successfully executed, it returns the position of the last byte read from the CSV file (including 0 if no bytes were read), which can be used in turn as the offset for the next batch of fields to be read.

If this function fails, it returns a negative value.

Notes "CSV" is an abbreviation of "comma-separated values", and in most cases, a CSV file is simply a plain text file that uses commas (,) to delimit its data fields. Only comma delimiters - as opposed to tabs, spaces, pipes, or other characters - are supported in CSV files. Each field in the file contains exactly one value, even if the value is empty, and each project tag or array element will receive exactly one value read from the file.

When this function is executed, it will read at most one line of data. If a line feed is found before reading the specified number of values, execution of the function will be aborted without reading any values.

Examples *FileReadFields - Examples*

```
FileReadFields("C:\FieldFiles\FieldFile01.csv",400,"IntValueTag003",5)
FileReadFields("FieldFile02.csv",0,"IntValueTag[0]",10)
FileReadFields("FieldFile03.csv",0,"IntValueTag[IndexTag]",7)
```

19.15.11 FileReadMessage

The function FileReadMessage reads a message (i.e., a string of characters) from a text file, and then it writes that message to a project tag.

Function	Group	Execution	Windows	Embedded	Thin Client
FileReadMessage	File	synchronous	supported	supported	not supported

Syntax

```
FileReadMessage(strFilename,numOffset,strMessageTag,numCharsToRead)
```

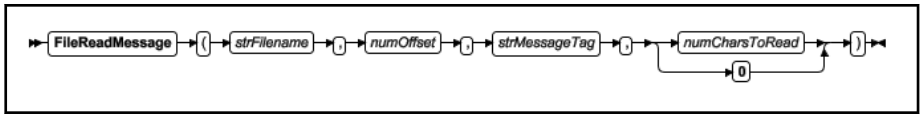


Fig. 19-39: Syntax

Inputs

strFilename

The file path and/or name of the text file. If the file is located inside your project folder, you can specify either just the file name or the file path relative to that folder. If the file is located outside your project folder, you must specify the absolute file path. The file path and/or name of the CSV file. If the file is located inside your project folder, you can specify either just the file name or the file path relative to that folder. If the file is located outside your project folder, you must specify the absolute file path.

numOffset

The number of bytes to skip in the CSV file before reading values. To read from the start of the file, "numOffset" should be `0`.

You can use this parameter to start reading from any position in the file, as long as you know how many bytes to skip. In most cases, you will simply take the value returned by the previous execution of this function (see "Returned value" below) and use it to resume reading where you previously stopped.

However, if you already know the structure of the file and where you want to start in it, you can do that. For example, if you know that each line of the file is exactly 100 bytes and you want to read from the start of the fifth line, "numOffset" should be `400`.

☞ The number of bytes per character in a file depends on the text encoding (i.e., ANSI, UTF-8, UTF-16, or other), the byte order, and the language or character set.

strMessageTag

The name of the project tag (String type) that will receive the message read from the text file.

☞ If the tag name is not enclosed in quotes, the function will try to use the value of specified tag.

numCharsToRead

The number of characters to read from the text file, starting from the position specified by "numOffset". If this value is 0, all characters up to the next line feed (LF) will be read.

Returned value

If this function is successfully executed, it returns the position of the last byte read from the text file (including 0 if no bytes were read), which be used in turn as the offset for the next message to be read.

If this function fails, it returns a negative value.

Notes

This function can be used to read from any plain text file, as long as the correct file extension is specified in "strFilename".

Examples

FileReadMessage - Examples

```
FileReadMessage("C:\Data\Messages01.txt",0,"MsgTag",0)
FileReadMessage("Messages02.txt",0,"MsgTag",140)
```

19.15.12 FileRename

Renames a file.

Function	Group	Execution	Windows	Embedded	Thin Client
FileRename	File	synchronous	supported	supported	supported

Syntax

```
FileRename(strOldName,strNewName)
```

Inputs

strOldName

Path and name of the file to be renamed.

strNewName

Path and new name of the file.

Returned value This function does not return any value.

Examples

FileRename - Examples

```
FileRename( "C:\readme.txt", "C:\readthis.txt" )
```

19.15.13 FileWrite

Writes a string to a specified ASCII or Unicode file. If the file does not exist, it is created by the function.

Function	Group	Execution	Windows	Embedded	Thin Client
FileWrite	File	synchronous	supported	supported	supported

Syntax

```
FileWrite(strFileName,strWriteText{ | ,optNumAppend })
```

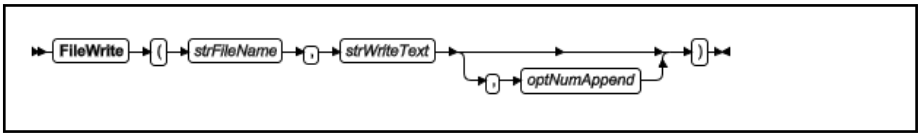


Fig. 19-40: Syntax

Inputs

strFileName

Path and name of the file to be written with the function (String tag or string in quotation marks).

By default, the file is written to the project folder (i.e., the folder that contains the file **project_name.APP**), on the computer that host the project runtime server.

To write to another folder, enter the complete file path.

strWriteText

A string in quotation marks or a String tag containing the text to be written to the file.

optNumAppend

A numeric value or the name of the Integer tag with the ID for file handling (see table):

Value	Description
0	Creates a new ASCII file with the given file name. If the file already exists, it is overwritten.
1	Appends to an existing ASCII file with the given file name. If the file does not exist, a new file is created.
2	Creates a new Unicode file (UTF-16LE) with the specified file name. If the file already exists, it is overwritten.
3	Appends to an existing Unicode file with the given file name. If the file does not exist, a new file is created.

This parameter is optional. If any value other than 0 to 3 is specified or if the parameter is not used, the default value 0 is used.

Returned value

0	Successful execution
−1	Invalid parameter(s)
−2	Failed to open file Disk is write-protected or file name is invalid

Notes

If this function is called in a project screen on a thin client, it will still write to the specified file in the project folder on the computer that hosts the project runtime server. You cannot use this function to write to a file on the thin client.

Examples

Create a new ASCII file in the project folder:

```
FileWrite("est.txt","This is a test.")
```

Append to an existing ASCII file in the project folder:

```
FileWrite("est.txt","This is a test.",1)
```

Append to an existing Unicode file in the Documents folder:

```
FileWrite("C:\Users\MyUser\Documents\est.txt","This is a test.",3)
```

19.15.14 FileWriteFields

The function "FileWriteFields" reads values contained in a series of project tags or array elements, and then it writes those values to the fields of a CSV file.

Function	Group	Execution	Windows	Embedded	Thin Client
FileWriteFields	File	synchronous	supported	supported	not supported

Syntax

```
FileWriteFields(strFilename,numOffset,strStartTagName,numNumberOfTags)
```

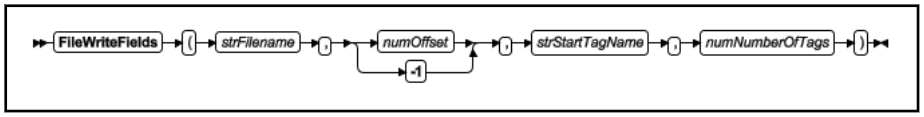


Fig. 19-41: Syntax

Inputs

strFilename

The file path and/or name of the CSV file to which the values will be written. If the file is located inside your project folder, you can specify either just the file name or the file path relative to that folder. If the file is located outside your project folder, you must specify the absolute file path.

numOffset

The number of bytes to skip in the CSV file before writing values. To write to the start of the file, numOffset should be `0`.

You can use this parameter to start writing to any position in the file, as long as you know how many bytes to skip. In most cases, you will simply take the value returned by the previous execution of this function (see "Returned value" below) and use it to resume writing where you previously stopped.

However, if you already know the structure of the file and where you want to start in it, you can do that. For example, if you know that each line of the file is exactly 100 bytes and you want to write to the start of the fifth line, numOffset should be `400`.

You can also specify a value of -1, which will automatically append the values to the end of the file.



The number of bytes per character in a file depends on the text encoding (i.e., ANSI, UTF-8, UTF-16, or other), the byte order, and the language or character set.

strStartTagName

The name of the first project tag or array element in the series from which the function will read the values to be written to the CSV file. Project tags should be sequentially numbered with a numerical suffix, and the series is determined by incrementing that suffix (e.g., `MyTag1, MyTag2, MyTag3`, and so on). Array elements are handled similarly: the series is determined by incrementing the array index (e.g., `MyArray[1], MyArray[2], MyArray[3]`, and so on). You do not need to begin the numbering with 1.



If the tag name is not enclosed in quotes, the function will try to use the value of specified tag.

numNumberOfTags

The number of project tags or array elements in the series to read. For example, if "strStartTagName" is `MyTag4` and "numNumberOfTags" is `5`, values will be read from the tags `MyTag4, MyTag5, MyTag6, MyTag7`, and `MyTag8` and then written to the CSV file.

Returned value

If this function is successfully executed, it returns the position of the last byte written to the CSV file (including 0 if no bytes were written), which can be used in turn as the offset for the next batch of values to be written.

If this function fails, it returns a negative value.

Notes

"CSV" is an abbreviation of "comma-separated values", and in most cases, a CSV file is simply a plain text file that uses commas (,) to delimit its data fields. Only comma delimiters - as opposed to tabs, spaces, pipes, or other

characters - are supported in CSV files. Each field in the file can contain exactly one value, even if the value is empty.

When this function is executed, it will write at most one line of data. If a line feed is found before writing the specified number of tags, execution of the function will be aborted without writing any values.

Examples

FileWriteFields - Examples

```
FileWriteFields("C:\FieldFiles\FieldFile01.csv",400,"IntValueTag003",5)
FileWriteFields("FieldFile02.csv",-1,"IntValueTag[0]",10)
FileWriteFields("FieldFile03.csv",0,"IntValueTag[IndexTag]",7)
```

19.15.15 FileWriteMessage

The function "FileWriteMessage" reads a message (i.e., a string) from a project tag, and then it writes that message to a text file.

Function	Group	Execution	Windows	Embedded	Thin Client
FileWriteMessage	File	synchronous	supported	supported	not supported

Syntax

```
FileWriteMessage(strFilename,numOffset,strMessage,numAddLineFeed)
```

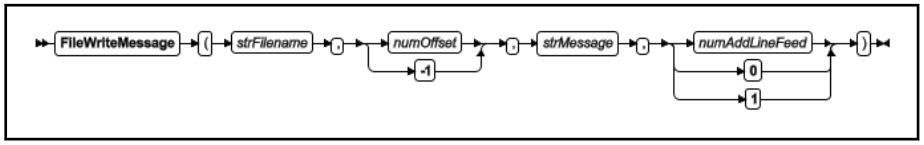


Fig. 19-42: Syntax

Inputs

strFilename

The file path and/or name of the text file. If the file is located inside your project folder, you can specify either just the file name or the file path relative to that folder. If the file is located outside your project folder, you must specify the absolute file path.

numOffset

The number of bytes to skip in the text file before writing the message. To write to the start of the file, numOffset should be `0`.

You can use this parameter to start reading from any position in the file, as long as you know how many bytes to skip. In most cases, you will simply take the value returned by the previous execution of this function (see "Returned value" below) and use it to resume reading where you previously stopped.

However, if you already know the structure of the file and where you want to start in it, you can do that. For example, if you know that each line of the file is exactly 100 bytes and you want to read from the start of the fifth line, "numOffset" should be `400`.

You can also specify a value of `-1`, which will automatically append the message to the end of the file.

☞ The number of bytes per character in a file depends on the text encoding (i.e., ANSI, UTF-8, UTF-16, or other), the byte order, and the language or character set.

strMessage

The message to be written to the text file.

numAddLineFeed

A boolean value specifying whether to add a line feed (LF) to the end of the message. If this value is `1`, a line feed is added. If this value is `0`, a line feed is not added.

Returned value

If this function is successfully executed, it returns the position of the last byte written to the text file (including 0 if no bytes were written), which can be used in turn as the offset for the next message to be written.

If this function fails, it returns a negative value.

Notes

This function can be used to write to any plain text file, as long as the correct file extension is specified in "strFilename".

Examples

FileWriteMessage - Examples

```
FileWriteMessage("C:\Data\Messages01.txt",0,MsgTag,1)
FileWriteMessage("Messages02.txt",-1,"Append this text.",0)
```

19.15.16 FindFile

FindFile is a built-in scripting function that searches for all files that match a given search string.

Function	Group	Execution	Windows	Embedded	Thin Client
FindFile	File	synchronous	supported	supported	supported (see "Notes" below)

Syntax

```
FindFile(strFile{ | ,"optTagFilesFound"{ | ,optNumTimeout } })
```

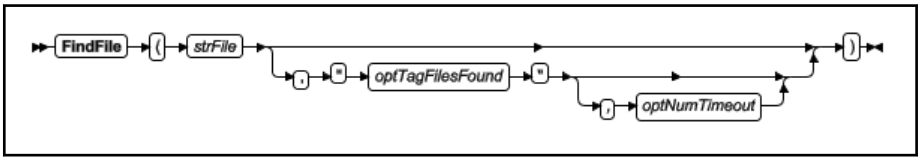


Fig. 19-43: Syntax

Inputs

strFile

The name of the file(s) to search for.

You may use wildcards (*) to find multiple files. For example, `*.gif` to find all GIF files or `log*.txt` to find all log files in a sequence (e.g., `log001.txt, log002.txt, log003.txt`).

By default, the function only searches the project folder, but you may specify a file path (either relative or absolute) to search elsewhere.

For example, if "strFile" is defined as…

```
\\volume name or IP address\Logs\log*.txt
```

…then the function will search the `Logs` directory on the specified network volume.

optTagFilesFound

An array (of String type) that will receive the names of the matching files. The array name must be enclosed in quotes; if it is not, then the function will try to get the contents of the array.

This is an optional parameter. If no value is specified, then the file names will not be saved and the function will only return the number of files found. For more information, see "Returned value" below.

The specified array will receive only the file names and not their paths, even if you define "strFile" to search outside the project folder. Also, the file names will be saved starting at array position `1` (e.g., `MyArray[1]`).

optNumTimeout

The timeout period (in milliseconds) for the function to be successfully executed.

This is an optional parameter. If no value is specified, then the project will continue searching until it has completely searched the specified directory.

☞ "optNumTimeout" is not supported if the target system / project server is a Windows Embedded device.

Returned value

This function returns one of the following values:

Value	Description
-1	Function timed out
0	No matching files found
n	Number of files found

Notes

This function can be called from scripts executed on project clients (e.g., Graphics Script, Screen Scripts, Command animations), but the function itself is always executed on the project server. By default, it only searches the project folder on the server, and if "strFile" includes a relative file path, then it is relative to that folder.

Furthermore, because the function is executed synchronously on the project server, if "strFile" is poorly defined and/or "optNumTimeout" is not used, then the entire project — both the server and the clients — may hang while it searches for the files.

Finally, "optNumTimeout" is not supported if the target system / project server is a Windows Embedded device.

Examples

Find all text files in the Server's project folder:

```
FindFile( "*.txt" )
```

Find all Microsoft Word files in the project folder and then send the names of the matching files to StringArray, within a timeout period of 1000 milliseconds:

```
FindFile( "*.doc", "StringArray", 1000 )
```

19.15.17 FindPath

Checks whether a directory exists.

Function	Group	Execution	Windows	Embedded	Thin Client
FindPath	File	synchronous	supported	supported	supported

Syntax

```
FindPath(strPathName)
```

Inputs

strPathName
The file path to search for.

Returned value

Value	Description
0	Path not found
1	Path found

Examples

Tag name	Expression
Tag	FindPath("C:\Windows")

19.15.18 GetFileAttributes

Reads the attributes of a specified file.

Function	Group	Execution	Windows	Embedded	Thin Client
GetFileAttributes	File	synchronous	supported	supported	supported

Syntax

```
GetFileAttributes(strFile)
```

Inputs

strFile
Path and name of the file the attributes are to be determined of.

Returned value

Value	Description
−1	Error
1	Write-protected
2	Hidden
4	System
16	Directory
32	Archive
128	Normal
256	Temporary

Examples

Tag name	Expression
Tag	GetFileAttributes("C:\Readme.txt")

19.15.19 GetFileTime

Reads the time and date the file was last changed.

Function	Group	Execution	Windows	Embedded	Thin Client
GetFileTime	File	synchronous	supported	supported	supported

Syntax

```
GetFileTime(strFileName,numFormat)
```

Inputs

strFileName

Path and name of the file.

numFormat

Parameter defining the format of the query:

- 0: Returns the date and time of the last file change.
- 1: Returns only the date of the last file change.
- 2: Returns only the time of the last file change.

Returned value Returns the time and/or date of the last file change.

Examples

Tag name	Expression
Tag	GetFileTime("C:\Readme.txt")

19.15.20 GetHstInfo

Returns the start time, end time, and duration of the specified history (*.HST) file.

Function	Group	Execution	Windows	Embedded	Thin Client
GetHstInfo	File	Synchronous	Supported	Supported	Supported

Syntax

```
GetHstInfo(strFileName,numInfoType)
```

Inputs

strFileName

Path and name of the history file to be read.

numFormat

Parameter specifying the type of information to be returned:

- 0: Returns the file recording start time.
- 1: Returns the file recording end time.
- 2: Returns the duration of file recording (in hours).

Returned value Returns the start time, end time and recording duration in the history (*.HST) file. If the file cannot be read or the specified information cannot be returned, an error code is returned:

Value	Description
-1	Failed to retrieve the start time. Check whether the history file exists and is valid.
-2	Failed to retrieve the end time. Check whether the history file exists and is valid.

Value	Description
-3	Internal program error. Please contact technical support.
-4	The Studio TCP/IP server returned a time that is incompatible with the format specified in the project screen or web page. Please use the "Verify Project" tool to update the project and try again (Verifying an application, page 50).

Examples

Tag name	Expression
Tag	GetHstInfo("batch", 0)
Tag	GetHstInfo("hst/02060801.hst", 1)
Tag	GetHstInfo("C:\batch.bat", 2)

19.15.21 GetLine

Gets a specific line or search string from a text file and stores the line in a String tag.

Function	Group	Execution	Windows	Embedded	Thin Client
GetLine	File	synchronous	supported	supported	supported

Syntax

```
GetLine(strFileName, Search, "tagStore", optNumCase, "optOveflowTag", optRunFromServer)
```

Inputs

strFileName

Path and name of the file to be searched. The parameter can be a fully qualified file path (e.g., **C:\File.txt**) or a simple file name (e.g., **File.txt**). In the latter case, the project searches for the file in the following directories:

- **Local Station**: The project searches for the file in the project folder and **Web** subfolder.
- **Thin Client**: If the parameter **optRunFromServer** is set to 0, the path where the file is to be searched is undetermined. If it is set to 1, the file is searched at the URL entered in the browser. If it is not found there, the search is continued under the backup URL.

☞ For executing the function on a web client, it is recommended to set the **optRunFromServer** parameter to 1 and placing the files in the **Web** subfolder of the project.

Search

This parameter serves for addressing the line to be copied. There are two possibilities of line addressing depending on the file type of the parameter:

- If it is a **String** value or tag, the function searches the text file for the first occurrence of the string and copies the entire line containing the occurrence to the tag specified by **tagStore**. Additional occurrences are counted (see Returned Values below) but not copied.
- If it is a **numeric** value or tag, the function enters that line number in the text file and copies the line to the tag specified by **tagStore**. The first line of the file is line 0.

tagStore

Name of the String tag receiving the contents of the line pointed to by the function. This name has to be set in quotation marks. Otherwise, the function uses the content of the tag instead of its name.

optNumCase

"Optional" numeric tag specifying whether the search is case-sensitive.

- 0: Not case-sensitive
- 1: Case-sensitive

optOverflowTag

"Optional" numeric tag receiving the result of the overflow verification.

- 0: OK
- 1: Overflow

optRunFromServer

"Optional" numeric tag that is only evaluated if the function is executed on a Thin Client. The parameter has the following meaning:

- 0: Retrieves the file from the Thin Client (do not use this value with names that are not fully qualified)
- 1: Retrieves the file from the web server. If the file name is not an URL, the function ignores the project path and searches for the file under the URL where the screen files are located.

Returned value

If the function is successfully executed, it returns the total number of lines in which the search string was found. Otherwise, the function returns one of the following errors:

Value	Description
0	String was not found in the target file
-1	File not found
-2	Invalid "strFileName" parameter
-3	Invalid "search" parameter
-4	Invalid "tagStore" parameter
-5	Invalid "optNumCase" parameter
-6	Invalid "optOverflowTag" parameter
-7	Invalid number of parameters
-8	Invalid line number

Notes



This function only supports ASCII and UTF 16LE text encoding. (UTF 16LE is the Unicode implementation natively supported by Windows.) If this function is used to retrieve text from a UTF-8 or UTF-16BE encoded file, invalid characters may occur.

This function can only read up to 509 characters in one function call. If a line contains more than 509 characters (i.e., alphanumeric + CR + LF), the function interprets the string as several subsequent lines with the maximum length of 509 characters. The text from line 100 with 1024 characters (i.e., 1022 alphanumeric + CR + LF) is "virtually" distributed to lines 100, 101 and 102. For this reason, the function has to be called three times.

This affects all subsequent line accesses of the function. Line 101 of the source file is assigned the virtual line number 103, etc. For this reason, the source file is to be limited to 509 characters per line to prevent unnecessarily complicated counting of lines.

... to read line 100, the function has to be called three times:

```
GetLine( "C:\FileName.txt", 100, "strTagStore[1]")  //Reads the first 509 characters
GetLine( "C:\FileName.txt", 101, "strTagStore[2]")  //Reads the second 509 characters
GetLine( "C:\FileName.txt", 102, "strTagStore[3]")  //Reads the last 6 characters
```

Examples

Tag name	Expression
Tag	GetLine("C:\TechRef v61.doc", "Studio version 6.1", "ReturnedLine")// Gets the first occurrence of "Studio version 6.1".
Tag	GetLine("C:\Readme.txt", 1, "ReturnedLine", 0, "Overflow")// Gets the second line of the file.

19.15.22 HST2TXT

Hst2Txt is a built-in function that exports Trend historical data from Rexroth WinStudio's proprietary binary format to a text file.

Function	Group	Execution	Windows	Embedded	Thin Client
HST2TXT	File	asynchronous	supported	supported	supported (see "Notes" below)

Syntax

```
Hst2Txt(strStartDate,strStartTime,numDuration,numGroup,strTargetFile,optStrSeparator,
        optNumMilliseconds,optStrFormat,optNumInterval)
```

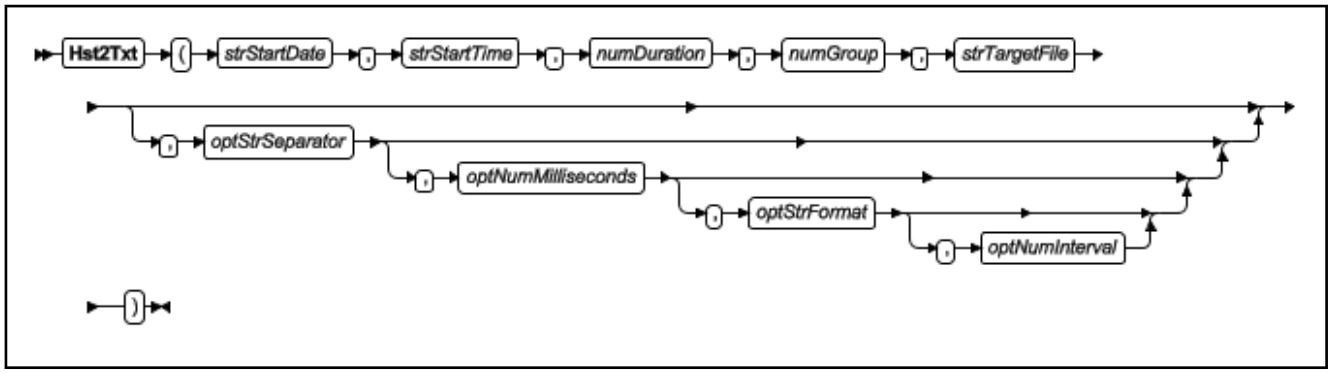


Fig. 19-44: Syntax

Inputs

strStartDate

The start date (e.g., 04/14/2002) of the data to be exported.

strStartTime

The start time (e.g., 06:30:00) of the data to be exported.

numDuration

The duration (in hours) of the data to be exported, starting from the specified start date and time.

numGroup

The trend group or worksheet number. For more information, see Trend worksheet, page 340.

strTargetFile

The name of the text file to which the data will be exported.

By default, the file must be located in your project folder (i.e., the folder that contains the file `projectname.APP`), on the computer that host the project runtime server.

If the file is or should be located in another folder, specify the complete file path.

optStrSeparator

The character that will be used to separate the values, a.k.a. the delimiter.

This parameter is optional; if no value is specified, a TAB character (`\t`) will be used, making the file a tab-delimited text file.

optNumMilliseconds

A numerical flag indicating whether to show millisecond-precision in the timestamp on each entry:

- 0: Do not show milliseconds.
- 1: Show milliseconds.

optStrFormat

The date format that will be used in the timestamp on each entry:

- "DMY": Day, Month, Year
- "MDY": Month, Day, Year
- "YMD": Year, Month, Day

If this optional parameter is not specified, the function uses the format DMY for the time stamp in the text file.

optNumInterval

The interval between entries to be exported. Only entries at this interval are exported to the specified file. For example, if "optNumInterval" is `10`, only every tenth entry is exported.

This parameter is optional; if no value is specified, every entry is exported.

Returned value

Value	Description
−3	Invalid number of parameters
−2	Dll functions not found
−1	"InStudiot.dll" not found
0	Function was executed successfully
1	Error. Previous execution of **HST2TXT** has not yet been completed

Notes

If this function is called in a project screen on a thin client, it will still write to the specified file in the project folder on the computer that hosts the project runtime server. You cannot use this function to write to a file on the thin client.

To export to a comma-separated values (CSV) file that can be easily imported into Microsoft Excel and other spreadsheet applications, specify a file name with the CSV suffix (e.g., `data.csv`) and a comma separator.

Examples

Export 0.1 hour (i.e., six minutes) of historical data from Trend group/worksheet 3 to a slash-delimited text file:

```
Hst2Txt("04/14/2002","06:30:00",0.1,3,"data.txt","\")
```

Export 0.1 hour (i.e., six minutes) of historical data from Trend group/worksheet 3 to a CSV file using the MDY date format:

```
Hst2Txt("04/14/2002","06:30:00",0.1,3,"data.csv",",",0,"MDY")
```

Export every tenth entry in 0.1 hour (i.e., six minutes) of historical data from Trend group/worksheet 3 to a CSV file using the MDY date format:

```
Hst2Txt("04/14/2002","06:30:00",0.1,3,"data.csv",",",0,"MDY",10)
```

19.15.23 HST2TXTIsRunning

Returns the status of the "HST2TXT" function.

Function	Group	Execution	Windows	Embedded	Thin Client
HST2TXTIsRunning	File	synchronous	supported	not supported	executed on the server

Syntax

```
HST2TXTIsRunning()
```

This function does not have any parameters.

Returned value

Value	Description
0	HST2TXT is still running
−1	Last conversion process was executed properly
−2	Reserved
−3	File not found. There are no history files in the configured time interval for the group specified
-4	Cannot open the **HST** file
-5	Cannot create/open the ASCII file
-6	Cannot read file information from the **HST** file

Value	Description
-7	Invalid file type
-8	Cannot read header information from the **HST** file
-9	Invalid number of tags in the header information ("0 > nTags > 250")
-10	Cannot create the Header file (**.hdr**)
-20	InStudiot.dll was not found
-30	Cannot access the dll function

Examples

HST2TXTIsRunning - Examples

```
HST2TXTIsRunning()
```

19.15.24 ImportXML

The function “ImportXML” is used to import Studio XML files into your project at runtime.

Function	Group	Execution	Windows	Embedded	Thin Client
ImportXML	File	Synchronous	Supported (see “Notes” in the following)	Not supported	Not supported

Syntax

```
ImportXML(strXMLFile,optStrDestFile,optNumFileType,optNumReplaceDuplicate)
```

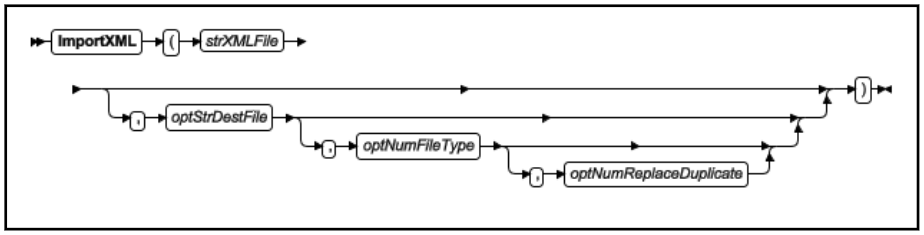


Fig. 19-45: Syntax

Inputs

strXMLFile

Path and name of the XML file to be imported. If a file name is specified without path, the runtime environment automatically browses through the subfolders “Mask” and “Web” of your project folder.

optStrDestFile

The new name of the project file if applicable.

This parameter is optional. If neither a tag nor a value is entered, the default value is „`Mask`“.

optNumFileType

A numeric flag specifying the type Studio XML file to be imported. Only Studio XML mask files are currently supported.

- 0: Studio XML mask file

This parameter is optional. If neither a tag nor a value is entered, the default value is `0`.

optNumReplaceDuplicate

A numeric flag specifying whether the imported XML file should replace an existing project file with the same name. `Objects.xml` and `Objects.scr` would be duplicates, for example.

- 0: Do not replace duplicate project file.
- 1: Replace duplicate project file.

This parameter is optional. If neither a tag nor a value is entered, the default value is `0`.

Return value This function returns one of the following possible values:

Value	Description
-2	Invalid license. This function is not available in demo mode.
-1	This function cannot be executed on the project client (i.e. Viewer). Execute the function on the project server.
0	Function executed successfully.
1	Invalid number of parameters.
2	Invalid parameter data type.
3	Duplicate file cannot be replaced.
4	Import module could not be loaded. (ImportXML does not work if your project is executed as Windows Service. For further information, refer to Executing project as Windows Service 93.)
5	File type (optNumFileType) is not supported.
6	Failed to create target file.
7	Internal error.
8	Imported mask could not be saved.
9	Could not query mask file.
10	Internal XML file error.

Notes This function can only be executed on the project server on which the Rexroth WinStudio engineering environment is installed and running with at least one runtime license. The functions StartTask, page 721, EndTask, page 697 and IsTaskRunning, page 699 can be used with the parameter "Studio" to control the engineering environment via the program.

For more information on the Studio XML mask files and how they are created, refer to Importing Studio XML masks, page 310.

Examples *Browse for Maske1.xml in the subfolders "Mask" and "Web", import it to create Maske1.scr if this file does not yet exist:*

```
ImportXML("Screen1.xml")
```

Browse for Maske1.xml in the specified file path, import it to create Maske1.scr if this file does not yet exist:

```
ImportXML("C:\Users\username\Documents\WinStudio v7.4 Projects\SMA_Project\Screen\Screen1.xml")
```

Browse for Maske1.xml in the specified file path, import it to create Maske9.scr if this file does not yet exist:

```
ImportXML("C:\Users\username\Documents\WinStudio v7.4 Projects\SMA_Project\Screen\Screen1.xml",
                                                                    "Screen9.scr")
```

Browse for Maske1.xml in the specified file path, import it to replace Maske9.scr:

```
ImportXML("C:\Users\username\Documents\WinStudio v7.4 Projects\SMA_Project\Screen\Screen1.xml",
                                                                    "Screen9.scr",0,1)
```

19.15.25 LookupContains

This function checks whether an external file contains the specified keyword in its key column.

Function	Group	Execution	Windows	Embedded	Thin Client
LookupContains	File	synchronous	supported	supported	not supported

Syntax

```
LookupContains(strKey)
```

Inputs

strKey
The keyword to look for in the key column of the file.

Returned value

This function returns the following possible values:

Value	Description
0	Specified keyword not found
1	Specified keyword found



The external file has to be already loaded by calling the LookupLoad, page 752 function.

Examples

LookupContains - Examples

```
LookupContains( "customer167" )
```

19.15.26 LookupGet

This function calls a value from a table in an external file. The value in the values column is referenced by means of a key column.

Function	Group	Execution	Windows	Embedded	Thin Client
LookupGet	File	synchronous	supported	supported	not supported

Syntax

```
LookupGet(strKey)
```

Inputs

strKey
The keyword to look for in the key column of the file.

Returned value

This function returns the value referenced by means of the keyword (as String).

If no value is found, this function returns **strKey**.



The external file has to be already loaded by calling the LookupLoad, page 752 function.

Examples

LookupGet - Examples

```
LookupGet( "customer167" )
```

19.15.27 LookupLoad

This function loads an external text file that can be used to look up table values. One column of the file is designated as the keywords column and another column is designated as the values column.

Function	Group	Execution	Windows	Embedded	Thin Client
LookupLoad	File	synchronous	supported	supported	not supported

Syntax

```
LookupLoad(strFileName,numColKey,numColValue,strDelimiters)
```

Inputs

strFileName

Path and name of the external file.

numColKey

The number of the column/field that contains the keywords.

numColValue

The number of the column/field that contains the desired values.

strDelimiters

The delimiter that separates the columns/fields.

Returned value

This function returns the number of rows in the specified file.

If the specified file cannot be found, this function returns a negative number as an error code.



This function only loads the specified file. The file itself is not modified. To read the file, the LookupContains, page 751 and LookupGet, page 751 functions can be used.

Only one file can be loaded at a time. If the function is called repeatedly with a new file, the old file is deleted from the project memory.

Examples

LookupLoad - Examples

```
LookupLoad( "C:\Temp\customerlist.csv", 1, 4, "," )
```

19.15.28 PDFCreate

Creates a PDF file from the specified source file.

Function	Group	Execution	Windows	Embedded	Thin Client
PDFCreate	File	synchronous	supported	not supported	supported

Syntax

```
PDFCreate(strSourceFile{ | ,optStrPdfFile })
```



Fig. 19-46: Syntax

Inputs

strSourceFile

String specifying the file path and name of the desired source file (***.doc**, ***.txt**, or ***.rtf**). If no complete path is specified, the function looks for the source file in the project folder.

optStrPdfFile

"Optional" string parameter specifying the file path and name of the created PDF file. If this parameter is not used, i.e., if no file path or name is specified, the PDF file is saved at the same location and with the same name as the source file. Only a new extension is added. For example:

\path\MyDocument.rtf becomes **\path\MyDocument.pdf**.

☞ If the file name is entered without a path, a leading backlash ("\") is not necessary.

Returned value

Value	Description
0	Successful
1	Error in PDF profile information
3	Error saving PDF file
4	Job canceled
101	Error initializing PDF resource
102	Specified source file not found
103	Error generating PDF file
104	Incorrect number of parameters
105	Incorrect parameter type

☞ This function only supports the execution of one job at a time. If more than one user or command attempts to call the function at the same time, the function fails and returns a value of "101".

Examples

Tag name	Expression
	PDFCreate("C:\Report1.rtf")
	PDFCreate("C:\Report2.doc", "C:\Converted1.pdf")
	PDFCreate("C:\Report3.txt", "C:\Data\Converted1.pdf")

19.15.29 Print

Prints a text file.

Function	Group	Execution	Windows	Embedded	Thin Client
Print	File	asynchronous	supported	supported	supported

Syntax

```
Print(strFilePath{ | ,optNumOrientation })
```

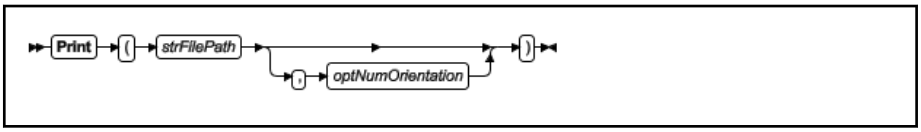


Fig. 19-47: Syntax

Inputs

strFilePath

Path and name of the text file to be printed.

optNumOrientation

Parameter specifying the page orientation:

- 0 (default) = Portrait
- 1 = Landscape

Returned value No returned values.

☞ The **optNumOrientation** parameter is not supported when running the project on a Windows Embedded target system.

Examples

Tag name	Expression
	Print("C:\ReadMe.txt")
	Print("C:\ReadMe.txt", 1)
	Print(TagFileName, 0)

☞ This function can only be used to print text files. Data in any other format (e.g., pictures, binary files, etc.) cannot be printed with this function.

19.15.30 RDFileN

Launches a "File Browser" window to select a file.

Function	Group	Execution	Windows	Embedded	Thin Client
RDFileN	File	synchronous	supported	supported	supported

Syntax

```
RDFileN("tagSelectedFile", strSearchPath, strMask, optNumChangeDir)
```

Inputs

tagSelectedFile

Parameter in form of a String tag in which the path and name of the file is entered that was selected in the file browser. The tag name **has to be** enclosed in quotation marks or the project tries to retrieve the contents of the tag. No VBScript variable name may be specified.

strSearchPath

The file path (directory) opened in the browser.

strMask

The mask to filter the files displayed in the browser.

optNumChangeDir

"Optional" numeric tag specifying whether the operator can change the browsing directory. If this parameter is not specified or set to "TRUE" (1), the open window enables to navigate to different directories. If it is set to "FALSE" (0), the window is restricted to the directory specified by **strSearchPath**.

Returned value

Value	Description
0	Successful
1	One of the parameters is not a string
2	Parameter 1 contains an invalid tag name
3	The user canceled the operation

Examples

Tag name	Expression
Tag	RDFileN("FileName", "C:\Studio\", "*.doc", 1)

19.15.31 WebGetFile

Downloads a file from a specified web address and saves it locally.

Function	Group	Execution	Windows	Embedded	Thin Client
WebGetFile	File	synchronous	supported	supported	supported

Syntax

```
WebGetFile(strURL,strLocalPath)
```

Inputs

strURL

The URL (i.e. the web address) of the file to be downloaded (String tag or string in quotation marks).

strLocalPath

The full file path of where the file is to be saved (String tag or string in quotation marks).

Returned value

Value	Description
−1	Timeout
0	File not found
1	File saved successfully

Examples

Tag name	Expression
Tag	WebGetFile("http://www.the-internet.com/myfile.txt", "C:\myfile.txt")
Tag	WebGetFile(myURL, myFilePath)

19.16 Graphic Functions

19.16.1 Graphic functions – General Information

These functions are used to handle and print WinStudio screens.

Function	Description
AutoFormat, page 756	Automatically formats a real number to a preset number of decimal places, according to the virtual table of settings created by the SetDecimalPoints function. (This is similar to the Format function, except that you do not need to specify the number of decimal places.)
ExecuteCustomCall, page 757	Opens a personalized code in the ActiveX container with an ISSymbol control. This function can for example be used to open personalized Java scripts on Web Thin Clients.
GetScrInfo, page 758	Retrieves information on an open project mask.
PrintSetup, page 760	Shows a default dialog box “Printer” on the client to select and configure a printer to print out project masks.
PrintWindow, page 761	Prints a screenshot of a project screen. The screen does not need to be open and active; the function can print a screen running in the background or even closed screen file.
ResetDecimalPointsTable, page 762	Resets the virtual table of settings created by the SetDecimalPoints function.
RGBColor, page 763	Returns the number of the color defined by the RGB (Red, Green, Blue) codes.
RGBComponent, page 763	Is a built-in scripting function that gets the level of a color component (red, green, or blue) in a specified color.
SaveScreenShot, page 764	Is a built-in function making a screenshot of a project mask and saving it as image file.
SetDecimalPoints, page 767	Sets the number of decimal places to be displayed, for a specified range of real numbers. This setting will be used by all screen objects and animations that have the Auto Format option enabled, as well as by the AutoFormat function.
SetDisplayUnit, page 768	Finds all tags and all Grid object and Trend Control object values that have a specific engineering unit (as stored in the Unit tag field), and then sets the DisplayUnit, UnitDiv, and UnitAdd fields on those tags.
SetTagDisplayUnit, page 768	Sets the DisplayUnit, UnitDiv, and UnitAdd properties on a specific tag.

19.16.2 AutoFormat

Formats a real number with a specified number of decimal places according to the virtual formatting table created by the "SetDecimalPoints" function. (This is similar to the "Format" function, except that the number of decimal places does not need to be specified.)

Function	Group	Execution	Windows	Embedded	Thin Client
AutoFormat	Graphic	synchronous	supported	supported	supported

Syntax

```
AutoFormat(numValue)
```

Inputs

numValue
The real number to be formatted.

Returned value This function returns a formatted string.

Examples In the following examples, the SetDecimalPoints, page 767 function has already been used to set 3 decimal places for values of at least 1.5 and 1 decimal place for values less than or equal to −3 before calling "AutoFormat".

Tag name	Expression
Tag	AutoFormat(1.543210) // Returned value = "1.543"
Tag	AutoFormat(−3.123456) // Returned value = "−3.1"

19.16.3 ExecuteCustomCall

The function ExecuteCustomCall invokes custom code in the ActiveX container that contains the ISSymbol control. For example, you can use this function to invoke custom JavaScript on web thin clients.

Function	Group	Execution	Windows	Embedded	Thin Client
ExecuteCustom-Call	Graphic	asynchronous	not supported	not supported	supported

Syntax

```
ExecuteCustomCall(optParam1,optParam2,…,optParamN)
```

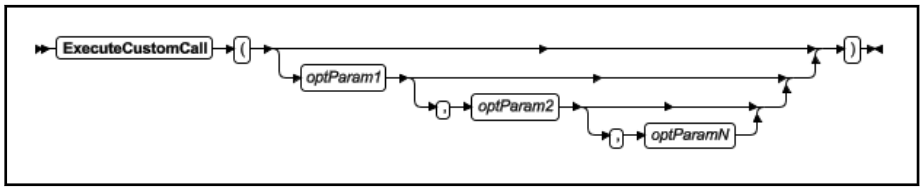


Fig. 19-48: Syntax

Inputs

optParam1, optParam2,.., optParamN

Parameters to be passed through the ISSymbol control to the ActiveX container. See "Notes" below.

Returned value This function returns one of the following possible values:

Value	Description
-1	Function is not supported. This error will be returned if the function was called by any of the Background Task worksheets (e.g., Math, Script, Scheduler) on the project server. The function only works on web thin clients, and it only affects the web thin client on which it was called.
0	Function executed successfully.

Notes This function can be used to invoke custom code in any ActiveX container that can add custom events to a control and receive VBArray objects. The following example only shows how this is done when the ActiveX container is Internet Explorer and the ISSymbol control is being used to drive a web thin client.

When you save a project screen as HTML, the resulting webpage by default is simply a frame for the ISSymbol control. The control is embedded in the webpage as an object, similar to a Flash player or a Java applet, and then it is configured to access your WinStudio project over the network.

For more information, see About Web Thin Client and Secure Viewer, page 505.

There is nothing, however, that prevents you from customizing the webpage, as long as the object configuration is maintained. And as part of customizing the webpage, you might insert the following piece of JavaScript code:

JavaScript code

```
<script language="JavaScript"><!--

    ISSymbol1.attachEvent("OnCustomCall",
        function(parameters) {
            var params = (new VBArray(parameters)).toArray();
            …
        }
    );

//--></script>
```

This code uses the method attachEvent to add the event OnCustomCall to the ISSymbol control, and then it binds a function to that event. The first line of the function sets up an array that will receive the parameters passed to the function from ExecuteCustomCall. (The parameters are passed as a VBArray object, from your WinStudio project through the ISSymbol control to the JavaScript function, and the first line of the function converts that VBArray object to a standard JavaScript array.) The rest of the function can be whatever you want it to be.

Once this code is inserted in the webpage, calling ExecuteCustomCall in your WinStudio project will cause the ISSymbol control to fire the event and execute the function in the web browser.



The method attachEvent is supported only in Internet Explorer 5 through Internet Explorer 10. It will be deprecated in Internet Explorer 11. Instead, you may use the DOM-standard methods createEvent, initEvent, and addEventListener to do the same thing.

For more information, go to: msdn.microsoft.com/library/ms536343.aspx

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

19.16.4 GetScrInfo

Gets information from the project about an open WinStudio screen.

Function	Group	Execution	Windows	Embedded	Thin Client
GetScrInfo	Graphic	synchronous	supported	supported	supported

Syntax

```
GetScrInfo(strScreenName, "tagResult", optNumResultType, optNumID)
```

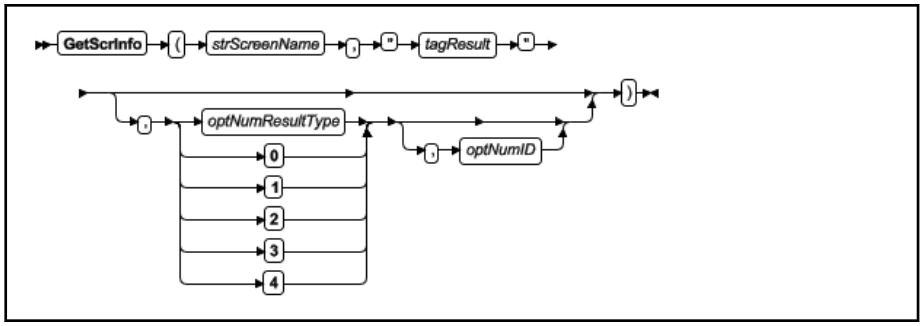


Fig. 19-49: Syntax

Inputs

strScreenName

The name of the screen the information of which is to be retrieved.

tagResult

The name of the tag that is to receive the information returned by the function.

This name has to be enclosed in quotation marks or the project tries to retrieve the contents of the tag.

To return multiple values (optNumResultType =0), an array tag has to be specified.

optNumResultType

Parameter for configuration of the return value

Value	Beschreibung
0	Default value. Writes the TOP, LEFT, BOTTOM and RIGHT screen coordinates to each consecutive position of the array tag, page 112 specified by the tagResult parameter
1	Writes the TOP screen coordinate to the tag specified by the tagResult parameter
2	Writes the LEFT screen coordinate to the tag specified by the tagResult parameter
3	Writes the BOTTOM screen coordinate to the tag specified by the tagResult parameter
4	Writes the RIGHT screen coordinate to the tag specified by the tagResult parameter

This parameter is optional. The default value is 0.

optNumID

The specific instance number of the screen. (The ID is assigned when the screen is opened with the Open, page 666, function.) This parameter is optional. The default value is 0.

Returned value

This function returns the following possible values:

Value	Description
0	Successful
-1	The first and/or second parameters are no strings
-2	Memory allocation error
-3	"optNumResultType" is zero, but "tagResult" does not specify an array tag
-4	Invalid tag by the "tagResult" parameter

Notes If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples *Retrieve the TOP, LEFT, BOTTOM and RIGHT coordinates of the "main" screen and then write them to the first four positions of the array tag TagXY:*

```
GetScrInfo("main","TagXY[0]")
```

Retrieve the BOTTOM coordinate of the "main" screen and then write it to TagXY:

```
GetScrInfo("main","TagXY",3)
```

Retrieve the LEFT coordinate of the "main" screen with ID 10 and then write it to TagXY:

```
GetScrInfo("main","TagXY",2,10)
```

19.16.5 PrintSetup

The function PrintSetup displays a standard print setup dialog box on the client, in which the user can select and configure a printer for printing project screens.

Function	Group	Execution	Windows	Embedded	Thin Client
PrintSetup	Graphic	asynchronous	supported	supported	supported

Syntax

```
PrintSetup()
```

This function does not have any parameters.

Returned value No returned values.

Notes If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

19.16.6 PrintWindow

The function PrintWindow prints a screenshot of a project screen. The screen does not need to be open and active; the function can print a screen running in the background or even closed screen file.

Function	Group	Execution	Windows	Embedded	Thin Client
PrintWindow	Graphic	asynchronous	supported	supported	supported

Syntax

```
PrintWindow(strScreenName , optNumOrientation, optNumID, optStrMnemonicList)
```

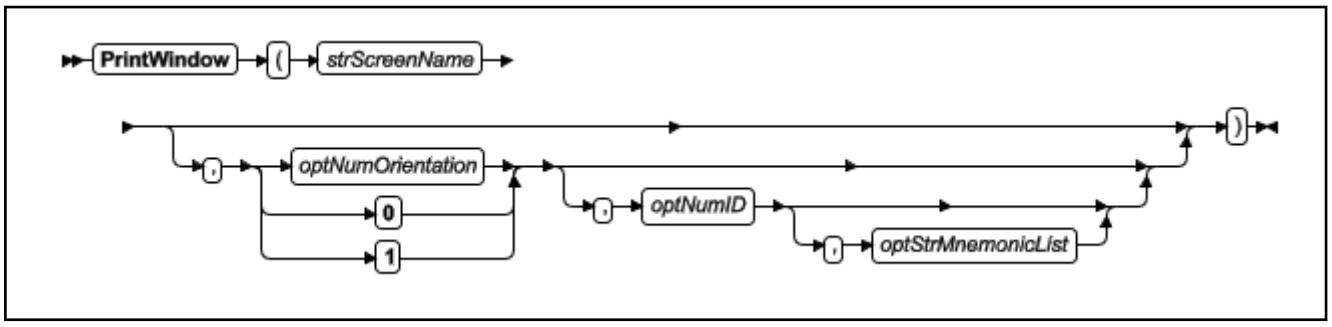


Fig. 19-50: Syntax

Inputs

strScreenName

The name of the screen to be printed. If this parameter is omitted, then the currently active screen will be printed. (This parameter must be omitted when executing the function on a Windows Embedded target system.)

optNumOrientation

Parameter specifying the paper orientation.

This parameter is optional; if no value is specified, the default value is 0.

Value	Description
0	Portrait
1	Landscape

Default value is 0.

☞ The **optNumOrientation** parameter is not supported when running the project on a Windows Embedded target system.

optNumID

The specific instance number of the screen. (The ID is assigned when the screen is opened with the Open, page 666, function.)

This is an optional command parameter. The default ID is 0.

optStrMnemonicList

A string that describes how the custom properties of generic objects or linked symbols, page 223, in the WinStudio screen are completed if the screen is printed. This string has the following syntax…

```
# Label:Value
```

…where "Label" is the name of the property and "Value" is the tag, expression or literal value the property is to receive. Also more than one transfer value can be configured by separating them with space characters. Please refer to the following section with examples.



The "optStrMnemonicList" parameter does not work for a screen that is already open. If the screen has been opened, the custom properties have already received their default values.

Returned value

This function does not return any value.

Notes

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples

Print the currently active screen in portrait orientation:

```
PrintWindow()
```

Print the screen named "Main" in landscape orientation:

```
PrintWindow("Main",1)
```

Print the screen specified by the tag "MyScreenName":

```
PrintWindow(MyScreenName)
```

Print the screen named "Main" with ID 10:

```
PrintWindow("Main",1,10)
```

Print the screen named "Main", replacing the custom properties Mne1 and Mne2 with the values of Tag1 and Tag2, respectively:

```
PrintWindow("Main",1,0,"#Mne1:Tag1 #Mne2:Tag2")
```



This function can be used to print graphical reports that include Alarm/Event Control, page 331 and Trend Control, page 349 objects.

19.16.7 ResetDecimalPointsTable

Resets the virtual table of settings created by the "SetDecimalPoints" function.

Function	Group	Execution	Windows	Embedded	Thin Client
ResetDecimal-PointsTable	Graphic	synchronous	supported	supported	supported

Syntax

```
ResetDecimalPointsTable()
```

This function does not have any parameters.

Returned value This function does not return any value.

Examples

Tag name	Expression
	ResetDecimalPointsTable( ) // Resets the virtual table of settings.

19.16.8 RGBColor

Returns the number of the color defined by the RGB (Red, Green, Blue) codes.

Function	Group	Execution	Windows	Embedded	Thin Client
RGBColor	Graphic	synchronous	supported	supported	supported

Syntax

```
RGBColor(numRed,numGreen,numBlue)
```

Inputs

numRed

RGB code for the red color component

numGreen

RGB code for the green color component

numBlue

RGB code for the blue color component

Returned value This function returns the number of the color defined by the RGB (Red, Green, Blue) codes.

Examples

Tag name	Expression
TagColor	RGBColor(51,153,102) // This function returns the value 13434828, which is the color code for Sea Green.
TagColor	RGBColor(TagRed,TagGreen,TagBlue) // This function returns the color code of the RGB values set in the tags TagRed, TagGreen and TagBlue, respectively.



A list of RGB codes and color values for the most used colors can be found in the in the color interface, page 277 section.

19.16.9 RGBComponent

"RGBComponent" is an internal scripting function determining the color code of one of the three basic colors (red, green or blue) from a color code.

Function	Group	Execution	Windows	Embedded	Thin Client
RGBComponent	Graphic	synchronous	supported	supported	supported

Syntax

```
RGBComponent(numColor,{ numComponent | 0 | 1 | 2 })
```

Inputs

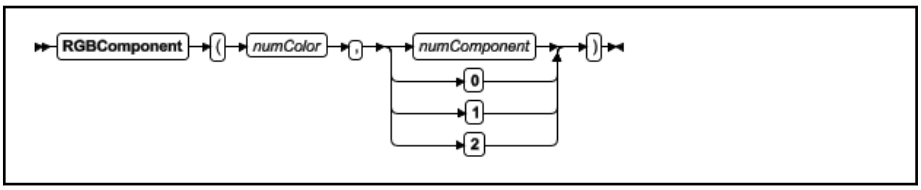


Fig. 19-51: Syntax

numColor

The decimal code for a 24-bit RGB color that can be any integer value between 0 and 16777215. (This color model is also referred to as "Truecolor" or "16 millions of colors".)

numComponent

The basic color to be determined: "0" is red, "1" is green and "2" is blue.

Returned value

This function returns an integer value between 0 and 255 representing the respective color code of the configured basic color.

For a list of frequently used RGB color codes and their equivalent "plain English" names, refer to Color interface, page 277.

Examples

Get the level of red in color code 13434828 (i.e., sea green):

```
RGBComponent( 13434828, 0 )
```

Get the level of the component specified by TagComponent in the color specified by TagCode:

```
RGBComponent( TagCode, TagComponent )
```

19.16.10 SaveScreenShot

SaveScreenShot is a built-in function making a screenshot of a project mask and saving it as image file.

Function	Group	Execution	Windows	Embedded	Thin Client
SaveScreenShot	Graphical	Synchronous	Supported	supported (only for open screens)	Supported (see “Notes” in the following)

Syntax

```
SaveScreenShot(optStrScreenName,optStrOutputFile,optNumFormat)
```

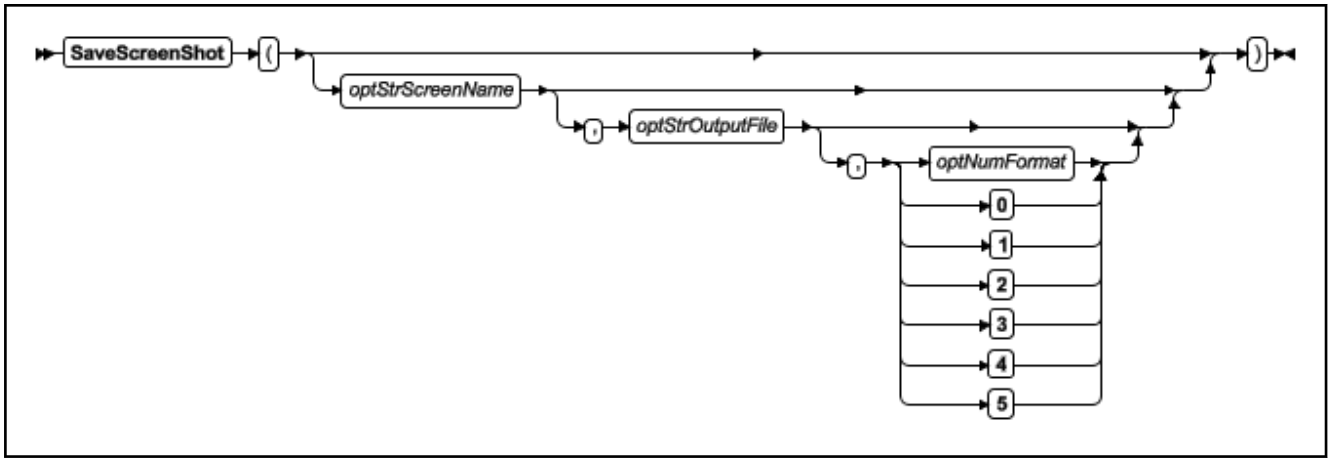


Fig. 19-52: Syntax

Inputs

optStrScreenName

The name of a screen file (`*.scr`). If no file path is specified, the file has to be located in the "Screens" folder under the project folder (e.g., `\project-name\Screen\screenname.scr`).

Whether the mask has to be open depends on where the function is executed. Refer to the following “notes”.

This parameter is optional. If no value (or "") is specified, then the currently open and active screen is used.

optStrOutputFile

The name of the output file.

This parameter can be either optional or mandatory depending on where the function is executed. Refer to the following “notes”.

optNumFormat

The format of the screen file:

Value	Description
0	BMP
1	JPG
2	PNG
3	GIF
4	TIFF
5	Auto

This parameter is optional. If no value is specified, the default value is "1" (JPG).

Return value

This function returns the following possible values:

Value	Description
0	Successful
-1	Incorrect number of parameters

Value	Description
-2	Incorrect parameter types.
-3	Invalid directory.
-4	Second parameter may not be empty.
-5	Incorrect format / invalid option for third parameter.
-6	Failed to save file.
-7	Failed to create compatible bitmap.
-10000	Project is not running. (This typically occurs when trying to use the function in the Database Spy.) See note.

Notes *This function responds differently depending on where it is executed:*

- **On the server**

  To execute the function on the project runtime server, the Viewer task (e.g. the local Viewer program of the server) has to be started. The function can be called by background tasks (e.g. by script or calculation worksheets) or in the database spy. However, it will fail if the if the Viewer task is not started as well. The reason is that the Viewer task is needed to render the screen.

  For projects running on Windows (with Rexroth WinStudio) and Windows Embedded Standard, the mask can either be open or closed. For projects running on Windows Embedded Compact (with CEView), the screen has to be open.

  "optStrOutputFile" is optional. If no value (or “ “) is specified, the file is saved in the “Web” subfolder of the project folder and the file name is either the value of optStrScreenName (if specified) or simply ScreenShot.jpg.

  Example:

  ```
  […]\Documents\WinStudio v7.4 Projects\projectname\Web
  \ScreenShot.jpg
  ```

- **Secure Viewer**

  If the function of the Secure Viewer (the Bosch Rexroth Thin Client) running on a remote station is executed, the mask can either be open or closed. Additionally, "optStrOutputFile" is required. Specify a complete file path or file name for the output file.

- **Web Thin Client**

  If the function of the Web Thin Client running in the Internet Explorer on a remote station is executed, the mask has to be open. Additionally, "optStrOutputFile" is required. Specify a complete file path or file name for the output file.



To save the output files, open the Internet Explorer with admin rights.

Examples *Creating a screenshot of the current mask and saving it as ScreenShot.jpg:*

```
SaveScreenShot()
```

Creating a screenshot of main.scr and saving it as main.jpg:

```
SaveScreenShot("main.scr")
```

Creating a screenshot of the current mask and saving it as bitmap under the name of the currently logged-in user:

```
SaveScreenShot("",UserName,0)
```

19.16.11 SetDecimalPoints

Sets the number of decimal places to be displayed, for a specified range of real numbers. This setting is to be used by all input and dynamic objects that have the **Auto Format** option enabled as well as by the **AutoFormat** function itself.

Function	Group	Execution	Windows	Embedded	Thin Client
SetDecimalPoints	Graphic	synchronous	supported	supported	supported

Syntax

```
SetDecimalPoints(numBaseValue,numDecimalPoints)
```

Inputs

numBaseValue

The base value of the range of real numbers. For negative values, the range includes all real numbers less than or equal to that value. For positive values, the range includes all real numbers greater than or equal to that number. (To specify both value ranges, the function has to be called twice with the respective values)

numDecimalPoints

The number of decimal places to be displayed for the range of real numbers specified by "numBaseValue".

Returned value

Value	Description
0	Error
1	Successful

☞ If this function is called repeatedly with different parameters, a virtual table with format settings can be created. A different number of decimal places can be set for each range of real numbers and all of the settings are saved for the duration of runtime or until the table is reset using the ResetDecimalPointsTable, page 762 function.

☞ This formatting does "not" change the actual value of any tag or expression. It only changes how the values of screen objects are displayed.

Examples

Tag name	Expression
Tag	SetDecimalPoints(1.5, 3) // Displays 3 decimal places for all real numbers greater than or equal to 1.5.
Tag	SetDecimalPoints(−3, 1) // Displays 1 decimal place for all real numbers less than or equal to −3.

19.16.12 SetDisplayUnit

Searches for all tags and all Grid object and Trend Control object values with a specific engineering unit (refer to the **Unit** tag field) and sets the **DisplayUnit**, **UnitDiv** and **UnitAdd** fields for these tags.

Function	Group	Execution	Windows	Embedded	Thin Client
SetDisplayUnit	Graphic	synchronous	supported	supported	supported

Syntax

```
SetDisplayUnit(strUnitOrigin,strDisplayUnit,numDiv,numAdd)
```

Inputs

strUnitOrigin

The original unit.

strDisplayUnit

The new value for the **DisplayUnit** tag field.

numDiv

The new value for the **UnitDiv** tag field.

numAdd

The new value for the **UnitAdd** tag field.

Returned value

Value	Description
0	Successful
-1	Incorrect number of parameters
-2	"strUnitOrigin" parameter is empty
-3	"numDiv" parameter is invalid (equal to 0)

☞ This function only affects how the tags are displayed. The actual tag values are not changed in any way.

Examples

Tag name	Expression
Tag	SetDisplayUnit("C", "F", 0.555556, 32) // For all tags and object values with a **Unit** of "C", the **DisplayUnit** tag field is set to "F", the **UnitDiv** tag field is set to 0.555556, and the **UnitAdd** tag field is set to 32.

19.16.13 SetTagDisplayUnit

Sets the **DisplayUnit**, **UnitDiv** and **UnitAdd** variable properties for a specific variable.

Function	Group	Execution	Windows	Embedded	Thin Client
SetTagDisplayUnit	Graphic	synchronous	supported	supported	supported

Syntax

```
SetTagDisplayUnit(strTagName,strDisplayUnit,numDiv,numAdd)
```

Inputs

strTagName

Name of the variable whose **DisplayUnit**, **UnitDiv** and **UnitAdd** properties are to be set. If a String variable is assigned to this parameter, it is to "include" the name of the variable whose properties are to be set.

strDisplayUnit

The new value for the **DisplayUnit** tag property.

numDiv

The new value for the **UnitDiv** tag property.

numAdd

The new value for the **UnitAdd** tag property.

Returned value

Value	Description
0	Successful
-1	Incorrect number of parameters
-2	Specified tag does not exist
-3	numDiv parameter is invalid (equal to 0)

Examples

Tag name	Expression
Tag	SetTagDisplayUnit("TagTemp", "F", 0.555556, 32) // For the tag "TagTemp", the **DisplayUnit** tag field is set to "F", the **UnitDiv** tag field is set to 0.555556, and the **UnitAdd** tag field is set to 32.

19.17 Translation functions

19.17.1 Translation functions – General information

These functions is used to translate texts at runtime.

Function	Description
Ext, page 770	Translates specified text using the active translation file.
SetLanguage, page 770	Sets the language of the project to one of the languages configured in the translation table.
SetTranslationFile, page 771	Sets the active translation file and translates all released texts in the project.
TranslationLoad, page 772	Loads an external translation file and replaces or supplements the current translation.
TranslationLookupClose, page 773	Closes a look-up map loaded before to the memory by the function "TranslationLookupLoad".

Function	Description
TranslationLookupGet, page 774	Opens the translation of a specified source text using a look-up map loaded before to the memory by the function "TranslationLookupLoad".
TranslationLookupLoad, page 774	Creates a look-up map from the translation table for the specified source and target languages and loads the map to the memory to provide it for the function "TranslationLookupGet".

19.17.2 Ext

Translates the specified text by means of the active translation file (refer to "SetLanguage" or "SetTranslationFile").

Function	Group	Execution	Windows	Embedded	Thin Client
Ext	Translation	synchronous	supported	supported	supported

Syntax

```
Ext(strText)
```

Inputs

strText
The text to be translated.

Returned value Returns the text translation using the active translation file.

Examples

Tag name	Expression
Tag	Ext("Start") // Returned value if translating to Portuguese = "Iniciar"
Tag	Ext("Stop") // Returned value if translating to German = "Stopp"

19.17.3 SetLanguage

This function changes the language of the project to one configured in the translation table.

Function	Group	Execution	Windows	Embedded	Thin Client
SetLanguage	Translation	Synchronous	Supported	Supported	Supported

Syntax

```
SetLanguage(numLanguageID)
```

Inputs

numLanguageID
The local code for the language to be selected. The language has to be already configured in the translation table, page 487,.

Returned value This function returns the following possible values:

Value	Description
0	Error
1	Successful

Local IDs are the unique language/regional codes used in Microsoft Windows.

For further information, refer to msdn.microsoft.com/library/ms91204

The function “SetLanguage” can only be executed in the viewer task (screen script or graphics script).

Examples

Setting the language to "French – France":

```
SetLanguage(1036)
```

Setting the language to “Portuguese – Portugal":

```
SetLanguage(1046)
```

19.17.4 SetTranslationFile

Sets the active translation file and translates all enabled texts within the project.

Function	Group	Execution	Windows	Embedded	Thin Client
SetTranslationFile	Translation	synchronous	supported	supported	supported

Syntax

```
SetTranslationFile(strFileName{ | ,optStrColumnName })
```



Fig. 19-53: Syntax

Eingaben

strFileName

The name of the translation file

optStrColumnName

The name of the column in the translation file the texts of which are to be used for translation.

This parameter is optional.

By default, the second column of the translation file is used.

Returned value

Value	Description
0	Successful
−1	Invalid number of parameters
−2	Incorrect parameter type
−3	Translation file could not be found or opened

Examples

Tag name	Expression
Tag	SetTranslationFile("Portuguese.tra")
Tag	SetTranslationFile("German.tra")

Tag name	Expression
Tag	SetTranslationFile("Mytranslation.csv" , "Portuguese")
Tag	SetTranslationFile("Mytranslation.csv" , "German")



For this function to work correctly, the translation option from the Project Settings, page 53, dialog have to be set.

NOTICE **A translation file has to be available in the Translation tool, page 487.**

19.17.5 TranslationLoad

This function loads an external translation file and either replaces or appends to the current translation.

Function	Group	Execution	Windows	Embedded	Thin Client
TranslationLoad	Translation	synchronous	supported	supported	not supported

Syntax

```
TranslationLoad( strFileName, numAppend )
```

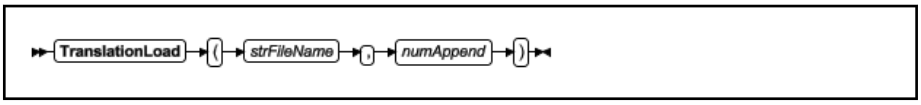


Fig. 19-54: Syntax

Inputs

strFileName

The file path and name of the translation file (*.TRA or *.CSV) to be loaded. If you specify a relative path, the function will look first in the project folder and then in the Web sub-folder. If you are running your application on the Web, you must place the file in the Web sub-folder and specify a relative path.

NumAppend

A numeric flag that indicates whether the loaded translation file replaces or appends to the existing contents of the translation table, page 487.

Specify one of the following values:

Value	Description
0	Completely replace the contents Translation Table and replace with the contents of the loaded translation file.
1	Keep the contents of the Translation Table, and then append the contents of the loaded translation file only if they are not already present.
2	Keep the contents of the Translation Table, and then append the contents of the loaded translation file, replacing any duplicates.
3	Reset the Translation Table to its initial contents when the project was run. If this value is specified, "strFileName" is ignored.

Returned value This function returns the following possible values:

Value	Description
-3	Failed to load the translation file. The file format is invalid or you do not have the necessary privileges.
-2	The specified file cannot be found.
-1	Invalid parameters.
0	Function executed successfully.

Examples

Examples

```
TranslationLoad( "alternate.tra", 0 )
TranslationLoad( "Web\pt-BR.csv", 1 )
TranslationLoad( "", 3 )
```

19.17.6 TranslationLookupClose

The function "TranslationLookupClose" closes a lookup map that was previously loaded into memory by the function "TranslationLookupLoad."

Function	Group	Execution	Windows	Embedded	Thin Client
TranslationLookupClose	Translation	synchronous	supported	supported	not supported

Syntax

```
TranslationLookupClose(numTranslationLookupID)
```

Inputs

numTranslationLookupID

The ID number of the lookup map that you want to close, as it was returned by the function TranslationLookupLoad, page 774.

Valid values range from 0 to 255.

Returned value

This function returns the following possible values:

Value	Description
0	Lookup map successfully closed.
-1	Function executed from the Viewer module, or other internal error.
-2	Invalid parameter(s).
-3	Other error.

☞ Closing a lookup map frees the system resources used by that map. You should always close a lookup map when you are done with it.

Examples

Close the lookup map specified by the ID number stored in the tag "MyLookupID":

Example

```
TranslationLookupClose(MyLookupID)
```

19.17.7 TranslationLookupGet

The function "TranslationLookupGet" gets the translation of a specified source text, using a lookup map that was previously loaded into memory by the function "TranslationLookupLoad".

Function	Group	Execution	Windows	Embedded	Thin Client
TranslationLookupGet	Translation	synchronous	supported	supported	not supported

Syntax

```
TranslationLookupGet(numTranslationLookupID,strSource)
```

Inputs

numTranslationLookupID

The ID number of the lookup map that you want to use, as it was returned by the function TranslationLookupLoad, page 774.

Valid values range from 0 to 255.

strSource

The source text for which you want to get the translation.

Returned value

If this function is successfully executed, it returns the translation of the specified source text. Otherwise, it returns one of the following possible values:

Value	Description
strSource	Source text returned untranslated, because it was not found in the lookup map.
-1	Function executed from the Viewer module, or other internal error.
-2	Invalid parameter(s).
-3	Other error.

Examples

Get the translation of "Tank #1", using the lookup map specified by the ID number stored in the tag "MyLookupID":

```
TranslationLookupGet(MyLookupID,"Tank #1")
```

Load a lookup map for translating from English (United States) to Portuguese (Brazil), and then use it to get the translation of "Tank #1":

```
TranslationLookupGet(TranslationLookupLoad(1033,1046),"Tank #1")
```

19.17.8 TranslationLookupLoad

The function "TranslationLookupLoad" creates a lookup map from the Translation Table for the specified source and target languages, and then it loads the map into memory so that it can be used by the function TranslationLookupGet.

Function	Group	Execution	Windows	Embedded	Thin Client
TranslationLookupLoad	Translation	synchronous	supported	supported	not supported

Syntax

```
TranslationLookupLoad(numSourceLanguageID,numTargetLanguageID)
```

Inputs

numSourceLanguageID, numTargetLanguageID

The locale IDs for the source and target langauges, respectively. Both languages must already be configured in the translation table, page 487,.

Returned value

If this function is successfully executed, it returns an ID number (from 0 to 255) for the lookup map. You can then use the ID number to call the functions TranslationLookupGet, page 774, and TranslationLookupClose, page 773,.

Otherwise, this function returns one of the following possible values:

Value	Description
-1	Function executed from the Viewer module, or other internal error.
-2	Invalid parameter(s).
-3	Cannot have more than 256 lookup maps loaded in memory at the same time.
-4	The source language is the same as the target language, or the specified languages have not been configured in the Translation Table.

When you call this function, you can specify any two languages that have been configured in the Translation Table, even if they were both originally configured as target languages. When the lookup map is created, it automatically cross-references the specified languages.

For example, if you configured the Translation Table for your project to include both an English-to-Spanish translation and an English-to-Portuguese translation, you can then specify Spanish and Portuguese when you call this function.

You can have up to 256 lookup maps loaded in memory at the same time, but each map uses system resources and that can affect run-time performance. You should use the function TranslationLookupClose to close a map when you are done with it.

Locale IDs are proprietary language/region codes used in Microsoft Windows.

For more information, go to: msdn.microsoft.com/library/ms91204

Examples

Load a lookup map for translating from English (United States) to Portuguese (Brazil):

```
TranslationLookupLoad(1033,1046)
```

19.18 Multimedia Functions

19.18.1 Multimedia Functions – General Information

These functions are used to play external audio and video files.

19.18.2 Play

Plays a specified WAV audio file.

Function	Group	Execution	Windows	Embedded	Thin Client
Play	Multimedia	synchronous	supported	supported	supported

For this function to work on a Thin Client, the target WAV file has to be located in the same file path as on the remote station.

Syntax

```
Play(strFileName{ | ,optNumSynchronous })
```

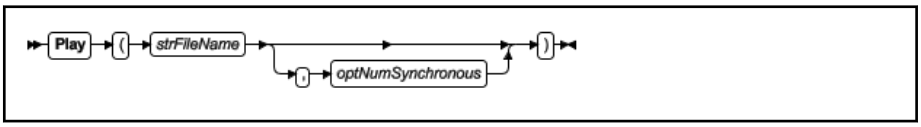


Fig. 19-55: Syntax

Inputs

strFileName

The file path and name of the WAV file to be played.

optNumSynchronous

A numeric parameter specifying whether the function is to be executed synchronously or asynchronously:

Value	Description
0	Asynchronous (i.e., the runtime environment continues without waiting for the function to complete)
1	Synchronous (i.e., the runtime environment continues and waits for the function to complete)

This is an optional parameter. If no value is specified, the default value is "0".

Returned value

This function does not return any value.

Examples

Tag name	Expression
	Play("C:\Sounds\Wav\alarm.wav")

19.19 System information functions

19.19.1 System information functions – General information

These functions are used to receive information on the computer running the project (either server or client, depending on the function) as well as to change various project settings on that computer.

Function	Description
DbVersion, page 778	Is a built-in scripting function that retrieves the version number of your project tags database.
GetAppHorizontalResolution, page 778	Gets the default horizontal screen resolution (in pixels) of the project.
GetAppPath, page 779	Returns the file path of the project folder.
GetAppVerticalResolution, page 779	Gets the default vertical screen resolution (in pixels) of the project.
GetComputerIP, page 779	Returns the first IP address of the local computer.
GetComputerName, page 780	Returns the local computer name.

Function	Description
GetCursorX, page 780	Gets the X-coordinate of the cursor on the screen.
GetCursorY, page 780	Gets the Y-coordinate of the cursor on the screen.
GetDisplayHorizontalResolution, page 781	Gets the horizontal resolution (in pixels) of the display connected to the local station.
GetDisplayVerticalResolution, page 781	Gets the vertical resolution (in pixels) of the display connected to the local station.
GetFreeMemoryCE, page 781	Returns the free memory available on a Windows Embedded device.
GetHardKeyModel, page 782	Returns the model name of the hardkey.
GetHardKeySN, page 782	Returns the serial number of the hardkey.
GetIPAll, page 783	Returns the number of IP addresses assigned to the local station and stores the IP addresses in a string array tag.
GetLicenseGracePeriod, page 784	This function returns the number of minutes until the license is expired.
GetMemoryCE, page 784	Returns the total memory available in a Windows Embedded device.
GetNetMACID, page 785	Gets the MAC ID unique code from the currently installed network adapter(s).
GetOS, page 786	The function “GetOS” retrieves the operating system of the computer hosting the project runtime server.
GetPerformanceMetric, page 787	The function “GetPerformanceMetric” returns selected metrics for the graphics performance of your project runtime client.
GetPrivateProfileString, page 791	Reads a specified parameter from such a file using the standard .ini format.
GetProductPath, page 791	Gets the path to the program directory.
GetRegValue, page 791	Gets the value of a variable in the Windows registry.
GetRegValueType, page 792	Gets the data type of the value of a variable in the Windows registry.
GetServerHostName, page 793	Gets the host name of the project's server station.
GetTickCount, page 794	Gets the current value of the clock counter.
InfoAppAlrDir, page 794	Returns the file path of the “Alarm” subfolder of the project.
InfoAppHSTDir, page 794	Returns the file path of the “History” subfolder of the project.
InfoDiskFree, page 795	Returns free disk space on the local computer.
InfoResources, page 795	Returns the available resources on the local computer.
IsActiveXReg, page 796	Determines whether an ActiveX control is registered with the operating system.
IsAppChangedOnServer, page 796	Checks whether the project files available on the server are newer than the files currently available on the client.
NoInputTime, page 797	Returns the time elapsed since the last keyboard action.
ProductVersion, page 798	Returns the program version number.
RegSaveCE, page 798	Saves the Windows Embedded system registry. This function will only work if the capability to save the registry is enabled in the Windows Embedded device image.
ReloadAppFromServer, page 799	Reloads the necessary project files from the server while maintaining the current state of the project on the client.
SaveAlarmFile, page 800	Use this function to enable/disable the saving function for alarm history and to set the path where the alarm history files have to be handled.

Function	Description
SetAppAlarmPath, page 800	Sets the alarm path for the current project.
SetAppHSTPath, page 801	Sets the file path (directory) where trend history files are to be saved in proprietary format (.HST).
SetDateFormat, page 801	Sets the format and separator for the date string.
SetKeyboardLanguage, page 802	Sets the language of all virtual keyboards in the project.
SetRegValue, page 802	Sets the value of a variable in the Windows registry.
SetWebConfig, page 804	Configures the web settings for the current project. The settings configured in the function are updated on the HTML files of the project.
SNMPGet, page 806	Gets information from computers or network devices through the SNMP protocol.
SNMPSet, page 807	Uses the Simple Network Management Protocol (SNMP) to set a value on a target computer or on a network device.
WritePrivateProfileString, page 808	Writes a specified setting to the project viewer initialization file using the standard .ini format.

19.19.2 DbVersion

"DbVersion" is an internal scripting function that retrieves the version number of the project tags database.

Function	Group	Execution	Windows	Embedded	Thin Client
DbVersion	System info	synchronous	supported	supported	supported

Syntax

```
DbVersion()
```

This function does not have any parameters.

Returned value

This function returns a numerical value equal to the version number of the tags database.



This function only applies to the native database within the project. Currently, there is no function to retrieve the version number of an external or historical database.

Examples

DbVersion - Examples

```
DbVersion()
```

19.19.3 GetAppHorizontalResolution

Gets the default horizontal screen resolution (in pixels) of the project.

Function	Group	Execution	Windows	Embedded	Thin Client
GetAppHorizontalResolution	System info	synchronous	supported	not supported	executed on the server

Syntax

```
GetAppHorizontalResolution()
```

This function does not have any parameters.

Returned value

Returns the default value as specified in the project configuration. The Windows configuration is not taken into account.

Examples

Tag name	Expression
Tag	GetAppHorizontalResolution() // Returned value = 640
Tag	GetAppHorizontalResolution() // Returned value = 800

19.19.4 GetAppPath

Returns the file path of the project folder.

Function	Group	Execution	Windows	Embedded	Thin Client
GetAppPath	System info	synchronous	supported	supported	executed on the server

Syntax

```
GetAppPath()
```

This function does not have any parameters.

Returned value

Returns the file path as a string.

Examples

Tag name	Expression
Tag	GetAppPath() // Returned value = "C:\DemoProject\"
Tag	GetAppPath() // Returned value = "C:\Studio\Projects\project_name\"



This function always returns a "\" at the end of the path.

19.19.5 GetAppVerticalResolution

Gets the default vertical screen resolution (in pixels) of the project.

Function	Group	Execution	Windows	Embedded	Thin Client
Avg	System info	synchronous	supported	supported	supported

Syntax

```
GetAppVerticalResolution()
```

This function does not have any parameters.

Returned value

Returns the default value as specified in the project configuration. The Windows configuration is not taken into account.

Examples

Tag name	Expression
Tag	GetAppVerticalResolution() // Returned value = 480
Tag	GetAppVerticalResolution() // Returned value = 600

19.19.6 GetComputerIP

Returns the first IP address of the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
GetComputerIP	System info	synchronous	supported	supported	supported

Syntax

```
GetComputerIP()
```

This function does not have any parameters.

Returned value Returns the first IP address of the local station as a string.

Examples

Tag name	Expression
Tag	GetComputerIP() // Returned value = "192.168.0.1"
Tag	GetComputerIP() // Returned value = "248.12.2.78"

19.19.7 GetComputerName

Returns the local computer name.

Function	Group	Execution	Windows	Embedded	Thin Client
GetComputer-Name	System info	synchronous	supported	not supported	supported

Syntax

```
GetComputerName()
```

This function does not have any parameters.

Returned value Returns the local computer name as a string.

Examples

Tag name	Expression
Tag	GetComputerName() // Returned value = "Terminal53"
Tag	GetComputerName() // Returned value = "BobsComputer"

19.19.8 GetCursorX

Gets the X-coordinate of the cursor on the screen.

Function	Group	Execution	Windows	Embedded	Thin Client
GetCursorX	System info	synchronous	supported	supported	supported

Syntax

```
GetCursorX()
```

This function does not have any parameters.

Returned value This function returns the X-coordinate of the cursor on the screen or "−1" if an error occurs.

Examples

Tag name	Expression
	GetCursorX( ) // Returned value = 1024

19.19.9 GetCursorY

Gets the Y-coordinate of the cursor on the screen.

Function	Group	Execution	Windows	Embedded	Thin Client
GetCursorY	System info	synchronous	supported	supported	supported

Syntax

```
GetCursorY()
```

This function does not have any parameters.

Returned value This function returns the Y-coordinate of the cursor on the screen or "−1" if an error occurs.

Examples

Tag name	Expression
	GetCursorY( ) // Returned value = 768

19.19.10 GetDisplayHorizontalResolution

Gets the horizontal resolution (in pixels) of the display connected to the local station.

Function	Group	Execution	Windows	Embedded	Thin Client
GetDisplayHorizontalResolution	System info	synchronous	supported	supported	supported

Syntax

```
GetDisplayHorizontalResolution()
```

This function does not have any parameters.

Returned value This function returns the horizontal resolution of the display as an integer.

Examples

Tag name	Expression
	GetDisplayHorizontalResolution() // Returned value = 1024

19.19.11 GetDisplayVerticalResolution

Gets the vertical resolution (in pixels) of the display connected to the local station.

Function	Group	Execution	Windows	Embedded	Thin Client
GetDisplayVerticalResolution	System info	synchronous	supported	supported	supported

Syntax

```
GetDisplayVerticalResolution()
```

This function does not have any parameters.

Returned value This function returns the vertical resolution of the display as an integer.

Examples

Tag name	Expression
	GetDisplayVerticalResolution( ) // Returned value = 768

19.19.12 GetFreeMemoryCE

Returns the free memory available on a Windows Embedded device.

Function	Group	Execution	Windows	Embedded	Thin Client
GetFreeMemoryCE	System info	synchronous	not supported	supported	supported

Syntax

```
GetFreeMemoryCE({ | optNumType })
```

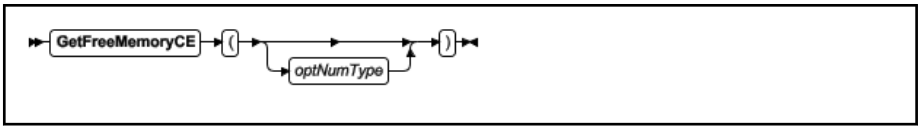


Fig. 19-56: Syntax

Inputs

optNumType

A numeric flag that specifies which type of free memory the project is to retrieve from the device:

```
0 – Total free program memory
```

This is an optional parameter. If no value is specified, the default value is "0".

Returned value

Value	Description
>0	Size of free memory in bytes
−1	**Coredll.dll** file not found
−2	GetMemoryCE, page 784 function not found
−3	Invalid optional parameter
−4	Type of memory unavailable

Examples

Tag name	Expression
Tag	GetFreeMemoryCE()

19.19.13 GetHardKeyModel

Returns the model name of the Hardkey.

Function	Group	Execution	Windows	Embedded	Thin Client
GetHardKeyModel	System info	synchronous	supported	not supported	executed on the server

Syntax

```
GetHardKeyModel()
```

This function does not have any parameters.

Returned value

If the Hardkey is installed, the function returns a string with the hardkey model name.

If the hardkey is not installed, the function returns "0".



The hardkey has to be connected before executing this function. Otherwise, the function does not execute properly.

Examples

Tag name	Expression
Tag	GetHardKeyModel() // Returned value = "Local Interface"
Tag	GetHardKeyModel() // Returned value = "Advanced Server"

19.19.14 GetHardKeySN

Returns the serial number of the hardkey.

Function	Group	Execution	Windows	Embedded	Thin Client
GetHardKeySN	System info	synchronous	supported	not supported	executed on the server

Syntax

```
GetHardkeySN()
```

This function does not have any parameters.

Returned value

If the hardkey is installed, the function returns a string with the serial number of the hardkey.

If the hardkey is not installed, the function returns "0".



The hardkey has to be connected before executing this function. Otherwise, the function does not execute properly.

Examples

Tag name	Expression
Tag	GetHardkeySN() // Returned value = 120.745
Tag	GetHardkeySN() // Returned value = 224.941

19.19.15 GetIPAll

Returns all IP addresses of the local station and stores them in a String array tag.

Function	Group	Execution	Windows	Embedded	Thin Client
GetIPAll	System info	synchronous	supported	supported	supported

Syntax

```
GetIPAll( "tagArrayIP", optRefresh )
```

Inputs

tagArrayIP

Name of the String array tag to receive the IP addresses. This name has to be enclosed in quotation marks or the project tries to get the contents of the array tag.

optRefresh

"**Optional**" tag triggering an update of this function if you use it in a Text Data Link animation, page 251. Every time this tag changes its value, the project updates the value returned by the function.

Returned value

Value	Description
"n"	Number of IP addresses found
−1	Invalid number of parameters
−2	Invalid parameter type

Examples

Tag name	Expression
Tag	GetIPAll("TagArrayIP") // Returned value = 1
Tag	GetIPAll("TagArrayIP", Second) // Returned value = 2

19.19.16 GetLicenseGracePeriod

This function returns the number of minutes until the license is expired.

Function	Group	Execution	Windows	Embedded	Thin Client
GetLicenseGrace-Period	System info	Synchronous	Supported	Not supported	Supported (refer to the following "notes")

Syntax

```
GetLicenseGracePeriod( )
```

This function does not have any parameters.

Returned value

If this function is executed successfully, it returns the amount of minutes until the license expires. Otherwise, this function returns one of the following values:

Value	Description
−1	The software is licensed.
0	The license has expired. In most of the cases, this value is not returned, as the project is stopped when the license expires.



The license period is only relevant if the software (i.e. the engineering application or the project runtime) is configured to retrieve its license from a license server and if the connection to the server is somehow lost. For further information, refer to License server, page 34.

By default, this license is activated for 60 minutes. If the period has to be changed, contact the provider of your Rexroth WinStudio software.

This function can also be called from a screen script or a command animation on a specified thin client, but the return value applies to the complete project runtime environment. Thin clients do not have own licenses. The project runtime environment is licensed instead to support a specified number of thin clients and the thin client used to execute the function, is included.

Examples

The following VBScript example shows how to check the license period and how to display a respective message for the user:

Example

```
Dim remainingGracePeriod
remainingGracePeriod = $GetLicenseGracePeriod()
If (remainingGracePeriod = -1) Then
    MsgBox "License OK"
Else
       MsgBox "Running on Grace Period! [" + remainingGracePeriod + "] minute(s) remaining"
End If
```

19.19.17 GetMemoryCE

Returns the total memory available in a Windows CE device.

Function	Group	Execution	Windows	Embedded	Thin Client
GetMemoryCE	System info	synchronous	not supported	supported	supported

Syntax

```
GetMemoryCE({ | optNumType })
```

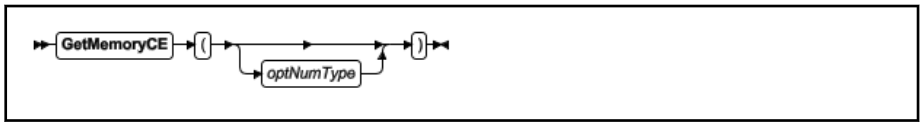


Fig. 19-57: Syntax

Inputs

optNumType

Numeric parameter to specify the type of memory to be determined:

Value	Description
0	Total program memory
1	Total storage memory
2	Total memory

This is an optional parameter. If no value is specified, the default value is "0".

Returned value

Value	Description
>0	Size of memory in bytes
−1	**CoreDll.dll** file not found
−2	GetMemoryCE function not found
−3	Invalid optional parameter

Examples

Tag name	Expression
Tag	GetMemoryCE(1)

19.19.18 GetNetMACID

Gets the MAC address of the installed network adapters.

Function	Group	Execution	Windows	Embedded	Thin Client
GetNetMACID	System info	synchronous	supported	supported	supported

Syntax

```
GetNetMACID( "optTagMACID", "optTagAdapterName" )
```

Inputs

optStrMACID

Name of the String tag in which the MAC address of the network adapter is to be stored. If more than one network adapter is installed on the station, a String type array tag can be configured for this parameter.

optStrAdapterName

Name of the String tag in which the name of the network adapter is to be stored. If more than one network adapter is installed on the station, a String type array tag can be configured for this parameter.

Returned value

Value	Description
>0	Number of network adapters found
0	No network adapters found
-1	Invalid number of parameters
-2	One of the parameters is not a string
-3	Tag configured in **optTagMACID** does not exist
-4	Tag configured in **optTagAdapterName** does not exist

Examples

Tag name	Expression
NumNIC	GetNetMACID("MACIDTag")
NumNIC	GetNetMACID("MACIDTag", "AdapterName")
NumNIC	GetNetMACID("MACIDTag[1]", "AdapterName[1]")

19.19.19 GetOS

The function GetOS gets the operating system of the computer that hosts the project runtime server.

Function	Group	Execution	Windows	Embedded	Thin Client
GetOS	System info	synchronous	supported	supported	supported

Syntax

```
GetOS()
```

This function takes no parameters.

Returned value

This function returns one of the following possible values:

Value	Description
2	Microsoft Windows oder Windows Embedded Standard, which includes the following: Microsoft Windows XP Service Pack 3 Microsoft Windows Vista Service Pack 2 Microsoft Windows 7 Service Pack 1 Microsoft Windows 8 or 8.1 (excluding Windows RT) Microsoft Windows Server 2003 Service Pack 2 Microsoft Windows Server 2008 Service Pack 2 Microsoft Windows Server 2008 R2 Service Pack 1 Microsoft Windows Server 2012 Microsoft Windows XP Embedded Service Pack 3 Microsoft Windows Embedded Standard 7
3	Windows Embedded Compact, which includes the following: Microsoft Windows Mobile or Windows CE 5.x/6.x Microsoft Windows Embedded Compact 7

Examples

Tag name	Expression
Tag	GetOS() //Returned value = 2

19.19.20 GetPerformanceMetric

The GetPerformanceMetric function returns selected metrics for the graphics performance of your project runtime client.

Function	Group	Execution	Windows	Embedded	Thin Client
GetPerformance-Metric	System info	synchronous	supported	supported	supported

Syntax

```
GetPerformanceMetric( numMetrictype, "optTagDescription" )
```

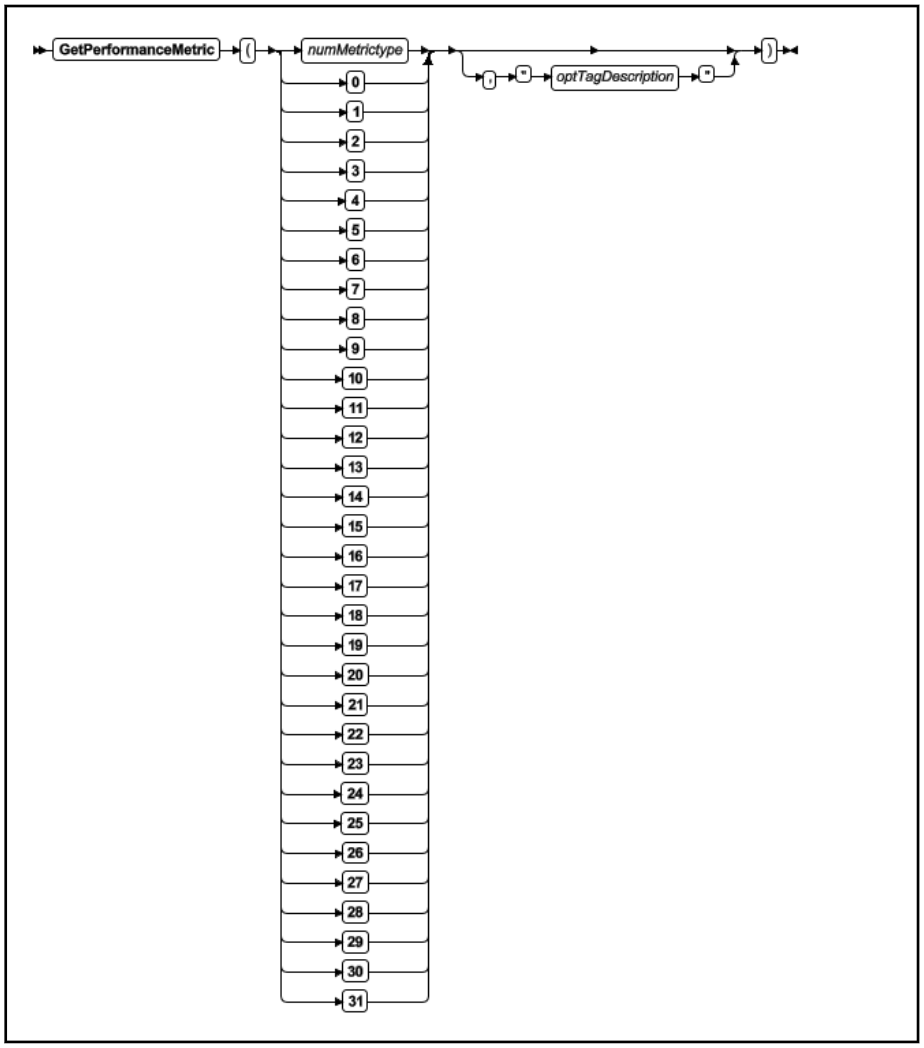


Fig. 19-58: Syntax

Inputs

numMetrictype

The type of performance metric to get, as shown in the table below:

Value	Description, including the string that is written to optTagDescription
0	`Memory Allocation Enabled` How the virtual memory is currently allocated to shared objects: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical. **Disabled** The **Enable memory allocation** option (**Project Settings ▸ Options ▸ Performance Control**) is not selected. Objects are loaded into memory only when they are actually used, and then they are immediately removed afterwards. **Always** Objects are always kept in memory, in order to increase run-time performance. **Dynamic** Objects are kept in memory until the memory is needed for new objects. If enough memory is freed, then the memory allocation state may return from Dynamic to Always. **Critical** All unused objects are cleared from memory and memory allocation is disabled until the project is restarted. This is effectively the same as Disabled (see above), except that the Enable memory allocation option is still selected.
1	`Critical Free Memory Limit (KB)` When the amount of free virtual memory (in KB) decreases to this limit, the memory allocation state is changed from Dynamic to Critical. This limit is set in **Project Settings ▸ Options ▸ Performance Control**.
2	`Min Free Memory Limit (KB)` When the amount of free virtual memory (in KB) decreases to this limit, the memory allocation state is changed from Always to Dynamic. This limit is set in **Project Settings ▸ Options ▸ Performance Control**.
3	`Free Memory (KB)` The amount of free virtual memory, in KBytes.
4	`Free Memory (%)` The amount of free virtual memory, as a percentage of total virtual memory.
5	`Total Allocation Allowed` The maximum number of total shared objects that can be allocated.
6	`Total Allocation Count` The number of total shared objects that are currently allocated.
7	`Total Allocation (%)` The percentage (count / allowed) of total shared objects that are allocated.
8	`Brush Allocation Allowed` The maximum number of shared brush objects that can be allocated.

Value	Description, including the string that is written to optTagDescription
9	`Brush Allocation Count` The number of shared brush objects that are currently allocated.
10	`Brush Allocation (%)` The percentage (count / allowed) of shared brush objects that are allocated.
11	`Brush Allocation State` How the memory for shared brush objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.
12	`Font Allocation Allowed` The maximum number of shared font objects that can be allocated.
13	`Font Allocation Count` The number of shared font objects that are currently allocated.
14	`Font Allocation (%)` The percentage (count / allowed) of shared font objects that are allocated.
15	`Font Allocation State` How the memory for shared font objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.
16	`Pen Allocation Allowed` The maximum number of shared pen objects that can be allocated.
17	`Pen Allocation Count` The number of shared pen objects that are currently allocated.
18	`Pen Allocation (%)` The percentage (count / allowed) of shared pen objects that are allocated.
19	`Pen Allocation State` How the memory for shared pen objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.
20	`Bitmap Allocation (%)` The percentage (count / allowed) of shared bitmap objects that are allocated.
21	`Bitmap Allocation Count` The number of shared bitmap objects that are currently allocated.
22	`Bitmap Allocation Allowed` The maximum number of shared bitmap objects that can be allocated.
23	`Bitmap Allocation State` How the memory for shared bitmap objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.
24	`Image Allocation Allowed` The maximum number of shared image objects that can be allocated.

Value	Description, including the string that is written to optTagDescription
25	`Image Allocation Count` The number of shared image objects that are currently allocated.
26	`Image Allocation (%)` The percentage (count / allowed) of shared image objects that are allocated.
27	`Image Allocation State` How the memory for shared image objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.
28	`Buffer Allocation Allowed` The maximum number of shared buffer objects that can be allocated.
29	`Buffer Allocation Count` The number of shared buffer objects that are currently allocated.
30	`Buffer Allocation (%)` The percentage (count / allowed) of shared buffer objects that are allocated.
31	`Buffer Allocation State` How the memory for shared buffer objects is currently allocated: 0 = Disabled; 1 = Always; 2 = Dynamic; 3 = Critical.

optTagDescription

The name of a String tag that will receive a description of the selected metric. The description is the same as shown in the table for numMetrictype above.

This parameter is optional.



The tag name should be enclosed in quotes, as shown in the syntax diagram, or else the function will try to use the value of the tag.

Returned value

If this function is successfully executed, then it returns a positive value according to the type of metric that was selected (see numMetrictype above). Otherwise, if it was not successfully executed, then it returns a negative error value:

Returned value	Description
-1	Invalid number of parameters
-2	Wrong parameter type
-3	Could not get the selected metric
-4	Tag specified by "optTagDescription" does not exist
-5	Could not write to tag specified by "optTagDescription"



Weitere Informationen über die Speicherzuteilung siehe Configure the performance control settings on page 54.

19.19.21 GetPrivateProfileString

Reads a specified parameter from an .ini file using the standard .ini format.

Function	Group	Execution	Windows	Embedded	Thin Client
GetPrivateProfile-String	System info	synchronous	supported	supported	supported

Syntax

```
GetPrivateProfileString( strSection, strName, strDefault, strFileName )
```

Inputs

strSection

The section name of the ini file to be read.

strName

The parameter of the ini file to be read.

strDefault

The default setting for the "strName" parameter to be read. If the parameter is not found in the **.ini** file, the function returns the default value specified here.

strFileName

The path and name of the **.ini** file to be read.

Returned value

Returns the value of the specified parameter.

Examples

Tag name	Expression
Tag	GetPrivateProfileString("boot loader", "timeout", "50", "C:\boot.ini") // Returned value = 30

19.19.22 GetProductPath

Gets the path to the program directory.

Function	Group	Execution	Windows	Embedded	Thin Client
GetProductPath	System info	synchronous	supported	supported	supported

Syntax

```
GetProductPath()
```

This function does not have any parameters.

Returned value

Returns the path to the program directory as a string.

Examples

Tag name	Expression
Tag	GetProductPath() // Returned value = "C:\Program Files\Rexroth \IndraWorks\WinStudio\"

19.19.23 GetRegValue

Gets the value of a registry entry in the Windows registry.

Function	Group	Execution	Windows	Embedded	Thin Client
GetRegValue	System info	synchronous	supported	supported	not supported

Syntax

```
GetRegValue( numMainKey, strKey, strValueName )
```

Inputs

numMainKey

Numeric tag to select the main key (see table):

Value	Description
0	HKEY_LOCAL_MACHINE
1	HKEY_CLASSES_ROOT
2	HKEY_CURRENT_USER
3	HKEY_USERS
4	HKEY_CURRENT_CONFIG
5	HKEY_PERFORMANCE_DATA

strKey

Path for the registry entry to be read under the main key (String variable or string in quotation marks)

strVariableName

Name of the registry entry to be retrieved. The maximum length is 255 characters.

Returned value

If the function succeeds, the value of the registry entry is returned. Otherwise one of the following error codes is returned:

Value	Description
−1	Invalid number of parameters or invalid main key
−2	Variable type is not supported. Only DWord or String values can be read from the registry
−3	Failed to read the variable value. Please check the security settings

Examples

Tag name	Expression
Tag	GetRegValue(0, "HARDWARE\DESCRIPTION\System", "SystemBiosDate") // Returned value = "08/14/03"
Tag	GetRegValue(2, "Control Panel\Current", "Color Schemes") // Returned value = "Windows Standard "

19.19.24 GetRegValueType

Gets the data type of a registry entry in the Windows registry.

Function	Group	Execution	Windows	Embedded	Thin Client
GetRegValue-Type	System info	synchronous	supported	supported	not supported

Syntax

```
GetRegValueType( numMainKey, strKey, strValueName )
```

Inputs

numMainKey

Numeric tag to select the main key (see table):

Value	Description
0	HKEY_LOCAL_MACHINE
1	HKEY_CLASSES_ROOT
2	HKEY_CURRENT_USER
3	HKEY_USERS
4	HKEY_CURRENT_CONFIG
5	HKEY_PERFORMANCE_DATA

strKey

Path of the registry entry under the main key

strVariableName

Name of the registry entry the file type of which is to be determined. The maximum length is 255 characters.

Returned value

Value	Description
1	Variable type is String
0	Variable type is DWord
−1	Invalid number of parameters or invalid main key
−2	Variable type is not supported. Only DWord or String values can be read from the registry
−3	Failed to read the variable value. Check whether appropriate security rights are available

Examples

Tag name	Expression
Tag	GetRegValueType(0, "HARDWARE\DESCRIPTION\System", "SystemBiosDate") // Returned value = 1
Tag	GetRegValueType(2, "Control Panel\Desktop", "Smooth Scroll") // Returned value = 0

19.19.25 GetServerHostName

Gets the host name of the server station on which the runtime environment is running.

Function	Group	Execution	Windows	Embedded	Thin Client
GetServerHost-Name	System info	synchronous	not supported	not supported	supported

Syntax

```
GetServerHostName()
```

This function does not have any parameters.

Returned value Server host name for ISSymbol and 127.0.0.1 for others.

19.19.26 GetTickCount

Gets the current value of the clock counter.

Function	Group	Execution	Windows	Embedded	Thin Client
GetTickCount	System info	synchronous	supported	supported	supported

Syntax

```
GetTickCount()
```

This function does not have any parameters.

Returned value Returns the number of milliseconds counted by the internal clock since initialization of the operating system.

Examples

Tag name	Expression
Tag	GetTickCount() // Returned value = 9400907

19.19.27 InfoAppAlrDir

Returns the file path of the Alarm subfolder of the project.

Function	Group	Execution	Windows	Embedded	Thin Client
InfoAppAlrDir	System info	synchronous	supported	supported	supported

Syntax

```
InfoAppAlrDir()
```

This function does not have any parameters.

Returned value Returns the **Alarm** directory of the current project as a string.

Examples

Tag name	Expression
Tag	InfoAppAlrDir() // Returned value = "C:\DemoApp\Alarm\"
Tag	InfoAppAlrDir() // Returned value = "C:\Studio\Projects\App\Alarm\"

19.19.28 InfoAppHSTDir

Returns the file path of the History subfolder of the project.

Function	Group	Execution	Windows	Embedded	Thin Client
InfoAppHSTDir	System info	synchronous	supported	supported	supported

Syntax

```
InfoAppHstDir()
```

This function does not have any parameters.

Returned value Returns the **History** directory for the current project as a string.

Examples

Tag name	Expression
Tag	InfoAppHSTDir() // Returned value = "C:\DemoApp\HST\"
Tag	InfoAppHSTDir() // Returned value = "C:\Studio\Projects\App\HST\"

19.19.29 InfoDiskFree

Returns the free disk space on the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
InfoDiskFree	System info	synchronous	supported	not supported	supported

Syntax

```
InfoDiskFree(strDisk)
```

Inputs

strDisk

The name of the disk to be checked.

Returned value Returns the available free space on the disk in bytes.

Examples

Tag name	Expression
Tag	InfoDiskFree("C") // Returned value = 2803804605.000000

19.19.30 InfoResources

Returns the available resources on the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
InfoResources	System info	synchronous	supported	supported	supported

Syntax

```
InfoResources(numSelect)
```

Inputs

numSelect

Numeric parameter specifying which resource to check:

Value	Description
0	System functions (%)
1	GDI functions (%)
2	USER functions (%)
3	Memory (in bytes)

Examples

Tag name	Expression
Tag	InfoResources(0) // Returned value = 76.000000
Tag	InfoResources(1) // Returned value = 76.000000

Tag name	Expression
Tag	InfoResources(2) // Returned value = 80.000000
Tag	InfoResources(3) // Returned value = 16150528.000000

☞ The only valid selection on a Windows PC is **3**. Selecting **0**, **1** or **2** only returns **0.000000**.

19.19.31 IsActiveXReg

Checks whether an ActiveX control element is registered on the operating system.

Function	Group	Execution	Windows	Embedded	Thin Client
IsActiveXReg	System info	synchronous	supported	supported	supported

Syntax

```
IsActiveXReg( numType, strProgIDorFileName )
```

Inputs

numType

Numeric parameter defining the format of the second **strProgIDorFileName** parameter:

Value	Description
0	Check by program ID
1	Check by file name

strProgIDorFileName

The program ID or file name including path of the ActiveX control element.

Returned value

Value	Description
0	ActiveX is not registered
1	ActiveX is registered

Examples

Tag name	Expression
Tag	IsActiveXReg(0, "ISSYMBOL.ISSymbolCtrl.1") // Returned value = 0
Tag	IsActiveXReg(1, "C:\WinNT\system32\MediaPlayer.ocx") // Returned value = 1

19.19.32 IsAppChangedOnServer

If executed on the client, this function checks whether the project files available on the server are newer than the files currently available on the client.

Function	Group	Execution	Windows	Embedded	Thin Client
IsAppChange-dOnServer	System info	synchronous	supported	supported	supported

Syntax

```
IsAppChangedOnServer( "optTagUpdateTrigger" )
```

Inputs

optTagUpdateTrigger

The name of a project tag that will server as a trigger. When the value of the specified tag changes, the function is executed. Unlike most other functions, once this function is called, it is kept in memory until the project is stopped.

To execute the function at regular intervals, use one of the system tags, page 949 like `Day` or `Month`.

This parameter is optional; if no value is specified, the function is executed immediately and not kept in memory.

The tag name should be enclosed in quotes, as shown in the syntax diagram, or else the function will try to use the value of the tag.

Returned value

Value	Description
0	FALSE (i.e., the project files on the server have not changed)
1	TRUE (i.e., the project files on the server have changed)

If the files on the server are newer, i.e., if this function returns TRUE, ReloadAppFromServer, page 799 can be used to update the client.

For this function, the server is the computer or device that hosts the downloadable project files (e.g., screens) for your thin clients. It might be different from the project runtime server (a.k.a. the data server), depending on how you deploy your project.

For further information, refer to Configuring a web solution, page 505.

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Examples

Immediately check whether the project files have changed:

```
IsAppChangedOnServer()
```

When the value of the tag CheckVersion changes, check whether the project files have changed:

```
IsAppChangedOnServer("CheckVersion")
```

19.19.33 NoInputTime

Returns the time elapsed since the last keyboard action.

Function	Group	Execution	Windows	Embedded	Thin Client
NoInputTime	System info	synchronous	supported	supported	keyboard input only

Syntax

```
NoInputTime( "optTagUpdateTrigger" )
```

Inputs

optTagUpdateTrigger

The name of a project tag. When the value of the tag changes, it triggers an update when this function is used in a Text data link animation, page 251.



The tag name should be enclosed in quotes, as shown in the syntax diagram, or else the function will try to use the value of the tag.

Returned value

Returns the time (in seconds) since the last keyboard action.



If VBScript debugging is enabled, then this function cannot be used; the normal execution cycle is suspended during debugging, so it is not possible to accurately measure the time elapsed without user input.

For more information, see Troubleshooting VBScript, General information, page 966.

Examples

Tag name	Expression
Tag	NoInputTime()



This function cannot be implemented directly from a text object.

19.19.34 ProductVersion

Returns the number of the WinStudio version.

Function	Group	Execution	Windows	Embedded	Thin Client
ProductVersion	System info	synchronous	supported	supported	supported

Syntax

```
ProductVersion()
```

This function does not have any parameters.

Returned value

Returns the WinStudio version number as a real number.

Examples

Tag name	Expression
Tag	ProductVersion() // Returned value = 7.000000

19.19.35 RegSaveCE

Saves the Windows Embedded system registry. This function can only be used if the save registry capability is enabled in the Windows Embedded device image.

Function	Group	Execution	Windows	Embedded	Thin Client
RegSaveCE	System info	synchronous	not supported	supported	not supported

Syntax

```
RegSaveCE()
```

This function does not have any parameters.

Returned value

Value	Description
0	Successful
−1	Failed to save HKEY_CLASSES_ROOT
−2	Failed to save HKEY_CURRENT_USER
−3	Failed to save HKEY_LOCAL_MACHINE
−4	Failed to save HKEY_USERS
−5	Function executed on NT platform

This function calls the **RegFlushKey** function from the **Windows CE API**. The implementation of this function is OEM dependent therefore it is not guaranteed to work with all the Windows Embedded devices.

Examples

Tag name	Expression
n/a	RegSaveCE( ) // Returned value = 0 if successful

19.19.36 ReloadAppFromServer

If executed on the client, this function reloads the necessary project files from the server while maintaining the current state of the project on the client.

Function	Group	Execution	Windows	Embedded	Thin Client
ReloadAppFrom-Server	System info	synchronous	supported	supported	supported

Syntax

```
ReloadAppFromServer()
```

This function does not have any parameters.

Returned value

This function always returns 0.

For this function, "server" means the station on which the runtime environment is actually executed and on which the TCP/IP server module is running. "client" refers to a Thin Client or Secure Viewer that is communicating with the server via TCP/IP. For further information, refer to Configuring a web solution, page 505.

If the option **Enforce Web functionality equivalence** is selected in the project settings, this function cannot be called from Global Procedures, page 927 or Script Worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Project preferences, page 69.

Before calling this function, IsAppChangedOnServer, page 796 can be used to check the version of the project files that are already on the client. If the files on the client are identical to the files on the server, this function does not need to be called.

Examples

Tag name	Expression
Tag	ReloadAppFromServer( )

19.19.37 SaveAlarmFile

Use this function to enable/disable the save function for the alarm history and to set the path where the alarm history files are to be saved.

Function	Group	Execution	Windows	Embedded	Thin Client
SaveAlarmFile	System info	synchronous	supported	supported	not supported

Syntax

```
SaveAlarmFile( numType, optRemotePath )
```

Inputs

numType

Specifies whether the alarm file is to be saved:

Value	Description
0	Disable saving the alarm file to the local disk
1	Enable saving the alarm file to the local disk
2	Enable saving the alarm file to the local disk and to the remote path specified in the "OptRemotePath" parameter

optRemotePath

Tag containing the path and name of the remote computer where the alarm file is to be saved additionally to the local computer and if **numType** is set to 2.

Returned value

Value	Description
0	Successful
1	2nd parameter is not a String
2	2nd parameter is missing

Examples

Tag name	Expression
Tag	SaveAlarmFile(0)
Tag	SaveAlarmFile(1)
Tag	SaveAlarmFile(2, "Z:\Apps\AppDemo")

19.19.38 SetAppAlarmPath

Sets the "Alarm" path for the current project.

Function	Group	Execution	Windows	Embedded	Thin Client
SetAppAlarmPath	System info	synchronous	supported	supported	executed on the server

Syntax

```
SetAppAlarmPath(strPath)
```

Inputs

strPath

The new **"Alarm"** path for the current project.

Returned value This function does not return any value.

Examples

Tag name	Expression
	SetAppAlarmPath("C:\Studio\Alarm\")

19.19.39 SetAppHSTPath

Sets the file path (directory) where trend history files are to be saved in proprietary format (**.HST**).

Function	Group	Execution	Windows	Embedded	Thin Client
SetAppHSTPath	System info	synchronous	supported	supported	executed on the server

Syntax

```
SetAppHSTPath(strPath)
```

Inputs

strPath

The file path (directory) where trend history files are to be saved.

Returned value This function does not return any value.



This function is useful if the file path is to be changed during runtime. The file path can also be set to a network drive by mapping it to the local station or by using the following syntax:

\Network Drive\File Path

Please note that this function does **not** copy existing history files from the default directory to a new one. It only sets the file path for new history files saved after the function is called.

Examples

Tag name	Expression
	SetAppHstPath("C:\Studio\History\")

19.19.40 SetDateFormat

Sets the format and separator for the date String.

Function	Group	Execution	Windows	Embedded	Thin Client
SetDateFormat	System info	synchronous	supported	supported	supported

Syntax

```
SetDateFormat( strSeparator, strFomat )
```

Inputs

strSeparator

The separator for the date String.

strFormat

String tag specifying the order of Month (M), Day (D) and Year (Y) in the date String.

Value	Description
DMY	Day, Month, Year
MDY	Month, Day, Year
YMD	Year, Month, Day

Returned value

Value	Description
0	No error
1	Invalid parameter

Examples

Tag name	Expression
Tag	SetDateFormat("/", "MDY") // Date = 04/18/2002
Tag	SetDateFormat(":", "MYD") // Date = 04:2002:18

19.19.41 SetKeyboardLanguage

Sets the language of the virtual keyboards in the project.

Function	Group	Execution	Windows	Embedded	Thin Client
SetKeyboardLanguage	System info	synchronous	supported	supported	supported

Syntax

```
SetKeyboardLanguage(strLanguage)
```

Inputs

strLanguage

String tag with the language code used for the virtual keyboards. The currently available options include:

Value	Description
EN	English (default)
GE	German
FR	French
CH	Chinese
JA	Japanese

Returned value

Value	Description
0	Successful
1	Error

Examples

Tag name	Expression
Tag	SetKeyboardLanguage("EN")
Tag	SetKeyboardLanguage(tagLanguage)

19.19.42 SetRegValue

Sets the value of a registry entry.

Function	Group	Execution	Windows	Embedded	Thin Client
SetRegValue	System info	synchronous	supported	supported	not supported

Syntax

```
SetRegValue( numMainKey, strKey, strVariableName, numType, strOrNumValue )
```

Inputs

numMainKey

Numeric tag to select the main key (see table):

Value	Description
0	HKEY_LOCAL_MACHINE
1	HKEY_CLASSES_ROOT
2	HKEY_CURRENT_USER
3	HKEY_USERS
4	HKEY_CURRENT_CONFIG
5	HKEY_PERFORMANCE_DATA

strKey

Path of the registry entry under the main key

strVariableName

Name of the registry entry to be changed. The maximum length is 255 characters.

numType

Format of the registry entry to be changed (currently supported: see table)

Value	Description
0	DWord
1	Character string

strOrNumValue

Value to be written (String variable or string in quotation marks).

Returned value

Value	Description
0	Successful
-1	Invalid number of parameters or invalid main key
-2	Invalid type
-3	Failed to read the registry key. Check whether appropriate security rights are available

Examples

Tag name	Expression
Tag	SetRegValue(0, "HARDWARE\DEVICEMAP\SERIALCOMM", "\Device \Serial1", 1, "COM3") // Returned value = 0 if successful
Tag	SetRegValue(2, "Control Panel\Desktop", "Smooth Scroll", 0, 1) // Returned value = 0 if successful

NOTICE **This entry can affect the Windows system configuration.**

Proceed with extreme caution and "only" edit the registry in case of no doubt about the configuration.

19.19.43 SetWebConfig

Configures the web settings for the current project. This function updates the settings in the HTML files of the project.

Function	Group	Execution	Windows	Embedded	Thin Client
SetWebConfig	System info	synchronous	supported	supported	executed on the server

Inputs

```
SetWebConfig( strServerIP, optStrBackupURL, optStrPathFile, optNumHostPort, optStrSecondaryServerIP,
              optNumProtocolFlag, optNumGtwPort, optStrGtwIP, optStrSecondaryGtwIP, optStrISSymbolURL )
```

Eingaben

strServerIP

Data server IP address. IP address (or hostname) of the computer where the TCP Server module of "WinStudio" is running.

optStrBackupURL

Alternative URL for the web pages of the project (optional). The Thin Client searches for the web pages under this URL if they cannot be found under the same URL written in the address field of the web browser.

optStrPathFile

File path and name of the HTML file to be updated (optional). If only the file path without a file name is specified, all of the HTML files at the specified file path are updated.

☞ An individual file can be specified (e.g., if only one file is to be updated.) This is particularly useful for projects running on Windows Embedded target systems.

optNumHostPort

TCP port number to be used by the Thin Client to exchange data with the TCP server module of "WinStudio" (optional).

optStrSecondaryServerIP

Alternative data server IP address (optional). The Thin Client attempts to connect to the TCP server module of "WinStudio" at this IP address if it cannot connect to the TCP Server module running at the IP address specified in the strServerIP parameter.

optNumProtocolFlag

If the Web Tunneling Gateway option is used, this parameter specifies whether the Thin Client uses HTTP or HTTPS (SSL – Secure Socket Layer) to exchange data with the web server (optional). If this flag has the value 0, the Thin Client uses HTTP. If this flag has the value 1, the Thin Client uses HTTPS (SSL).

optNumGtwPort

TCP port number to be used by the Thin Client to exchange data with the web server when using the Web Tunneling Gateway.

optStrGtwIP

IP address (or hostname) of the computer where the Web Tunneling Gateway is running (optional).

optStrSecondaryGtwIP

Alternative IP address of the computer where the Web Tunneling Gateway is running (optional). The Thin Client attempts to connect to the Web Tunneling Gateway at this IP address if it cannot connect to the Web Tunneling Gateway running at the IP address specified in the optStrGtwIP parameter.

optStrISSymbolURL

URL from where the updated version of ISSymbol (ActiveX control element) has to be downloaded if it is not properly registered on the Thin Client station (optional).



- Tags or expressions can be used as arguments in this function. For this reason, this function can be used to configure the web settings automatically during runtime, depending on the network settings of each project (IP address, web server URL, etc.).
- Only the first parameter of this function ("strServerIP") has to be specified. All other parameters are optional. The parameters that are not configured in the function assume the default value configured on the Web, page 71 tab of the project settings.
- The following parameters are only to be used if the Web Tunneling Gateway is used: "optNumProtocolFlag", "optNumGtwPort", "optStrGtwIP", "optStrSecondaryGtwIP" and "optStrISSymbolURL".

Returned value

Error	Description
0	No error
1	Invalid number of parameters
2	Invalid Server IP address 1
3	Invalid URL
4	Invalid optional path
5	No web pages found

Examples

Tag name	Expression
Tag	SetWebConfig("192.168.1.28")
Tag	SetWebConfig("192.168.1.28", "http://192.168.1.28/")
Tag	SetWebConfig(GetComputerIP(), "http://" + GetComputerIP() + "/")
Tag	SetWebConfig("192.168.1.28", "http://192.168.1.28/", "C:\MyWebPages\")
Tag	SetWebConfig("192.168.1.28", "http://192.168.1.28/", "C:\MyWebPages\", 1234)
Tag	SetWebConfig("192.168.1.28", "http://200.0.0.10/", "C:\MyWebPages\", 1234, "192.168.1.29", 0, 80, "200.0.0.1", "200.0.0.10", "http://200.0.0.10/MyISSymbol/")

19.19.44 SNMPGet

Gets information from computers or network devices via the Simple Network Management Protocol (SNMP).

Function	Group	Execution	Windows	Embedded	Thin Client
SNMPGet	System info	synchronous	supported	not supported	supported

Syntax

```
SNMPGet( strAddress, strCommunity, strOID, "strTagName" )
```

Inputs

strAddress

The address of the device/computer (e.g., **"127.0.0.1"** or **"localhost"**).

strCommunity

SNMP community name when communicating with the computer (e.g., **"public"**).

strOID

OID to be consulted (e.g., **".1.3.6.1.2.1.1.1.0"**).

strTagName

Name of the tag that is to receive the requested value.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Value	Description
0	No error
−1	Invalid number of parameters
−2	Invalid parameter
−3	Cannot connect to the remote device
−4	Cannot connect to the remote device
−5	GET operation failed
−6	Invalid OID
−7	Invalid tag name
−8	Invalid tag type
−9	This function is not supported by the current OS

Examples

Tag name	Expression
ErrorTag	SNMPGet("127.0.0.1", "public", ".1.3.6.1.2.1.1.1.0", "SysDescrTag") // ErrorTag will receive the error code. If the function succeeds, the value in the OID ".1.3.6.1.2.1.1.1.0" will be saved in the tag SysDescrTag.

19.19.45 SNMPSet

Uses the Simple Network Management Protocol (SNMP) to set a value on a target computer or network device.

Function	Group	Execution	Windows	Embedded	Thin Client
SNMPSet	System info	synchronous	supported	not supported	supported

Syntax

```
SNMPSet( strAddress, strCommunity, strOID, Value, optNumType )
```

Inputs

strAddress

The address of the target computer or device (e.g., "127.0.0.1" or "localhost").

strCommunity

The SNMP community name (e.g., "public") when communicating with the target computer or device.

strOID

The object ID (OID) to be set.

Value

The value to be set to the specified OID.

optNumType

A numeric value or tag of type Integer, page 111 specifying the data type. This is an optional parameter, however, if it is included, it requires one of following values:

Value	Type	Description
0	OCTETSTRING	An Octet String variable
1	INTEGER32	A 32-bit signed Integer variable
2	TIMETICKS	A time ticks variable
3	GAUGE32	A gauge variable
4	COUNTER32	A counter variable
5	IPADDRESS	An IP address variable
6	OBJECTIDENTIFIER	An object identifier variable
7	SEQUENCE	An ASN sequence variable
8	OPAQUE	A non-transparent variable

Returned value

Value	Description
0	No error
−1	Invalid number of parameters
−2	Invalid parameter
−5	SET operation failed
−7	Invalid tag name
−8	Invalid tag type
−9	This function is not supported by the current OS

If you receive any other values, please contact technical support.

Examples

Tag name	Expression
	SNMPSet("127.0.0.1", "public", ".1.3.6.1.2.1.1.1.0", 123, 1) //Sets an integer value of 123 to the specified OID on the localhost (127.0.0.1).

19.19.46 WritePrivateProfileString

Writes a specified setting to the project viewer initialization file, using the standard format of the **.ini** file.

Function	Group	Execution	Windows	Embedded	Thin Client
WritePrivateProfileString	System info	synchronous	supported	supported	supported

Syntax

```
WritePrivateProfileString ( strSection, strName, strValue, strFileName )
```

Inputs

strSection

The section name of the ini file to be written.

strName

Parameter within the selected section to be written.

strValue

Value to be written.

strFileName

Name and path of the .ini file to be written.

Returned value

The function returns 1 if the file was updated successfully.

When running on Windows Embedded devices, this function re-writes the entire file. For this reason, it is not recommended to use this function for long files on Windows Embedded devices. If it is executed on a Windows Embedded device, the function also adds the following lines to the end of the file: [FileBackUpControl] Valid=1

Examples

Tag name	Expression
Tag	WritePrivateProfileString(Section, Name, Value, FileName)
Tag	WritePrivateProfileString("Options", "ds1", "Value", "C:\Viewer.ini")

19.20 Tags Database Functions

19.20.1 Tags Database Functions – General Information

These functions are used to directly change the values of project tags.

Function	Description
ExecuteAlarmAck, page 811	Acknowledges an active alarm on the specified tag. The advantage of using this function is that if used from the Thin Client, the Alarm task will store the user name and station from which the alarm was acknowledged.
ForceTagChange, page 812	Forces the database to write a value to a tag and trigger events as if the tag changed, even if the new value is equal to the old value.
GetAlarmCount, page 812	Gets the number of active alarms.
GetAlarmInfo, page 813	Gets information about an active alarm.
GetTagValue, page 815	Gets the value of the specified tag from the project tags database.
SetTagValue, page 816	Sets the value of the specified tag in the project tags database.
TagsDBAddClass, page 817	This function adds a new class to the tags database during run time.
TagsDBAddClassMember, page 817	This function adds a new class member to an existing class during run time.
TagsDBAddTag, page 819	This function adds a new project tag to the tags database during run time.
TagsDBBeginEdit, page 820	Locks the tags database for editing during run time. You must call this function before you call any other Tags Database functions to add or remove project tags or set tag properties.

Function	Description
TagsDBEndEdit, page 821	Finishes changes made by other Tags Database functions and releases the database so that normal run-time execution may resume.
TagsDBGetAlarm, page 822	Gets the current value of a property of an alarm condition during run time.
TagsDBGetClassMember, page 825	Gets the data type or description of a specified class member during run time.
TagsDBGetClassMemberCount, page 826	Count of the members in a specified class.
TagsDBGetFirstClass, page 826	Returns the name of the first class in the tags database. It acts like an array pointer, with the array being the tags database, and it may be used in coordination with the function TagsDBGetNextClass either to generate a list of classes in the tags database or to process the classes one at a time, depending on how you write your script.
TagsDBGetFirstClassMember, page 827	Returns the name of the first member in a specified class. It acts like an array pointer, with the array being the class members, and it may be used in coordination with the function TagsDBGetNextClassMember either to generate a list of members in the class or to process the class members one at a time, depending on how you write your script.
TagsDBGetFirstTag, page 828	Returns the name of the first project tag in the tags database. It acts like an array pointer, with the array being the tags database, and it may be used in coordination with the function TagsDBGetNextTag either to generate a list of tags in the tags database or to process the tags one at a time, depending on how you write your script.
TagsDBGetNextClass, page 829	This function returns the name of the next class in the tags database, after the function TagsDBGetFirstClass has been executed to get the first class. It acts like an array pointer, with the array being the tags database, and it may be used to generate a list of classes in the tags database or to process the classes one at a time, depending on how you write your script.
TagsDBGetNextClassMember, page 829	Returns the name of the next member in the specified class, after the function TagsDBGetFirstClassMember has been executed to get the first member. It acts like an array pointer, with the array being the members of the class, and it may be used to generate a list of classes members or to process the class members one at a time, depending on how you write your script.
TagsDBGetNextTag, page 830	This function returns the name of the next tag in the tags database, after the function TagsDBGetFirstTag has been executed to get the first tag. It acts like an array pointer, with the array being the tags database, and it may be used to generate a list of tags in the tags database or to process the tags one at a time, depending on how you write your script.
TagsDBGetTagCount, page 831	Gets a count of the project tags in the tags database.
TagsDBGetTagProperty, page 832	Gets the value of a specified tag property during run time. Tag properties are the metadata on project tags, such as tag name, array size, data type, description, scope, and so on. In some cases, calling this function is the same as using tag fields (e.g., tagname->property).
TagsDBGetTrend, page 833	Determines the trend group to which a project tag is assigned.
TagsDBRemoveAlarm	Removes an alarm condition from a project tag during run time.
TagsDBRemoveClass, page 836	Removes an existing class from the tags database during run time.
TagsDBRemoveClassMember, page 836	Removes an existing class member from a specified class during run time.
TagsDBRemoveTag, page 837	Removes an existing project tag from the tags database during run time.

Function	Description
TagsDBRemoveTrend, page 838	Removes a project tag from its trend group during run time.
TagsDBSetAlarm, page 839	Sets the value of a property of an alarm condition during run time.
TagsDBSetTagProperty, page 842	Sets the value of a specified tag property during run time. Tag properties are the metadata on project tags, such as tag name, array size, data type, description, scope, and so on.
TagsDBSetTrend, page 844	Assigns a project tag to a trend group.

19.20.2 ExecuteAlarmAck

This function acknowledges an active alarm with the specified tag. The advantage of using this function is that if used from the Thin Client, the Alarm task stores the user name and station from which the alarm was acknowledged.

Function	Group	Execution	Windows	Embedded	Thin Client
ExecuteAlarmAck	Tags database	synchronous	supported	supported	supported

Syntax

```
ExecuteAlarmAck( "strTagName", optStrComment, optStrAlarmType )
ExecuteAlarmAck("strTagName",{ | ,optStrComment{ | ,optStrAlarmType } })
```

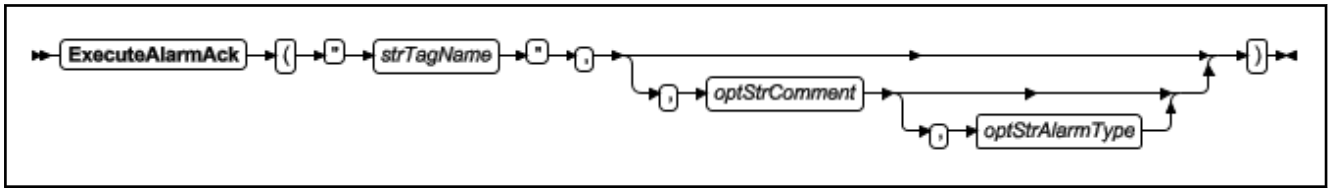


Fig. 19-59: Syntax

Inputs

strTagName

Name of the tag the alarm of which is to be acknowledged.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

optStrComment

An optional comment to send to the Alarm task along with the user name and station.

optStrAlarmType

If more than one alarm is active on the specified tag, the alarm (e.g., Hi, Lo, HiHi, LoLo) to be acknowledged can be specified. Otherwise, the function acknowledges the most recently activated alarm.

Returned value

Value	Description
0	Successfully executed
-1	Invalid number of parameters

Value	Description
-2	Invalid tag name
-3	Executed, but did not wait for confirmation from Alarms task. See note

Notes If this function is used to acknowledge an alarm, it typically waits for an acknowledgement by the Alarm task before returning a value of "0" to indicate successful execution. In some cases, however, waiting for acknowledgement may cause the runtime environment to crash. If this is the case and if the function is properly called with valid parameters, it executes as intended, however, does not wait for acknowledgement.

Examples *Acknowledge the active Hi alarm on tag A, with the comment "Hi alarm on tag A":*

```
ExecuteAlarmAck ( "A", "Hi alarm on tag A", "Hi" )
```

19.20.3 ForceTagChange

Forces writing of a value to a tag in the database so that the new value is recognized as a change in value if it has not changed compared to the old value.

Function	Group	Execution	Windows	Embedded	Thin Client
ForceTagChange	Tags database	synchronous	supported	supported	supported

Syntax

```
ForceTagChange( "strTagName", numValue )
```

Inputs

strTagName

The name of the tag forced to accept the new value.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

numValue

The new value to be written to the specified tag.

Returned value This function does not return any value.

Examples

Tag name	Expression
n/a	ForceTagChange("TagA", 5)

19.20.4 GetAlarmCount

Gets the value of the specified tag from the project tags database.

Function	Group	Execution	Windows	Embedded	Thin Client
GetAlarmCount	Tags database	synchronous	supported	supported	not supported

Syntax

```
GetAlarmCount()
```

This function has no parameters.

Returned value

This function returns the number of active alarms.

Please note that the function will return a valid value only when it is executed on the project server.

If it is called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, on a project client, it will return -1.

19.20.5 GetAlarmInfo

This function gets information about an active alarm.

Function	Group	Execution	Windows	Embedded	Thin Client
GetAlarmInfo	Tags database	synchronous	supported	supported	not supported (see "Notes" below)

Syntax

```
GetAlarmInfo(numIndex,numInfo)
```

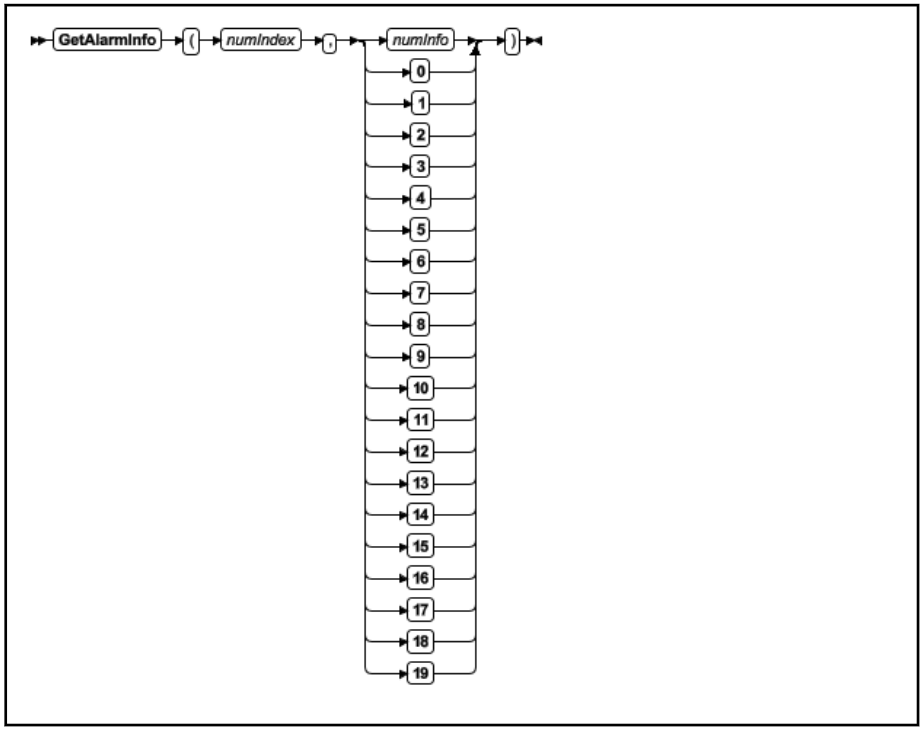


Fig. 19-60: Syntax

Inputs

numIndex

The index number of the alarm about which you want to get information.

The index number must be between 0 and the total number of alarms minus 1. To get the total number of alarms, use the function GetAlarmCount, page 812. For example, if GetAlarmCount returns 6, then numIndex must be between 0 and 5.

The list of alarms is sorted by the times when the alarms first became active, so that the oldest alarm is index 0. When an alarm is both normalized and acknowledged, it is removed from the list and the remaining alarms are shifted accordingly. For example, when the alarm at index 3 is removed, the alarm at index 4 is shifted to index 3, the alarm at index 5 is shifted to index 4, and so on.

☞ It is possible for an alarm to be normalized (i.e., become inactive) and then become active again without being acknowledged. That alarm will remain on the list of alarms throughout, and its Activation Time - that is, time when the alarm first became active - will not be updated.

numInfo

The tag to be used as a trigger to update the function. If the value of the specified tag changes, the function is executed again.

Value	Alarm Property
0	Alarm Group
1	Tag Name
2	Alarm Message
3	Alarm Type
4	Tag value when the alarm became active
5	Activation Time
6	Norm Time
7	Ack Time
8	Alarm Priority
9	Alarm Selection
10 to 19	Custom Field 1 to Custom Field 10 Tip: You can create custom fields in the Alarms worksheet, page 319.

Returned value

This function returns the current value of the specified property (numInfo) of the specified alarm (numIndex). For the property Alarm Type (i.e., numInfo is 3), this function returns one of the following possible values:

Value	Alarm Type
1	HiHi
2	Hi
4	Lo
8	LoLo
16	Rate of change
32	Deviation+
64	Deviation-

For all other properties, this function returns the actual value of the property. As such, the project tag or VBScript variable that you configure to receive the returned value should be of the appropriate data type. For example, you may configure an Integer tag to receive the value of Alarm Group or Alarm Type, but you should configure a String tag to receive the value of Tag Name or Ack Time. It is safest to always configure a String tag to receive the value, because a String tag can hold any value as a string, but that might make it more difficult to process the value after it is received. Please keep this in mind as you design and develop your project.

If the specified alarm has not yet been normalized or acknowledged, this function returns no values at all for the properties Norm Time and Ack Time, respectively.

If the function is not executed successfully, it returns one of the following error codes:

Value	Description
-1	Function was called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, on a project client. (See "Notes" below.)
- 2	Invalid alarm index (numIndex).
-3	Invalid alarm property (numInfo).

Hinweise

This function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a thin client.

For more information, see Editing the tags database during run time, page 124.

Also, this function is similar to the function TagsDBGetAlarm, page 822, except that it gets information about a specified alarm in the list of active alarms rather than about an alarm on a specified tag.

Examples

Get the type (e.g., 2, meaning Hi) of the first alarm:

```
GetAlarmInfo(0,3)
```

Get the name of the project tag (e.g., MyTag) that the second alarm is on:

```
GetAlarmInfo(1,1)
```

Get the date and time (e.g., 12/05/2013 14:11:29) when the newest alarm became active, and then use the function DateTime2UTC to convert that from the current time zone to Coordinated Universal Time (UTC):

Get the date and time:

```
DateTime2UTC(GetAlarmInfo(GetAlarmCount()-1,5))
```

19.20.6 GetTagValue

Gets the value of the specified tag from the project tags database.

Function	Group	Execution	Windows	Embedded	Thin Client
GetTagValue	Tags database	synchronous	supported	supported	supported

Syntax

```
GetTagValue( "strTagName", optNumRefresh )
```

Inputs

strTagName

The name of the tag the value of which is to be retrieved.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

optNumRefresh

The tag to be used as a trigger to update the function. If the value of the specified tag changes, the function is executed again.

Normally, a function is only executed if the object on which it is configured changes in some way, e.g., if a pushbutton object, page 197 is clicked. To execute the function at a regular interval, one of the system tags, page 108 can be used. For example, "Second", "Minute" or "Hour".

Returned value

This function only returns the value of the tag specified by **strTagName**. If the specified tag does not exist, the function returns the value 0.

The value of the tag specified by **optNumRefresh** does not affect the value returned by the function in any way.

Examples

Tag name	Expression
requiredTag = 15	GetTagName("requiredTag", Second) // Return = 15
pointerTag = "Required Tag" requiredTag = 15 optNum = Second	GetTagName(pointerTag, optNum) // Return = 15

19.20.7 SetTagValue

Sets the value of the specified tag in the project tags database.

Function	Group	Execution	Windows	Embedded	Thin Client
SetTagValue	Tags database	synchronous	supported	supported	supported

Syntax

```
SetTagValue( "strTagName", TagValue )
```

Inputs

strTagName

The name of the tag to be set.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

TagValue

The new value to be set to the specified tag.

Returned value

Value	Description
−1	Invalid tag name
0	No error

Examples

Tag name	Expression
TagA	SetTagValue("TagA", "Hello") // Value of TagA = Hello, Returned value = 0
TagA	SetTagValue("TagA", 123) // Value of TagA = 123, Returned value = 0
TagA, TagB= 15	SetTagValue("TagA", TagB) // Value of TagA = 15, Returned value = 0

19.20.8 TagsDBAddClass

This function adds a new class to the tags database during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBAddClass	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBAddClass( strClassName )
```

Inputs

strClassName

The name of the class to be added.

Returned value

This function returns one of the following possible values:

Value	Description
-5	A class with the specified name already exists in the tags database.
-4	Maximum number of classes reached.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.9 TagsDBAddClassMember

This function adds a new class to the tags database during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBAddClass	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBAddClassMember( strClassName, strMemberName, strMemberType, strDescription )
```

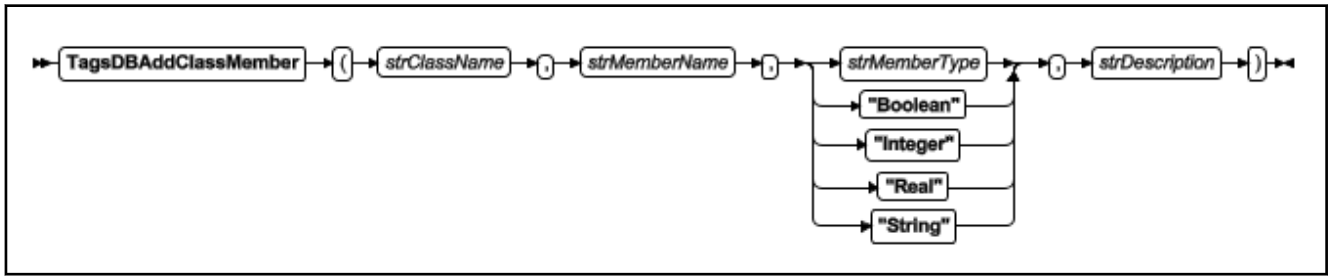


Fig. 19-61: Syntax

Inputs

strClassName

The name of the class to which the member will be added.

strMemberName

The name of the class member to be added.

strMemberType

The data type of the class member to be added. This parameter accepts only the following values: "Boolean", "Integer", "Real", or "String".

strDescription

A description of the class member to be added.

Returned value

This function returns one of the following possible values:

Value	Description
-4	The specified class does not exist in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and

- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.10 TagsDBAddTag

This function adds a new project tag to the tags database during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBAddClass	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBAddTag( strTagName, optstrTagType, optNumArraySize, optstrDescription, optNumScope )
```

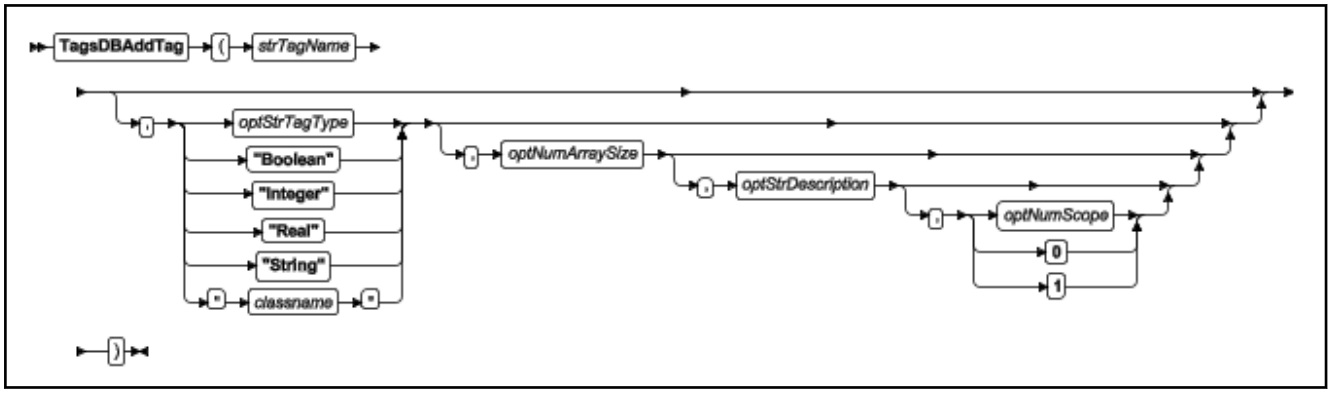


Fig. 19-62: Syntax

Inputs

strTagName

The name of the project tag to be added.

optStrTagType

The data type of the project tag to be added. This parameter accepts only the following values: `"Boolean"`, `"Integer"`, `"Real"`, `"String"`, or `"classname"`.

This parameter is optional; if you do not specify a tag or value, then the default value is `"Integer"`.

optNumArraySize

The array size of the project tag to be added.

This parameter is optional; if you do not specify a tag or value, then the default value is 0.

optStrDescription

A description of the project tag to be added.

This parameter is optional; if you do not specify a tag or value, then the default value is `""`.

optNumScope
The scope of the project tag to be added. This parameter accepts only the following values: 0 (Local) or 1 (Server). This parameter is optional; if you do not specify a tag or value, then the default value is 1.

Returned value This function returns one of the following possible values:

Value	Description
-5	The maximum number of tags (as determined by the project's target system, page 86) has been reached.
-4	A project tag with the specified name already exists in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.11 TagsDBBeginEdit

TagsDBBeginEdit is a built-in function that locks the tags database for editing during run time. You must call this function before you call any other Tags Database functions to add or remove project tags or set tag properties.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBBeginEdit	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBBeginEdit()
```

This function has no parameters.

Returned value This function returns one of the following possible values:

Value	Description
-4	The project has Collaboration enabled. To edit the tags database, you must check the tags database files out of source control. For more information, see About collaboration and source control.
-3	Project runtime is busy; an Alarm or Trend task is using the tags database. Try again later.
-2	Invalid number of parameters.
-1	Project runtime is busy; another function is editing the tags database. Try again later.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

Please note this function has a persistent effect, which means that if you call the function to lock the tags database during run time and then stop the project, the database will remain locked and you will not be able to manually edit it.

Restarting the project may or may not unlock the database, depending on how you designed your project and which function call locked the database in the first place. As such, while the project is stopped, you should use the Database Spy to manually call the function TagsDBEndEdit. When it is successfully executed, you can safely restart the project.

19.20.12 TagsDBEndEdit

TagsDBEndEdit is a built-in function that finishes changes made by other Tags Database functions and releases the database so that normal run-time execution may resume.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBEndEdit	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBEndEdit()
```

This function has no parameters.

Returned value

This function returns one of the following possible values:

Value	Description
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

Please note that when this function is executed, screens that were already open on connected clients will be reopened and their associated OnOpen screen scripts will be executed again.

19.20.13 TagsDBGetAlarm

TagsDBGetAlarm is a built-in function that gets the current value of a property of an alarm condition during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetAlarm	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetAlarm(strTagName,numAlarmType,numProperty)
```

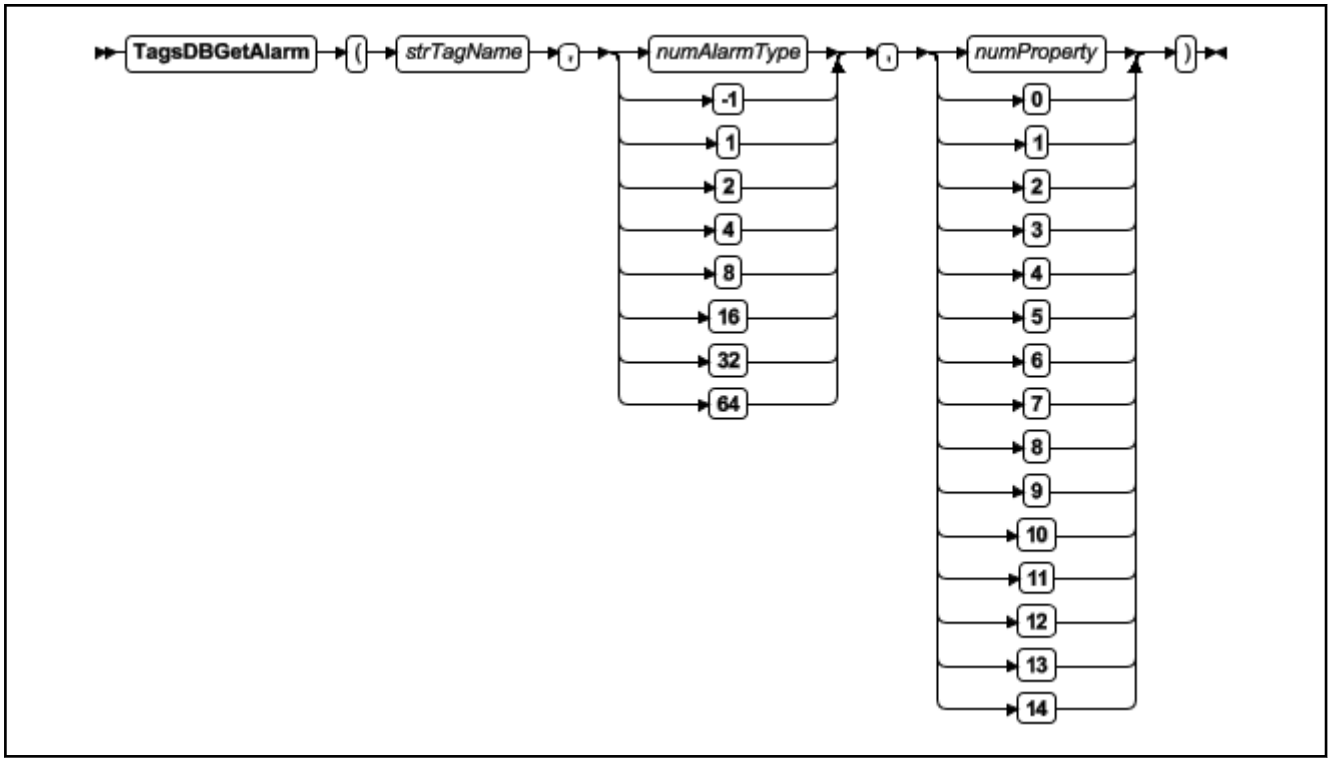


Fig. 19-63: Syntax

Inputs

strTagName

The name of the project tag on which the alarm property is set.

To get other properties on project tags, use the function TagsDB-GetTagProperty, page 832.

numAlarmType

The type of alarm on the specified project tag, identified by one of the following values:

Value	Alarm Type
-1	General properties that apply to all alarm types on the specified project tag
1	HiHi
2	Hi
4	Lo
8	LoLo
16	Rate of change
32	Deviation+
64	Deviation-

numProperty

The alarm property that you want to get.

When getting a property of a specific alarm type (i.e., if numAlarmType is greater than 0), the property is identified by one of the following values:

Value	Property	Data Type
0	Limit value	Real
1	Message	String
2	Alarm group (or work-sheet) number	Integer
3	Priority	Integer
4	Selection	String
5	Custom field 1	String
6	Custom field 2	String
7	Custom field 3	String
8	Custom field 4	String
9	Custom field 5	String
10	Custom field 6	String
11	Custom field 7	String
12	Custom field 8	String
13	Custom field 9	String
14	Custom field 10	String

When getting a property that applies to all alarm types on the specified project tag (i.e., if numAlarmType is -1), the property is identified by one of the following values:

Value	Property	Data Type
0	Alarms Enabled	Boolean
1	Remote Ack tag	String (tag name)
2	Translation Enabled	Boolean
3	Dead Band Value	Real
4	Off	String
5	On	String
6	Ack	String
7	Deviation Setpoint	String (tag name)
8	Deviation Dead Band	Real

For more information about all of these alarm properties, see About tag properties, page 122.

Returned value

If this function is executed succesfully, it returns the value of the specified property. Otherwise, it returns one of the following possible values:

Value	Description
-4	The specified project tag (strTagName) or alarm type (numAlarmType) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes

The following restrictions apply to the execution of this function:

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.14 TagsDBGetClassMember

This function gets the data type or description of a specified class member during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetClass-Member	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetClassMember( strClassName, strMemberName, numPropertyID )
```

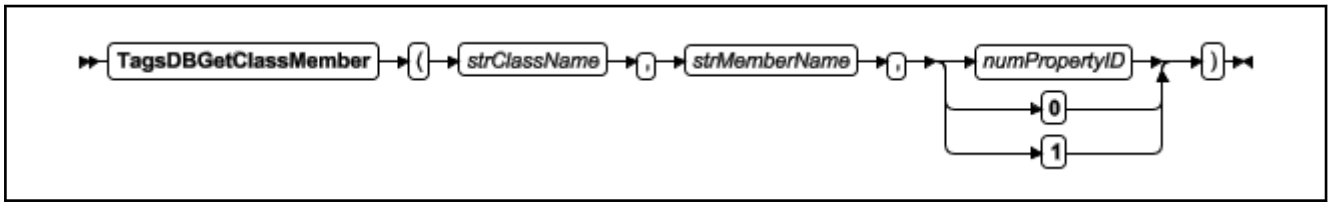


Fig. 19-64: Syntax

Inputs

strClassName

The name of the class that contains the member.

strMemberName

The name of the class member.

numPropertyID

The property to be gotten, identified by one of the following values:

Value	Property	Data Type
0	Data type	String ("Boolean", "Integer", "Real", "String")
1	Description	String

Returned value

This function returns one of the following possible values:

Value	Description
-5	The specified member does not exist in the specified class.
-4	The specified class does not exist in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes The following restrictions apply to the execution of this function:

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.15 TagsDBGetClassMemberCount

This function gets a count of the members in a specified class.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetClass-MemberCount	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetClassMemberCount( strClassName )
```

Inputs

strClassName
The name of the class in which to count members.

Returned value If this function is executed succesfully, then it returns the requested count of class members. Otherwise, it returns one of the following possible values:

Value	Description
-4	The specified class does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes The following restrictions apply to the execution of this function:

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.16 TagsDBGetFirstClass

This function returns the name of the first class in the tags database. It acts like an array pointer, with the array being the tags database, and it may be used in coordination with the function TagsDBGetNextClass, page 829, either to generate a list of classes in the tags database or to process the classes one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetFirst-Class	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetFirstClass()
```

This function has no parameters.

Returned value

If this function is executed succesfully, then it returns the name of the first class. Otherwise, it returns one of the following possible values:

Value	Description
-3	No classes found in the tags database.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.17 TagsDBGetFirstClassMember

This function returns the name of the first member in a specified class. It acts like an array pointer, with the array being the class members, and it may be used in coordination with the function TagsDBGetNextClassMember, page 829 either to generate a list of members in the class or to process the class members one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetFirst-ClassMember	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetFirstClassMember( strClassName )
```

Inputs

strClassName
The name of the class in which to get the first member.

Returned value

If this function is executed succesfully, then it returns the name of the first class member. Otherwise, it returns one of the following possible values:

Value	Description
-5	No members found in the specified class.
-4	The specified class does not exist in the tags database.

Value	Description
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.18 TagsDBGetFirstTag

This function returns the name of the first project tag in the tags database. It acts like an array pointer, with the array being the tags database, and it may be used in coordination with the function TagsDBGetNextg, page 830, either to generate a list of tags in the tags database or to process the tags one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetFirst-Tag	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetFirstTag( optNumProjectOrSystem )
```

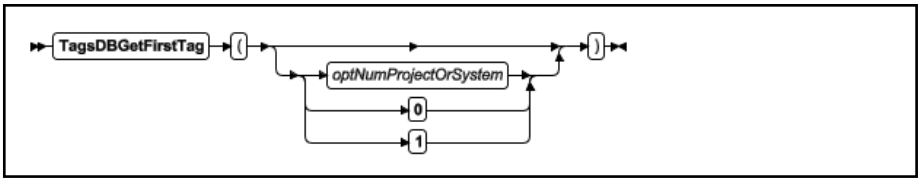


Fig. 19-65: Syntax

Inputs

optNumProjectOrSystem

Determines whether to get the first tag of the user-defined Project Tags or the pre-defined System Tags:

Value	Description
0	Project Tags
1	System Tags

This parameter is optional; if you do not specify a tag or value, then the default value is 1.

Returned value

If this function is executed succesfully, then it returns the name of the first project tag. Otherwise, it returns one of the following possible values:

Value	Description
-3	No project tags found in the tags database
-2	Invalid number of parameters.

Hinweise

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.19 TagsDBGetNextClass

This function returns the name of the next class in the tags database, after the function TagsDBGetFirstClass, page 826, has been executed to get the first class. It acts like an array pointer, with the array being the tags database, and it may be used to generate a list of classes in the tags database or to process the classes one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetNext-Class	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetNextClass()
```

This function has no parameters.

Returned value

If this function is executed succesfully, then it returns the name of the first class. Otherwise, it returns one of the following possible values:

Value	Description
-3	No class found; at end of database.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.20 TagsDBGetNextClassMember

This function returns the name of the next member in the specified class, after the function TagsDBGetFirstClassMember, page 827 has been executed to get the first member. It acts like an array pointer, with the array being the

members of the class, and it may be used to generate a list of classes members or to process the class members one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetNext-ClassMember	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetNextClassMember( strClassName )
```

Inputs

strClassName
The name of the class in which to get the next member.

Returned value

If this function is executed succesfully, then it returns the name of the first class member. Otherwise, it returns one of the following possible values:

Value	Description
-5	No members found; at end of class.
-4	The specified class does not exist in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.21 TagsDBGetNextTag

This function returns the name of the next tag in the tags database, after the function TagsDBGetFirstTag, page 828, has been executed to get the first tag. It acts like an array pointer, with the array being the tags database, and it may be used to generate a list of tags in the tags database or to process the tags one at a time, depending on how you write your script.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetNext-Tag	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetNextTag()
```

This function has no parameters.

Returned value

If this function is executed succesfully, then it returns the name of the next tag. Otherwise, it returns one of the following possible values:

Value	Description
-3	No tag found; at end of database.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

Also, this function may be called at any time; the function TagsDBBeginEdit, page 820, does not need to have been executed previously. If that is the case, however, then TagsDBGetFirstClass and TagsDBGetNextClass can only generate a list of classes. They cannot be used, along with with the other Tags Database functions, to edit the classes.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.22 TagsDBGetTagCount

This function gets a count of the project tags in the tags database.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetTag-Count	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetTagCount( numCountType )
```

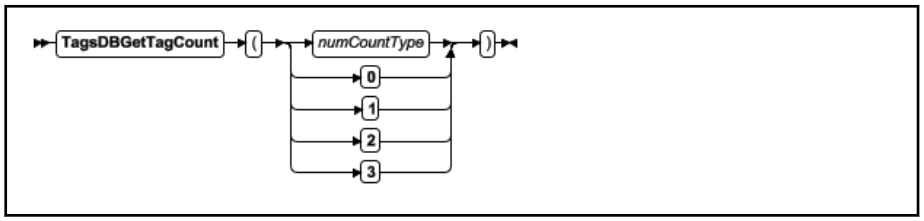


Fig. 19-66: SYntax

Inputs

numCountType

The type of count to be performed, identified by one of the following values:

Value	Description
0	The total number of project tags in the tags database.
1	The number of project tags supported by the target system, page 86.
2	The number of project tags that may still be created before exceeding the number supported by the target system.
3	The total number of pre-defined system tags in the tags database.

Returned value

If this function is executed succesfully, then it returns the name of the first project tag. Otherwise, it returns one of the following possible values:

Value	Description
-3	No project tags found in the tags database
-2	Invalid number of parameters.

Notes The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.23 TagsDBGetTagProperty

This function gets the value of a specified tag property during run time. Tag properties are the metadata on project tags, such as tag name, array size, data type, description, scope, and so on. In some cases, calling this function is the same as using tag fields (e.g., tagname->property).

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetTag-Property	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetTagProperty( strTagName, numPropertyID )
```

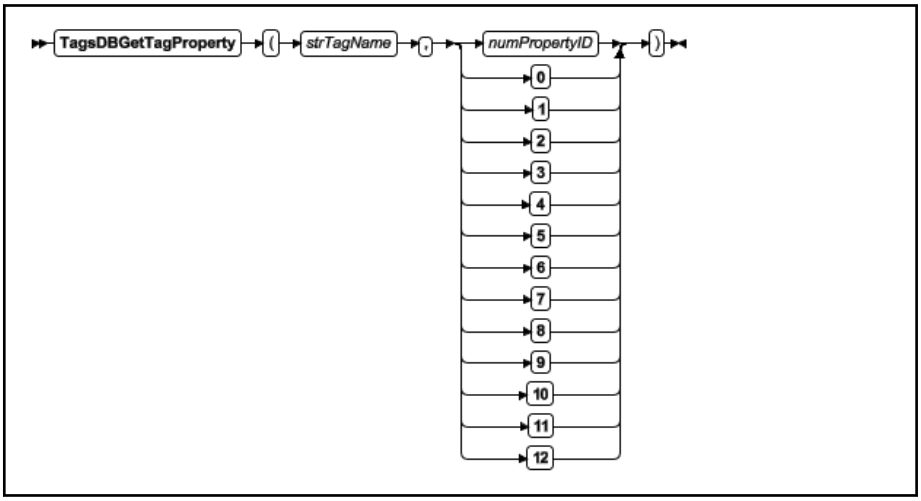


Fig. 19-67: Syntax

Inputs

strTagName

The name of the project tag from which the tag property will be gotten.

numPropertyID

The tag property to be gotten, identified by one of the following values:

Value	Property	Data Type	Same as...
0	Tag name	String	tagname->Name
1	Array size	Integer	tagname->Size
2	Data type	String ("Boolean", "Integer", "Real", "String", "classname")	-
3	Description	String	tagname->Description
4	Scope	Integer (0 for Local, 1 for Server)	-
5	Startup value	String	-

Value	Property	Data Type	Same as...
6	Minimum value	Real	tagname->Min
7	Maximum value	Real	tagname->Max
8	Engineering unit	String	tagname->Unit
9	Retentive value	Integer (0 for disabled, 1 for enabled)	-
10	Retentive properties	Integer (0 for disabled, 1 for enabled)	-
11	Dead band	Real	-
12	Smoothing	Integer (0 for disabled, 1 for enabled)	-



To get alarm conditions on project tags, use the function TagsDB-GetAlarm, page 822. Other tag properties not listed here cannot be gotten during run time.

numOrStrPropertyValue

The value to set to the tag property. The value must be of the appropriate data type.

Returned value

If this function is executed succesfully, then it returns the value of the specified tag property. Otherwise, it returns one of the following possible values:

Value	Description
-4	Project tag (as specified by strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.24 TagsDBGetTrend

This function gets the value of a specified tag property during run time. Tag properties are the metadata on project tags, such as tag name, array size, data type, description, scope, and so on. In some cases, calling this function is the same as using tag fields (e.g., tagname->property).

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBGetTrend	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBGetTrend( strTagName, numProperty )
```

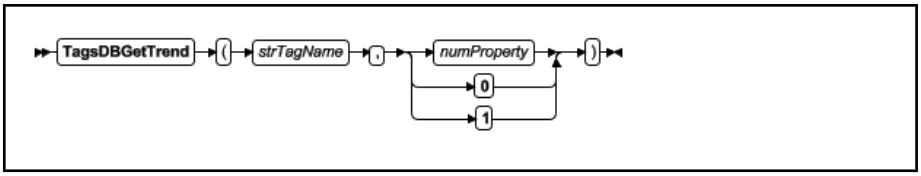


Fig. 19-68: Syntax

Inputs

strTagName

The name of the project tag that is assigned to a trend group.

numProperty

The specific property to be gotten, identified by one of the following values:

Value	Property	Data Type
0	Trend group (or worksheet) number	Integer
1	Log dead band	Real

numOrStrPropertyValue

The value to set to the tag property. The value must be of the appropriate data type.

Returned value

If this function is executed succesfully, then it returns the value of the specified property. Otherwise, it returns one of the following possible values:

Value	Description
-4	Project tag (as specified by strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.

Notes

The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.25 TagsDBRemoveAlarm

TagsDBRemoveAlarm is a built-in function that removes an alarm condition from a project tag during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBRemoveAlarm	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBRemoveAlarm(strTagName,numAlarmType)
```

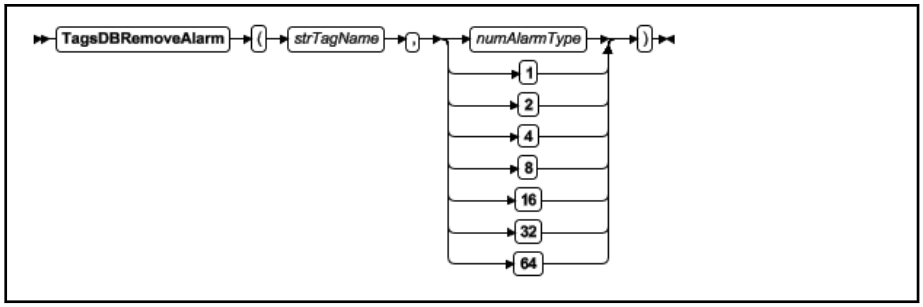


Fig. 19-69: Syntax

Inputs

strTagName

The name of the project tag from which the alarm condition will be removed.

numAlarmType

The type of alarm that you want to remove, identified by one of the following values:

Value	Alarm Type
1	HiHi
2	Hi
4	Lo
8	LoLo
16	Rate of change
32	Deviation+
64	Deviation-

Returned value

If this function is executed succesfully, it returns the value of the specified property. Otherwise, it returns one of the following possible values:

Value	Description
-4	The specified project tag (strTagName) or alarm type (numAlarmType) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and

- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.26 TagsDBRemoveClass

This function removes an existing class from the tags database during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBRemove-Class	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBRemoveClass( strClassName )
```

Inputs

strClassName

The name of the class to be removed.

Returned value

This function returns one of the following possible values:

Value	Description
-5	The specified class does not exist in the tags database.
-4	The specified class is being used (i.e., it has one or more tags associated with it).
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.27 TagsDBRemoveClassMember

This function removes an existing class member from a specified class during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBRemove-ClassMember	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBRemoveClassMember( strClassName, strMemberName )
```

Inputs

strClassName

The name of the class that contains the member to be removed.

strMemberName

The name of the class member to be removed.

Returned value

This function returns one of the following possible values:

Value	Description
-5	The specified member does not exist in the specified class.
-4	The specified class does not exist in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.28 TagsDBRemoveTag

This function removes an existing project tag from the tags database during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBRemove-Tag	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBRemoveTag( strTagName )
```

Inputs

strTagName
The name of the project tag to be removed.

Returned value

This function returns one of the following possible values:

Value	Description
-4	The specified tag does not exist in the tags database.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

Also, please note that executing this function does not completely remove the specified project tag from the tags database. That is not possible, due to how the database is maintained during run time. Instead, executing this function removes all uses of the specified tag in the project, so that they do not count against the tag limit on your runtime license, and then it reduces the tag to a single blank line in the database. Therefore, any time you execute this function to remove tags, you should stop the project as soon as it is practical to do so and then verify the project to remove the blank lines.

For more information, see Verify the project , page 50.

19.20.29 TagsDBRemoveTrend

This function removes a project tag from its trend group during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBRemove-Trend	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBRemoveTrend( strTagName )
```

Inputs

strTagName
The name of the project tag to be removed.

Returned value

This function returns one of the following possible values:

Value	Description
-4	Project tag (as specified by strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.30 TagsDBSetAlarm

TagsDBSetAlarm is a built-in function that sets the value of a property of an alarm condition during run time.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBSetAlarm	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBSetAlarm(strTagName,numAlarmType,numProperty,strOrNumVal)
```

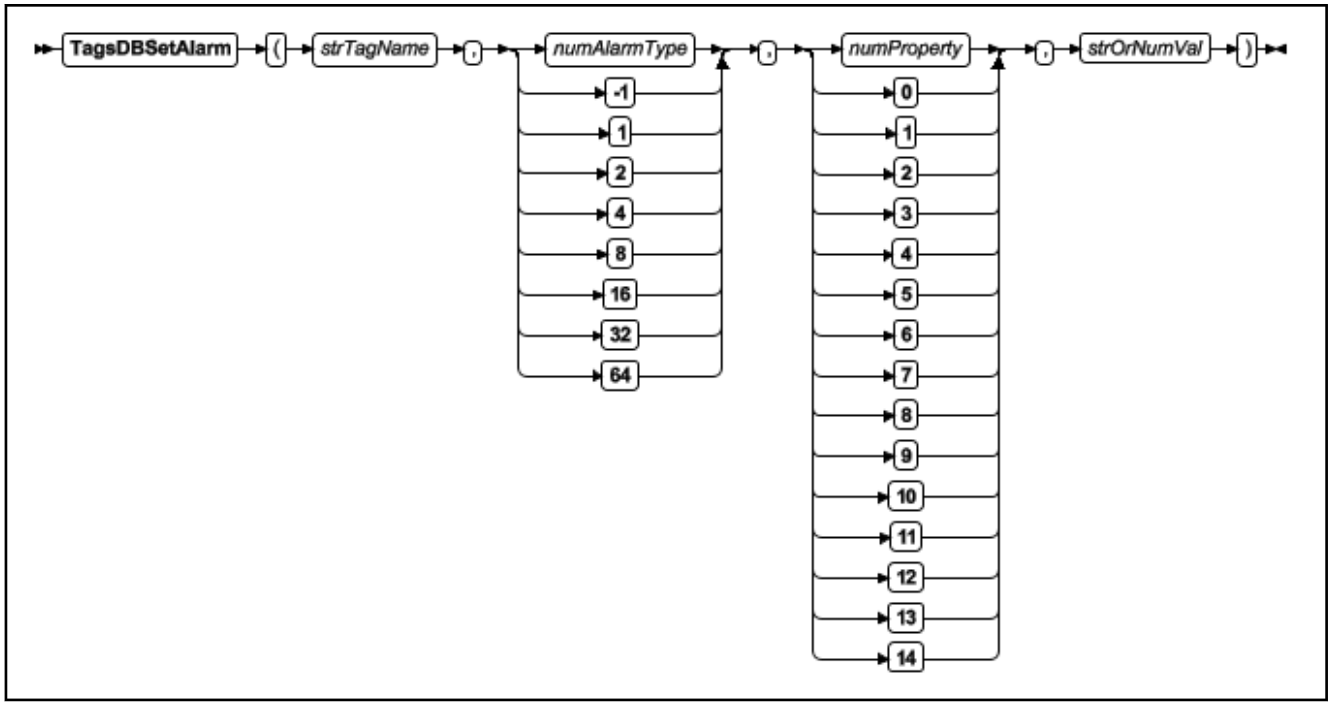


Fig. 19-70: Syntax

Inputs

strTagName

The name of the project tag on which the alarm property will be set.

To set other properties on project tags, use the function TagsDB-GetTagProperty, page 842.

numAlarmType

The type of alarm on the specified project tag, identified by one of the following values:

Value	Alarm Type
-1	General properties that apply to all alarm types on the specified project tag
1	HiHi
2	Hi
4	Lo
8	LoLo
16	Rate of change
32	Deviation+
64	Deviation-

numProperty

The alarm property that you want to set.

When setting a property of a specific alarm type (i.e., if numAlarmType is greater than 0), the property is identified by one of the following values:

Value	Property	Data Type
0	Limit value	Real
1	Message	String
2	Alarm group (or work-sheet) number	Integer
3	Priority	Integer
4	Selection	String
5	Custom field 1	String
6	Custom field 2	String
7	Custom field 3	String
8	Custom field 4	String
9	Custom field 5	String
10	Custom field 6	String
11	Custom field 7	String
12	Custom field 8	String
13	Custom field 9	String
14	Custom field 10	String

When setting a property that applies to all alarm types on the specified project tag (i.e., if numAlarmType is -1), the property is identified by one of the following values:

Value	Property	Data Type
0	Alarms Enabled	Boolean
1	Remote Ack tag	String (tag name)
2	Translation Enabled	Boolean
3	Dead Band Value	Real
4	Off	String
5	On	String
6	Ack	String
7	Deviation Setpoint	String (tag name)
8	Deviation Dead Band	Real

For more information about all of these alarm properties, see About tag properties, page 122.

strOrNumVal
The value to set to the specified property. The value must be of the appropriate data type.

Returned value

This function returns one of the following possible values:

Value	Description
-4	The specified project tag (strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

☞ A specific alarm condition, as defined by the tag name and the alarm type (e.g., the HiHi alarm on MyTag1), can be in only one alarm group at a time. Therefore, please remember that when you assign an alarm condition to an alarm group, it will be removed automatically from its previous group, if any.

Also, make sure that an alarm group with the correct group/worksheet number actually exists before you try to assign an alarm condition to it.

19.20.31 TagsDBSetTagProperty

This function sets the value of a specified tag property during run time. Tag properties are the metadata on project tags, such as tag name, array size, data type, description, scope, and so on.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBSetTagProperty	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBSetTagProperty( strTagName, numPropertyID, numOrStrPropertyValue )
```

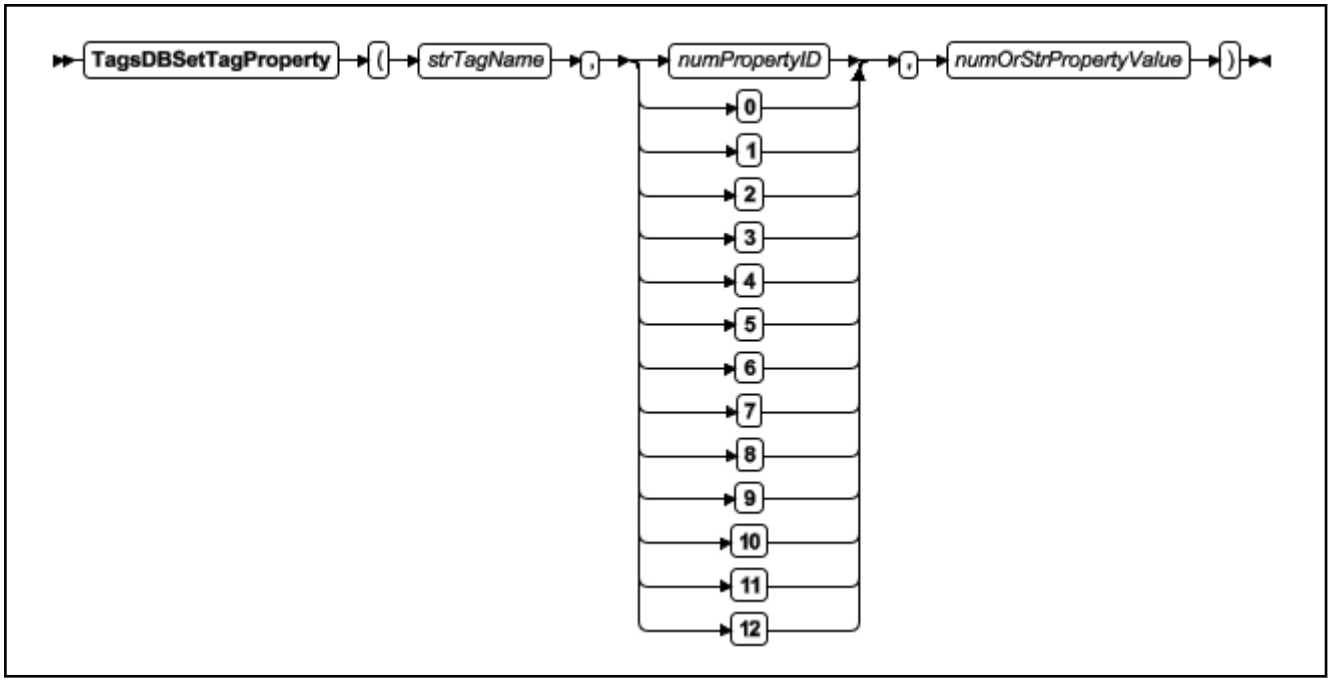


Fig. 19-71: Syntax

Inputs

strTagName

The name of the project tag on which the tag property will be set.

numPropertyID

The tag property to be set, identified by one of the following values:

Value	Property	Data Type	Same as…
0	Tag name	String	tagname->Name
1	Array size	Integer	tagname->Size
2	Data type	String ("Boolean", "Integer", "Real", "String", "classname")	-
3	Description	String	tagname->Description
4	Scope	Integer (0 for Local, 1 for Server)	-
5	Startup value	String	-
6	Minimum value	Real	tagname->Min
7	Maximum value	Real	tagname->Max
8	Engineering unit	String	tagname->Unit
9	Retentive value	Integer (0 for disabled, 1 for enabled)	-
10	Retentive properties	Integer (0 for disabled, 1 for enabled)	-
11	Dead band	Real	-
12	Smoothing	Integer (0 for disabled, 1 for enabled)	-

To set alarm conditions on project tags, use the function TagsDB-GetAlarm, page 839. Other tag properties not listed here cannot be set during run time.

numOrStrPropertyValue

The value to set to the tag property. The value must be of the appropriate data type.

Returned value

If this function is executed succesfully, then it returns the value of the specified tag property. Otherwise, it returns one of the following possible values:

Value	Description
-4	Project tag (as specified by strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.

19.20.32 TagsDBSetTrend

This function assigns a project tag to a trend group.

Function	Group	Execution	Windows	Embedded	Thin Client
TagsDBSetTrend	Tags database	synchronous	supported (see "Notes" below)	not supported	not supported

Syntax

```
TagsDBSetTrend( strTagName, numProperty, strOrNumVal )
```

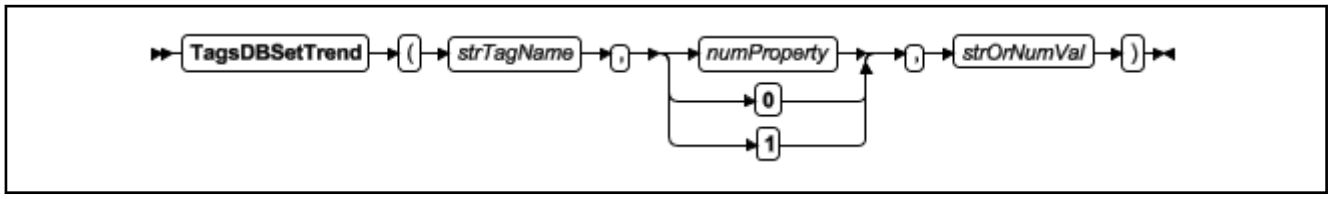


Fig. 19-72: Syntax

Inputs

strTagName

The name of the project tag to assign to the trend group.

numProperty

The specific property to be set, identified by one of the following values:

Value	Property	Data Type
0	Trend group (or worksheet) number	Integer
1	Log dead band	Real

strOrNumVal

The value to set to the specified property. The value must be of the appropriate data type.

Returned value

This function returns one of the following possible values:

Value	Description
-4	Project tag (as specified by strTagName) does not exist.
-3	Wrong parameter type or inconsistent value.
-2	Invalid number of parameters.
-1	The function TagsDBBeginEdit, page 820, was not executed successfully before this function was called.
0	Function executed successfully.

Notes

The following restrictions apply to the execution of this function:

- The function can only be executed on the project server - it cannot be called by a Graphics Script, page 930, Screen Script, page 932, or Command animation, page 246, running on a project client;
- The function TagsDBBeginEdit, page 820, must have been executed previously, in order to lock the tags database for editing; and
- The function TagsDBEndEdit, page 821, must be executed when the editing is finished, in order to release the tags database and resume normal run-time operations.

For more information about the Tags Database functions and examples of how to use them, see Editing the tags database during run time, page 124.



A specific project tag can be in only one trend group at a time. Therefore, when you assign a project tag to a trend group, it will be removed automatically from its previous group, if any.

Also, make sure that a trend group with the correct group/worksheet number actually exists before you try to assign a project tag to it.

Finally, new trends will not start recording until either the system clock changes to a new day or you restart the project.

19.21 Loop Functions

19.21.1 Loop Functions – General Information

These functions are used to implement repeating, incrementing loops within a script.

Function	Description
For … Next, page 846	implements an incrementing loop in a Math worksheet.

19.21.2 For … Next

"For" and "Next" are built-in functions that implement an incrementing loop in a Math worksheet. The section of script included in the loop begins with the For() call and ends with the Next notation. The Next notation directs back to the beginning of the loop.

Function	Execution	Windows	Embedded	Thin Client
For … Next	N/A	supported	supported	supported

Syntax

```
For( numInitialValue, numFinalValue, numStep ) … Next
```

Inputs

numInitialValue

The initial step (increment) of the loop.

numFinalValue

The final step (the final increment) of the loop.

numStep

The step (increment) of the loop.

Returned value

Returns the step on which the loop is currently running.

This type of For…Next loop can be used only in Math worksheets, page 400. It cannot be used in any Command animation, page 246, or VBScript interface, page 925.

(When you are using VBScript, you should use that language's own tools for looping. For more information, see Looping Through Code, page 963).

You must partner every `For()` call with a `Next` notation, although you may have any number of worksheet rows between them. And as shown in the example below, you must place the `Next` notation in the **Tag Name** column of the worksheet.

The loop ends when its current value equals the value of "numFinalValue". Specifically, each time the worksheet is executed and the `Next` notation is encountered, the following happens: The value of "numStep" is added to the current value of the loop; The execution returns to the `For()` call at the beginning of the loop; and the current value of the loop is compared to the value of "numFinalValue".

If the current value is less than "numFinalValue", the loop is executed again.

If the current value is greater than or equal to "numFinalValue", the loop is skipped and execution resumes with the first row after the `Next` notation.

Examples

Tag name	Expression
MyTag	For(1, 5, 1)
tagname	expression
tagname	expression
tagname	expression
Next	

Tab. 19-1: Example of For…Next loop in a Math worksheet

19.22 ODBC Functions

19.22.1 ODBC Functions – General Information

These functions are used to interact with an external database via Open Database Connectivity (ODBC).

These functions are provided to support legacy database interfaces. In most cases, it is recommended to use the newer database/ERP connections manager, page 404, and the respective functions, page 900.

Function	Description
ODBCBeginTrans, page 848	Begins a transaction with the connected data source.
ODBCBindCol, page 849	Binds a column to a tag.
ODBCCanAppend, page 850	Returns whether the database will allow you to add new records.

Function	Description
ODBCCanTransact, page 850	Transact Returns whether the database allows transactions.
ODBCCanUpdate, page 851	Update Returns whether the database can be updated.
ODBCClose, page 851	Closes a connection to the database.
ODBCCommitTrans, page 852	Commits a transaction. Call this function upon completing transactions.
ODBCDelete, page 852	Deletes the current record.
ODBCExecuteSQL, page 853	Directly executes an SQL statement.
ODBCInsert, page 853	Inserts a new record in the database.
ODBCIsBOF, page 854	Returns whether you have gone above the first record of the record set. (Call this function before scrolling from record to record.)
ODBCIsDeleted, page 854	Reports whether the current record was deleted.
ODBCIsEOF, page 855	Reports whether you have gone beyond the last record of the record set. (Call this function as you scroll from record to record.)
ODBCIsFieldNULL, page 856	Reports whether a specified field in a record set was flagged as NULL.
ODBCIsFieldNullable, page 856	Reports whether a specified field is nullable (i.e., can be set to a NULL value).
ODBCMove, page 857	Moves the current record pointer within a record set, either forward or backward.
ODBCMoveFirst, page 857	Moves to the first record within the record set.
ODBCMoveLast, page 858	Moves to the last record within the record set.
ODBCMoveNext, page 858	Moves to the next record within the record set.
ODBCMovePrev, page 859	Moves to the previous record within the record set.
ODBCOpen, page 859	Opens a connection to the database and returns a numeric handler to be used by other ODBC functions.
ODBCQuery, page 860	Retrieves the currently selected record from a database.
ODBCRollback, page 861	Reverses the changes made during a transaction.
ODBCSetFieldNULL, page 861	Flags a field data member in the record set as NULL (specifically having no value) or as non-NULL.
ODBCSetFilter, page 862	Constrains the records selected in a database.
ODBCSetSort, page 863	Sorts the records selected in a database.
ODBCUnbindCol, page 863	Unbinds a column that was previously bound using the "ODBCBindCol" function.
ODBCUpdate, page 864	Updates the current record.

19.22.2 ODBCBeginTrans

Begins a transaction with the connected data source.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCBeginTrans	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCBeginTrans(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Error beginning transaction

Examples

Tag name	Expression
Tag	ODBCBeginTrans(5)

19.22.3 ODBCBindCol

Binds a column to a tag.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCBindCol	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCBindCol( numHandler, strColName, strColType, strTagName )
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strColName

The database column name.

strColType

The SQL data type (of the following list):

- SQL_BIT
- SQL_TINYINT
- SQL_LONGVARCHAR
- SQL_CHAR
- SQL_VARCHAR
- SQL_DECIMAL
- SQL_NUMERIC
- SQL_DATE
- SQL_TIME
- SQL_TIMESTAMP
- SQL_DOUBLE
- SQL_REAL
- SQL_SMALLINT
- SQL_INTEGER

strTagName

The name of the tag to be bound to the column.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Invalid parameter type
3	One of the parameters has an empty string
4	ColType contains an invalid type

☞ Every time after a column is bound, the ODBCQuery, page 860 function has to be called.

Examples

Tag name	Expression
Tag	ODBCBindCol(5, "OrderDate", "SQL_DATE", "Order_Date")

For further information, refer to: ODBCUnbindCol(), page 863

19.22.4 ODBCCanAppend

Checks whether the database enables adding new records.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCCanAp-pend	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCCanAppend(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Database does not enable adding new records
Not zero	Database does enable adding new records

Examples

Tag name	Expression
Tag	ODBCCanAppend(5)

19.22.5 ODBCCanTransact

Checks whether the database enables transactions.

Function	Group	Execution	Windows	Embedded	Thin Client
Avg	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCCanTransact(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Database does not enable transactions
Not zero	Database does enable transactions

Examples

Tag name	Expression
Tag	ODBCCanTransact(2)

19.22.6 ODBCCanUpdate

Checks whether the database can be updated.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCCanUpdate	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCCanUpdate(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Database does not enable updates
Not zero	Database does enable updates

Examples

Tag name	Expression
Tag	ODBCCanUpdate(6)

19.22.7 ODBCClose

Closes a connection to the database.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCClose	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCClose(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler

Examples

Tag name	Expression
Tag	ODBCClose(5)

19.22.8 ODBCCommitTrans

Commits a transaction. Call this function after completing transactions.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCCommit-Trans	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCCommitTrans(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Error committing transaction

Examples

Tag name	Expression
Tag	ODBCCommitTrans(1)

19.22.9 ODBCDelete

Deletes the current record.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCDelete	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCDelete(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Delete error

☞ After successful deletion, one of the "move" functions has to be called explicitly (i.e., ODBCMove, page 857, ODBCMoveFirst, page 857, ODBCMoveLast, page 858, ODBCMoveNext, page 858, ODBCMovePrev, page 859) to exit the deleted record.

Examples

Tag name	Expression
Tag	ODBCDelete(5)

19.22.10 ODBCExecuteSQL

Directly executes an SQL statement.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCExecu-teSQL	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCExecuteSQL( numHandler, strSqlCommand )
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strSqlCommand

A valid SQL statement.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Invalid parameter
4	Error executing SQL command

☞ This function does not return any records, regardless of the statement that is executed.

Examples

Tag name	Expression
Tag	ODBCExecuteSQL(3, '')
Tag	ODBCExecuteSQL(4, '')

19.22.11 ODBCInsert

Inserts a new record into the database.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCInsert	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCInsert(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Insert error

This function uses the values of the tags linked by the ODBCBindCol, page 849 function to create the new record.

Examples

Tag name	Expression
Tag	ODBCInsert(7)

19.22.12 ODBCIsBOF

Returns whether the first record of the record set was exceeded. (Call this function before browsing through the records.)

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCIsBOF	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCIsBOF(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Record found
Not zero	Record set contains no records or it is moved backward, exceeding the first record

This function can also be used in combination with the ODBCIsEOF, page 855 function to determine whether the record set contains any records or is empty. Immediately after calling the ODBCQuery, page 860 function and if the record set contains no records, **ODBCIsBOF** returns a non-zero value. If a record set with at least one record is opened, the first record is the current record and **ODBCIsBOF** returns zero ("0"). If the first record is the current record, and ODBCMovePrev, page 859 is called, **ODBCIsBOF** returns a non-zero value.

Examples

Tag name	Expression
Tag	ODBCIsBOF(1)

19.22.13 ODBCIsDeleted

Reports whether the current record was deleted.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCIsDeleted	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCIsDeleted(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Record set is not positioned on a deleted record.
Not zero	Record set is positioned on a deleted record

If it is moved to a record and this function returns a non-zero value, it has to be switched to another record before any other operations can be performed.

Examples

Tag name	Expression
Tag	ODBCIsDeleted(8)

19.22.14 ODBCIsEOF

Reports whether the last record of the record set was exceeded. (Call this function on browsing through the records.)

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCIsEOF	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCIsEOF(numHandler)
```

Inputs

numHandler
The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Record found
Not zero	Record set contains no records or it is moved forward, exceeding the last record.

This function can be used in combination with the ODBCIsBOF, page 854 function to determine whether the record set contains any records or is empty. Immediately after calling the ODBCQuery, page 860 function and if the record set contains no records, **ODBCIsBOF** returns a non-zero value. If a record set with at least one record is opened, the first record is the current record and **ODBCIsEOF** returns a zero ("0"). If the last record is the current record and ODBCMoveNext, page 858 is called, **ODBCIsEOF** returns a non-zero value.

Examples

Tag name	Expression
Tag	ODBCIsEOF(5)

19.22.15 ODBCIsFieldNULL

Reports whether a specified field in a record set was flagged as "ZERO".

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCIsField-NULL	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCIsFieldNULL(numHandler,strColName)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strColName

The column name.

Returned value

Value	Description
0	The specified field is not flagged as Null
Not zero	The specified field is flagged as Null

Examples

Tag name	Expression
Tag	ODBCIsFieldNULL(7, "CustomerName")
Tag	ODBCIsFieldNULL(3, "CompanyName")

19.22.16 ODBCIsFieldNullable

Checks whether a specified field is nullable (i.e., can be set to a "NULL" value).

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCIsFieldNul-lable	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCIsFieldNullable(numHandler,strColName)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strColName

The column name.

Returned value

Value	Description
0	The specified field is not flagged as Nullable
Not zero	The specified field is flagged as Nullable

Examples

Tag name	Expression
Tag	ODBCIsFieldNullable(1, "Price")
Tag	ODBCIsFieldNullable(1, "Model")

19.22.17 ODBCMove

Moves the current record pointer either forward or backward within a record set.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCMove	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCMove(numHandler,numOffset)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

numOffset

The number of rows to move forward or backward:

- Positive values move forward towards the end of the record set.
- Negative values move backward towards the beginning of the record set.
- A value of "0" updates the current record.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Move error

Examples

Tag name	Expression
Tag	ODBCMove(2, 3)
Tag	ODBCMove(8, 2)

19.22.18 ODBCMoveFirst

Moves to the first record within the record set.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCMoveFirst	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCMoveFirst(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Move error

Examples

Tag name	Expression
Tag	ODBCMoveFirst(4)

19.22.19 ODBCMoveLast

Moves the pointer to the last record within the record set.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCMoveLast	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCMoveLast(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Move error

Examples

Tag name	Expression
Tag	ODBCMoveLast(7)

19.22.20 ODBCMoveNext

Moves the pointer to the next record within the record set.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCMoveNext	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCMoveNext(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	End of record set reached
4	Move error

Examples

Tag name	Expression
Tag	ODBCMoveNext(9)

19.22.21 ODBCMovePrev

Moves the pointer to the previous record within the record set.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCMovePrev	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCMovePrev(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Beginning of record set reached
4	Move error

Examples

Tag name	Expression
Tag	ODBCMovePrev(2)

19.22.22 ODBCOpen

Opens a connection to the database and returns a numeric handler to be used by other ODBC functions.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCOpen	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCOpen( strDsn, strUser, strPassw, strTable, strFilter, strSort )
```

Inputs

strDsn

The name of the data source.

strUser

The user name.

strPassw

The password.

strTable

The name of the database table.

strFilter

The SQL **WHERE** clause.

strSort

The SQL **ORDER BY** clause.

Returned value

Value	Description
n	On successful execution, returns the handler to identify the database
−1	Invalid parameter
−2	DSN or TableName contain an empty string



This function does not read or write any data. It simply creates a handler to manipulate the database. The columns have to be bound by using the ODBCBindCol, page 849 function and the ODBCQuery, page 860 function has to be called to retrieve the first record.

Examples

Tag name	Expression
Tag	ODBCOpen("MyDSNFile", "Alex", "", "Table1", "Name='Mayer'", "Name ASC")
Tag	ODBCOpen("DSNFileName", "Robert", "Robot", "Table1", "", "")

19.22.23 ODBCQuery

Retrieves the currently selected record from a database.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCQuery	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCQuery(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	No columns bound
3	Cannot open database
4	Cannot restart database
5	Query error

If column binding, page 849 or filter, page 862 and sorting, page 863 is changed, this function has to be called again.

Examples

Tag name	Expression
Tag	ODBCQuery(6)

19.22.24 ODBCRollBack

Discards the changes made during a transaction.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCRollBack	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCRollBack(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Error discarding the changes

Examples

Tag name	Expression
Tag	ODBCRollback(4)

19.22.25 ODBCSetFieldNULL

Flags a field data member in the record set as "NULL" (specifically without value) or as "not NULL".

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCSetField-NULL	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCSetFieldNULL( numHandler, strColName, numValue )
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strColName

The column name.

numValue

A numeric tag that specifies the field data as NULL if 0 and not NULL if not zero.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Invalid parameter
4	Invalid column name

Examples

Tag name	Expression
Tag	ODBCSetFieldNULL(2, "Price", 1)
Tag	ODBCSetFieldNULL(4, "CompanyName", 0)

19.22.26 ODBCSetFilter

Limits the records selected in a database.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCSetFilter	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCSetSort(numHandler,strSort)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strFilter

The SQL **WHERE** clause.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Invalid parameter

This function can be used for selecting a subordinate set of records, e.g., "all salespersons based in California" (**"state = 'CA'"**). After calling this function, ODBCQuery, page 860 has to be called.

Examples

Tag name	Expression
Tag	ODBCSetFilter(3, "Name='Morgan'")

19.22.27 ODBCSetSort

Sorts the records selected in a database.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCSetSort	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCSetSort(numHandler,strSort)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strSort

The SQL **ORDER BY** clause.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Invalid parameter type

This property can be used to sort records in one or more columns. After calling this function, ODBCQuery, page 860 has to be called.

Examples

Tag name	Expression
Tag	ODBCSetSort(5, "Name DESC")

19.22.28 ODBCUnbindCol

Unbinds a column that was previously linked by the "ODBCBindCol" function.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCUnbindCol	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCUnbindCol(numHandler,strColName)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

strColName

The column name.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Invalid parameter type
3	Column not bound

Examples

Tag name	Expression
Tag	ODBCUnbindCol(7, "Name")

19.22.29 ODBCUpdate

Updates the current record.

Function	Group	Execution	Windows	Embedded	Thin Client
ODBCUpdate	ODBC	synchronous	supported	not supported	executed on the server

Syntax

```
ODBCUpdate(numHandler)
```

Inputs

numHandler

The handler returned by the ODBCOpen, page 859 function.

Returned value

Value	Description
0	Successful
1	Invalid handler
2	Database not open
3	Update error



This function uses the values of the tags linked by the ODBC-BindCol, page 849 function to update the current record.

Examples

Tag name	Expression
Tag	ODBCUpdate(1)

19.23 Email Functions

19.23.1 Email Functions – General Information

These functions are used to configure and send emails from within a project.

Function	Description
CnfEmail, page 865	Configures the email settings used by other features in the project that can send email, such as Alarm worksheets and the "SendEmail" and "SendEmailExt" functions.
GetStatusSendEmailExt, page 867	Returns status of the last email sent using the "SendEmailExt" function.
SendEmail, page 868	Sends an email message.
SendEmailExt, page 869	Sends email messages with attached files.

19.23.2 CnfEmail

This function configures the email settings used by other features in the project that can send emails. For example, alarm worksheets and the **SendEmail** and **SendEmailExt** functions.

Function	Execution	Windows	Embedded	Thin Client
CnfEmail	synchronous	supported	supported, except for encryption	supported

Syntax

```
CnfEmail(strSMTP, strFrom, strPOP3, strUser, strPassword, optNumTimeout, optNumAuthType, optStrSMTPUser,
         optStrSMTPPassword)

CnfEmail(strSMTP,strFrom,strPOP3,strUser,strPassword{ | ,optNumTimeout{ | ,{ optNumAuthType | 1 | 2 }
         { | ,optStrSMTPUser,optStrSMTPPassword } } })
```

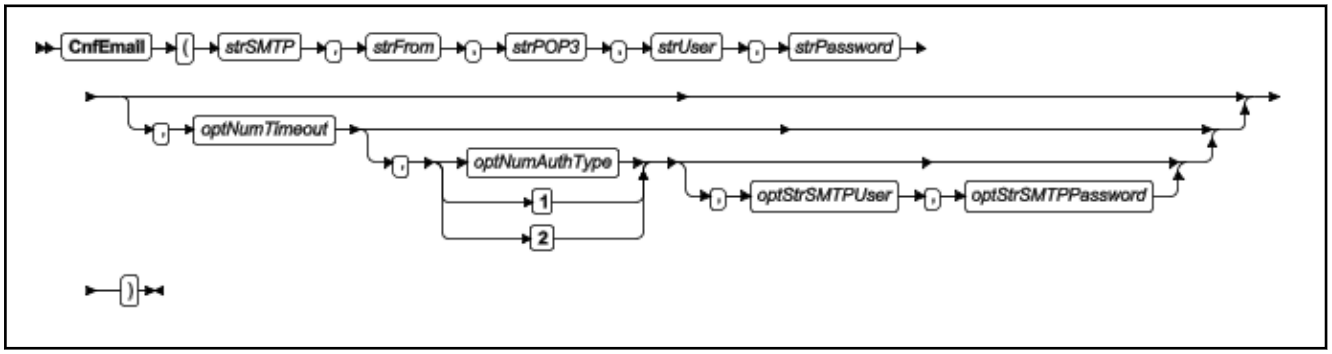


Fig. 19-73: Syntax

Inputs

strSMTP

The hostname or IP address of the outgoing email server, which is also referred to as SMTP server. A port number can be specified if the server does not use any of the standard SMTP ports.

☞ On Windows CE devices, an IP address has to be specified.

strFrom

The email address to send and receive emails. This is to be a valid address on the POP3 server (see **strPOP3** in the following).

strPOP3

The hostname or IP address of the incoming email server, also referred to as the POP3 server. A port number can be specified if the server does not use any of the standard POP3 ports.

☞ On Windows CE devices, an IP address has to be specified.

strUser

The user name to be used to log onto the POP3 server.

strPassword

The password to be used to log onto the POP3 server.

optNumTimeout

The timeout limit (in seconds) for sending emails. If no response is received from the SMTP server within this period of time, the operation is aborted.

This is an optional parameter. If no timeout is specified, the project keeps trying to send until a response is received. However, a timeout limit is to be specified to ensure that the project does not freeze.

optNumAuthType

optStrSMTPUser

optStrSMTPPassword

By default, SMTP servers do not require authentication for outgoing emails. If the server does require authentication, set **optNumAuthType** to "1" (unencrypted) or "2" (encrypted via TLS/SSL) and specify the user name and password. (If the SMTP and POP3 user name and password are identical, **optStrSMTPUser** and **optStrSMTPPassword** can be skipped. In this case, the project automatically uses the values from **strUser** and **strPassword**.)

☞ Encryption via TLS/SSL is not supported in projects running on Windows CE target systems.

Returned value

Value	Description
0	Successful
1	Invalid format for strSMTP
2	Invalid format for strFrom
3	Invalid format for strPOP3
4	Invalid format for strUser
5	Invalid format for strPassword
6	Invalid format for optNumTimeout
7	Incorrect number of parameters
8	Error getting host IP address (invalid POP3 server)
9	Error connecting to POP3 server

Value	Description
10	Error sending user name
11	Error sending password
12	SMTP server does not support selected authentication mode
13	Invalid SMTP user name
14	Authentication failed

The email configuration created by this function works only within the Windows process where the function was called.

For example, if a button object is placed in a screen and the object is set to call this function when it is pressed, the resulting email configuration works only on the client station where the screen is displayed and the button is pressed.

It does not work on any other client station nor on the server station as the project viewer running on the client station only exchanges data (i.e., changes in tag values) with the data server running on the server station. A server cannot directly call functions on another server. Only triggers can be used to call functions on other servers.

This also applies if the client station and the server station are the same physical device as project viewer and data server are two separate processes in Windows.

If an email configuration is to apply to the Background tasks of the project – e.g., to be able to send emails when alarms become active –, either the "Email Settings" dialog has to be used to configure default settings for the entire project OR this function, for example, with the startup script, page 402, a script group, page 401 or a math worksheet, page 400.

Examples *CnfEmail - Examples*

```
CnfEmail("smtp.company.com","Robert@company.com","pop.company.com","RobertH","Shades556",100)

CnfEmail("smtp.company.com:4455","Robert@company.com","pop.company.com:9900","RobertH","Shades556",5,1)

CnfEmail("195.11.22.33:4455","Robert@company.com","195.66.77.88:9900","RobertH","Shades556",5,2,"JohnS",
         "abcd1234")
```

19.23.3 GetStatusSendEmailExt

Returns the status of the last email sent using the "SendEmailExt" function.

Function	Execution	Windows	Embedded	Thin Client
GetStatusSendEmailExt	synchronous	supported	supported	supported

Syntax

```
GetStatusSendEmailExt({ | optTagName })
```

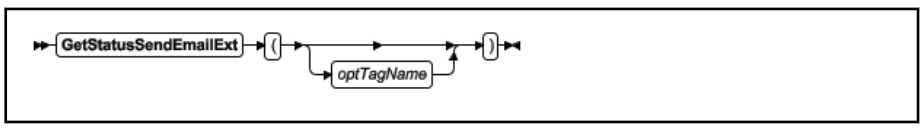


Fig. 19-74: Syntax

Inputs

optTagName

"Optional" tag causing the function to update its return value. This parameter is optional. However, it has to be used if this function is configured for an object animation, page 246 (e.g., Text Data Link, Position).

Returned value

Value	Description
−2	Incorrect **INDMail.DLL** library version
−1	Corrupted **INDMail.DLL** library
0	**SendEmailExt** function is not executed
1	Still sending last email. Cannot execute the **SendEmailExt** function
2	Last email was sent successfully. The **SendEmailExt** function can be executed again
3	There was an error sending the last email. Execute the **SendEmailExt** function again

Examples

Tag name	Expression
Tag	GetStatusSendEmailExt(Second)
Tag	GetStatusSendEmailExt()

19.23.4 SendEmail

This function sends an email to a receiver (without attached files).

Function	Execution	Windows	Embedded	Thin Client
SendEmail	synchronous	supported	supported	supported

Syntax

```
SendEmail( strSubject, strMessage, strTo )
```

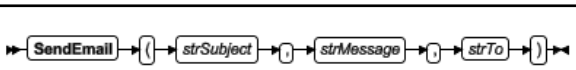


Fig. 19-75: Syntax

Inputs

strSubject

The subject of the email.

strMessage

The message text of the email, up to 255 characters long.

strTo

The email address of the recipient.

Returned value

Value	Description
0	Successful
1	Invalid format for strSubject

Value	Description
2	Invalid format for strMessage
3	Invalid format for strTo
4	Incorrect number of parameters
5	Start socket error
6	Error getting host IP address (i.e., invalid SMTP server)
7	Error connecting to SMTP server
8	Error sending HELO command (i.e., initialization)
9	Error sending MAIL command (i.e., the "From" address)
10	Error sending RCPT command (i.e., the "To" address)
11	Error sending DATA (i.e., message test)
12	Error sending SMTP authentication command
13	Invalid user name
14	Invalid password

☞ Before any email can be sent, the email settings of the project have to be configured first, either by using the **Email Settings** dialog or by calling the **CnfEmail** function. Incorrect configurations are indicated by the returned value (see above).

The function **SendEmail** cannot be used when encryption via TLS/SSL is enabled or to send an email that contains Unicode characters. For this purpose, the **SendEmailExt** function is available.

Examples

SendEmail - Examples

```
SendEmail( "Hi!", "How are you?", "rogers@pnd.net" )

SendEmail( statusSummary, statusDetail, adminAddress )
```

19.23.5 SendEmailExt

Sends an email with attached files.

Function	Execution	Windows	Embedded	Thin Client
SendEmailExt	asynchronous	supported	supported	supported

Syntax

```
SendEMailExt( strSubject, strMessage, strTo, optStrCc, optStrBcc, optStrFile1, … , optStrFileN )
```

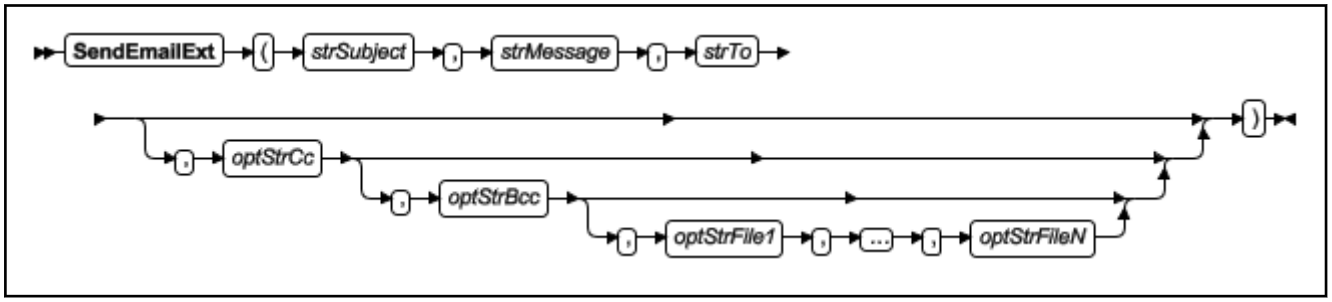


Fig. 19-76: Syntax

Inputs

strSubject

The email subject (up to 255 characters).

strMessage

The email message (up to 255 characters).

strTo

The address of the recipient. More than one recipient can be specified. The addresses are separated by semicolons (;).

optStrCc

The recipient addresses for message copies. More than one recipient can be specified. The addresses are separated by semicolons (;).

This parameter is optional. If the following parameters are to be used a null string ("") can be specified.

optStrBcc

The recipient addresses for message "blind copies". More than one recipient can be specified. The addresses are separated by semicolons (;).

This parameter is optional. If the following parameters are to be used a null string ("") can be specified.

optStrFile (1-N)

Complete file paths and names of file attachments.

Returned value

Value	Description
−4	Some of the attached files could not be found
−3	Incorrect number of parameters (at least three parameters are required)
−2	Incorrect **INDMail.DLL** library version
−1	Corrupted **INDMail.DLL** library
0	Successful
1	The function cannot be executed as the last email has not been sent yet
2	Internal error



Before any email can be sent, the CnfEmail, page 865 function has to be used to configure the email settings. Incorrect configurations are indicated by the returned value (see above).

Examples

Tag name	Expression
Tag	SendEmailExt("Subject", "Message", "Sam@universe.com", "", "", "C:\Projects eport.txt")
Tag	SendEmailExt("Subject", "Message", "David@Ohio.net", "Ted@Austin.com", "Bart@Springfield.gov", "C:\TechRef51.doc")

19.24 Dial-up Functions

19.24.1 Dial-up Functions – General Information

These functions are used to configure computer modems (if applicable) and establish dial-up connections to other computers.



This function is not supported by Windows 7.

Function	Description
DialError, page 871	Returns the error codes regarding each connection.
DialGetClientIP, page 877	DialGetClientIP is a built-in function that gets the IP address of a Remote Access Service (RAS) client station.
DialGetServerIP, page 879	DialGetServerIP is a built-in scripting function that gets the IP address of a Remote Access Service (RAS) server station.
DialStatus, page 880	Returns the status of the dial-up connection.
DialUp, page 882	Establishes a dial-up connection.
DialUpToCE, page 884	Runs the DialUpToCE program, which sends the information necessary to CE-RasSvr.exe calls back to the Server.
FindAllDevices, page 885	Returns the list of all the available modems and direct connection interfaces (COM ports) in the local station.
FindModem, page 886	Returns the list of all available modems in the local station.
HangUp, page 886	Hangs-up a dial-up connection.
PhoneDialUp, page 887	Dials to a phone number using Telephony Application Program Interface (TAPI).
PhoneDisableListen, page 888	Stops listening to the modem for incoming calls.
PhoneEnableListen, page 888	Resumes listening to the modem for incoming calls.
PhoneHangUp, page 889	Hangs up a dial-up connection previously established with the PhoneDialUp function.
PhoneStatus, page 890	Checks the status of the current connections.

19.24.2 DialError

Returns the error codes regarding each connection.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
DialError	synchronous	supported (see note)	supported	supported

Syntax

```
DialError(numType, strPhonebookEntryOrModem, "optError", optRefresh)
```

Inputs

numType

A numeric parameter specifying the content for the **strPhonebookEntryorModem** parameter.

- 0: Phonebook name
- 1: Modem name
- 2: Direct connection name

strPhonebookEntryOrModem

The name of the phonebook, modem or direct connection to establish the connection. The **numType** parameter specifies which of these methods is used.

optError

"Optional" The name of the String tag receiving the error message.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the function tries to get the value of the specified tag.

optRefresh

"Optional" tag causing the function to update its return value. This parameter is optional. However, it has to be configured if this function is to be used for an object animation, page 246 (e.g., Text Data Link, Position).

Returned value

Value	Description
0	OK
−1	Error: INDRas.DLL not found
−2	Error: INDRas.DLL damaged
−3	Error: invalid number of parameters (minimum=2)
-4	Invalid value for the **numType** parameter (0 or 1)
-5	Phonebook or modem does not exist
600	An operation is pending
601	The port handle is invalid
602	The port is already open
603	Insufficient caller buffer
604	Incorrect information specified
605	Cannot set port information
606	The port is not connected
607	The event is invalid
608	The device does not exist
609	The device type does not exist
610	Invalid buffer

Value	Description
611	The route is not available
612	The route is not allocated
613	Invalid compression specified
614	Out of buffer
615	The port could not be found
616	An asynchronous request is pending
617	The port or device is already disconnecting
618	The port is not open
619	The port is disconnected
620	There are no endpoints
621	Cannot open the phonebook file
622	Cannot load the phonebook file
623	Cannot open the phonebook file
624	Cannot write the phonebook file
625	Invalid information found in the phonebook file
626	Cannot load string
627	Cannot find key
628	The port was disconnected
629	The data link was terminated by the remote device
630	The port was disconnected due to hardware failure
631	The port was disconnected by the user
632	Incorrect structure size
633	The port is already in use or not configured for remote access dial out
634	The computer cannot be registered on the remote network
635	Unknown error
636	An incorrect device is connected to the port
637	The string could not be converted
638	The request has timed out
639	No asynchronous net available
640	A NetBIOS error occurred
641	The server cannot allocate NetBIOS resources needed to support the client
642	One of the NetBIOS names is already registered on the remote network
643	A network adapter at the server failed
644	No network message pop-ups are received.
645	Internal authentication error

Value	Description
646	The account is not permitted to log on at this time of day
647	The account is disabled
648	The password has expired
649	The account does not have remote access rights
650	The Remote Access server is not responding
651	The modem (or other connecting device) has reported an error
652	Unrecognized response from the device
653	A macro required by the device was not found in the device **.INF** file section
654	A command or response in the device **.INF** file section refers to an undefined macro
655	The <message macro could not be found in the device **.INF** file section.
656	The <defaultoff macro in the device **.INF** file section contains an undefined macro
657	The device **.INF** file could not be opened
658	The device name in the device **.INF** or media **.INI** file is too long
659	The media **.INF** file refers to an unknown device name
660	The device **.INF** file contains no responses for the command
661	The device**.INF** file is missing a command
662	Attempted to set a macro not listed in the device **.INF** file section
663	The media **.INI** file refers to an unknown device type
664	Cannot allocate memory
665	The port is not configured for remote access
666	The modem (or other connecting device) does not work
667	Cannot read the media **.INI** file
668	The connection was disconnected
669	Invalid parameter used in the media **.INI** file
670	Cannot read the section name from the media **.INI** file
671	Cannot read the device type from the media **.INI** file
672	Cannot read the device name from the media **.INI** file
673	Cannot read the usage from the media **.INI** file
674	Cannot read the maximum connection BPS rate from the media **.INI** file
675	Cannot read the maximum carrier BPS rate from the media **.INI** file
676	The line is busy
677	The call was answered by a person instead of a modem
678	No answer
679	No carrier found

Value	Description
680	No dial tone
681	General error reported by the device
682	ERROR_WRITING_SECTIONNAME
683	ERROR_WRITING_DEVICETYPE
684	ERROR_WRITING_DEVICENAME
685	ERROR_WRITING_MAXCONNECTBPS
686	ERROR_WRITING_MAXCARRIERBPS
687	ERROR_WRITING_USAGE
688	ERROR_WRITING_DEFAULTOFF
689	ERROR_READING_DEFAULTOFF
690	ERROR_EMPTY_INI_FILE
691	Access denied because of invalid user name and/or password on the domain
692	Hardware failure in port or attached device
693	ERROR_NOT_BINARY_MACRO
694	ERROR_DCB_NOT_FOUND
695	ERROR_STATE_MACHINES_NOT_STARTED
696	ERROR_STATE_MACHINES_ALREADY_STARTED
697	ERROR_PARTIAL_RESPONSE_LOOPING
698	A response key name in the device **.INF** file has an unexpected format
699	The device response caused a buffer overflow
700	The expanded command in the device **.INF** file is too long
701	The device moved to a BPS rate not supported by the COM driver
702	Unexpected device response received
703	The Application does not allow user interaction. The connection requires user interaction to run successfully
704	ERROR_BAD_CALLBACK_NUMBER
705	ERROR_INVALID_AUTH_STATE
706	ERROR_WRITING_INITBPS
707	X.25 diagnostic indication
708	The account has expired
709	Error changing password on the domain. The password may be too short or too long or identical to a previously used password
710	Serial overrun errors were detected while communicating with the modem
711	RasMan initialization error. Check the event log
712	Biplex port initializing. Wait a few seconds and dial up again

Value	Description
713	No active ISDN lines are available
714	No ISDN channels are available for the call
715	Too many errors occurred due to poor phone line quality
716	The remote access IP configuration cannot be used
717	No IP addresses are available in the static pool of remote access IP addresses
718	Timeout waiting for a valid response from the remote PPP peer
719	PPP terminated by remote device
720	No PPP control protocols configured
721	Remote PPP peer is not responding
722	The PPP package is invalid
723	The phone number including area code and extension number is too long
724	The IPX protocol cannot dial out on the port because the device is an IPX router
725	The IPX protocol cannot dial up on the port because the IPX router is not installed
726	The IPX protocol cannot be used for dial out on more than one port at a time
727	Cannot access **TCPCFG.DLL**
728	IP adapter linked to remote access cannot be found
729	SLIP cannot be used unless the IP protocol is installed
730	Computer registration is not complete
731	The protocol is not configured
732	The PPP negotiation is not converging
733	The PPP control protocol for this network protocol is not available on the server
734	The PPP link control protocol was terminated
735	The requested address was rejected by the server
736	The remote computer terminated the control protocol
737	Loopback detected
738	The server has not assigned an address
739	The authentication protocol required by the remote server cannot use the encrypted password reconnection of Windows NT, explicitly requesting password entry
740	Invalid TAPI configuration
741	The local computer does not support the required encryption type
742	The remote computer does not support the required encryption type
743	The remote computer requires encryption

Value	Description
744	The IPX network number assigned by the remote server cannot be used. Check the event log
745	ERROR_INVALID_SMM
746	ERROR_SMM_UNINITIALIZED
747	ERROR_NO_MAC_FOR_PORT
748	ERROR_SMM_TIMEOUT
749	ERROR_BAD_PHONE_NUMBER
750	ERROR_WRONG_MODULE
751	Invalid callback number. Only the characters "0" to "9", "T", "P", "W", "(", ")", "-", "@" and space are allowed in the number
752	A syntax error occurred while processing a script
753	The connection could not be disconnected because it was created by the Multi-Protocol Router

Examples

Tag name	Expression
Tag	DialError(0, "Office DialUp")
Tag	DialError(1, "USRobotics_SportsterFaxModem", "StatusMessage", second)
Tag	DialError(2, "DirectDial", "DialupError")

For further information, refer to: FindModem(), page 886

19.24.3 DialGetClientIP

Gets the IP address of a Remote Access Service (RAS) client station.



This function is not supported by Windows 7.



This function is not supported by Windows 8.

Function	Execution	Windows	Embedded	Thin Client
DialGetClientIP	synchronous	supported (see note)	supported	supported

Syntax

```
DialGetClientIP({ numType | 0 | 1 | 2 },strPhoneBookOrModem,"tagClientIP"{ | ,optTagRefresh })
```

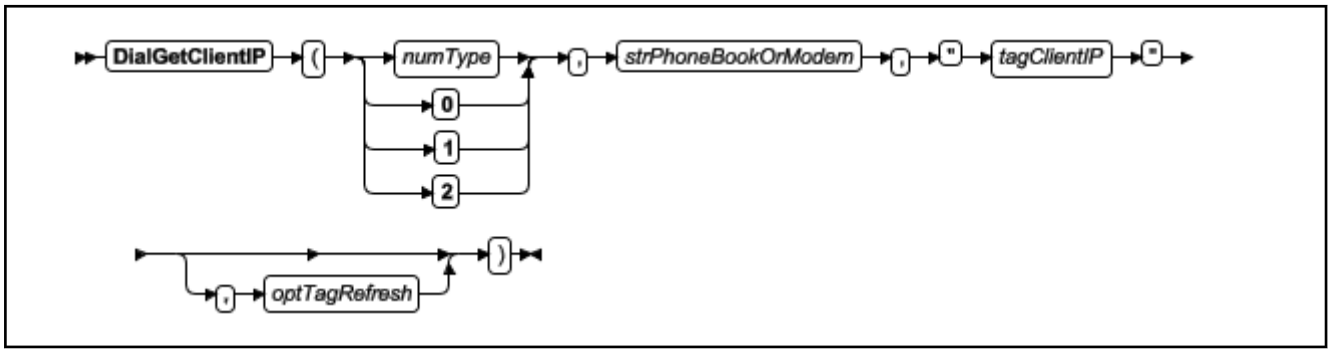


Fig. 19-77: Syntax

Inputs

numType

The type of information specified by **strPhoneBookOrModem**:

"0" is a phonebook name,

"1" is a modem name and

"2" is a direct connection name.

strPhoneBookOrModem

The name of the phonebook, modem or direct connection to establish the connection.

tagClientIP

The name of a String tag that is to receive the IP address.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the function tries to get the value of the specified tag.

optTagRefresh

The name of a tag that triggers the function to update the returned value every time the value of the tag is changed.

This parameter is optional. However, it has to be specified if the function is to be used in an object animation, e.g., Text Data Link or Position.

Returned value

This function returns the following possible values:

Value	Description
-5	**DialGetClientIP** could not be found in **IndRAS.dll**
-4	Invalid type. Check **numType**
-3	Incorrect number of parameters
-2	**DialStatus** could not be found in **IndRAS.dll**
-1	**IndRAS.dll** was not loaded
n	Status code returned by **DialStatus**

The IP address is written to the tag specified by **tagClientIP**.

☞ This function has to be executed on the RAS client station, **not** on the server station. (Essentially, the station is assigned its own IP address.)

Examples

Use the Phonebook Name "Office DialUp" and write the resulting IP address to ClientIPTag:

```
DialGetClientIP(0,"Office DialUp","ClientIPTag")
```

Use the Modem Name "USRobotics_SportsterFaxModem", write the resulting IP address to ClientIPAddress, and refresh every second (that is, every time the system tag Second changes):

```
DialGetClientIP(1,"USRobotics_SportsterFaxModem","ClientIPAddress",Second)
```

Use the Direct Connection Name "DirectDial" and write the resulting IP address to IPAdd:

```
DialGetClientIP(2,"DirectDial","IPAdd")
```

19.24.4 DialGetServerIP

"DialGetServerIP" is an internal scripting function that gets the IP address of a Remote Access Service (RAS) server station.

This function is not supported by Windows 7.

This function is not supported by Windows 8.

Function	Execution	Windows	Embedded	Thin Client
DialGetServerIP	synchronous	supported (see note)	not supported	supported

Syntax

```
DialGetServerIP({ numType | 0 | 1 | 2 },strPhoneBookOrModem,"tagServerIP"{ | ,optTagRefresh })
```

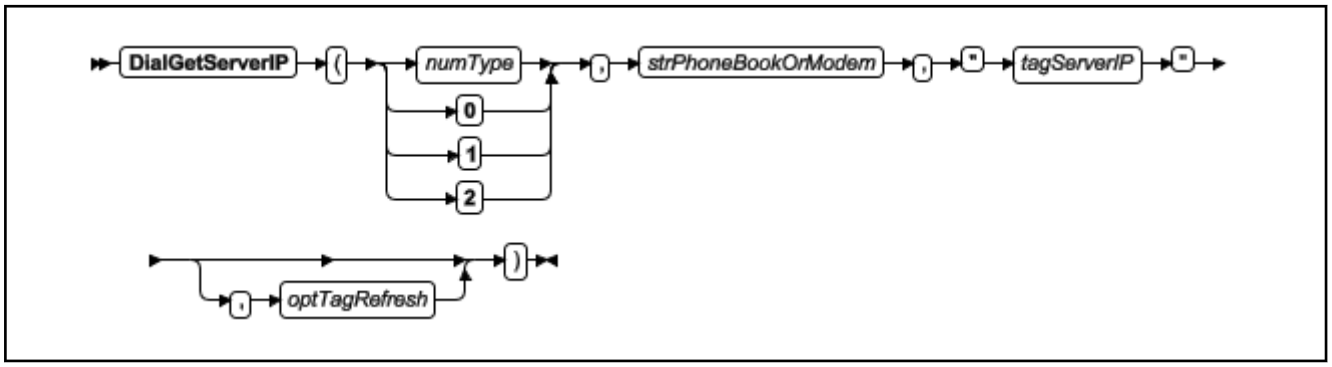


Fig. 19-78: Syntax

Inputs

numType

The type of information specified by **strPhoneBookOrModem**:

"0" is a phonebook name,

"1" is a modem name and

"2" is a direct connection name.

strPhoneBookOrModem

The name of the phonebook, modem or direct connection to establish the connection.

tagServerIP

The name of a String tag that is to receive the IP address.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

optTagRefresh

The name of a tag that triggers the function to update the returned value every time the value of the tag is changed.

This parameter is optional. However, it has to be specified if the function is to be used in an object animation, e.g., Text Data Link or Position.

Returned value This function returns the following possible values:

Value	Description
-5	**DialGetServerIP** could not be found in **IndRAS.dll**
-4	Invalid type. Check **numType**
-3	Incorrect number of parameters
-2	**DialStatus** could not be found in **IndRAS.dll**
-1	**IndRAS.dll** was not loaded
n	Status code returned by **DialStatus**

The IP address is written to the tag specified by **tagServerIP**.

☞ This function has to be executed on the RAS client station, "not" on the server station. (Essentially, the station is assigned the IP address of the server to which it is connected.)

Examples *Using the phonebook name "Office DialUp" and writing the resulting IP address to **ServerIPTag**:*

```
DialGetServerIP(0,"Office DialUp","ServerIPTag")
```

*Using the modem name "USRobotics_SportsterFaxModem", writing the resulting IP address to **ServerIPAddress** and updating every second (i.e., every time the system tag **Second** changes):*

```
DialGetServerIP(1,"USRobotics_SportsterFaxModem","ServerIPAddress",Second)
```

*Using the direct connection name "DirectDial" and writing the resulting IP address to **IPAdd**:*

```
DialGetServerIP(2,"DirectDial","IPAdd")
```

19.24.5 DialStatus

Determines the status of the dial-up connection.

☞ This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
DialStatus	synchronous	supported (see note)	supported	supported

Syntax

```
DialStatus( numType, strPhonebookEntryOrModem, "optStatus", optRefresh )
```

Inputs

numType

A numeric parameter specifying the content for the **strPhonebookEntryorModem** parameter.

- 0: Phonebook name
- 1: Modem name
- 2: Direct connection name

strPhonebookEntryOrModem

The name of the phonebook, modem or direct connection to establish the connection. The **numType** parameter specifies which of these methods is used.

optStatus

"Optional" The name of the String tag receiving the status message.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the function tries to get the value of the specified tag.

optRefresh

"Optional" tag causing the function to update its return value. This parameter is optional. However, it has to be configured if this function is to be used for an object animation, page 246 (e.g., Text Data Link, Position).

Returned value

Value	Description
-5	Phonebook or modem does not exist
-4	Invalid value for the **numType** parameter ("0" or "1")
-3	Error: Invalid number of parameters (minimum of 2)
-2	Error: **INDRAS.DLL** damaged
-1	Error: **INDRAS.DLL** not found
0	Opening the port…
1	Port was opened successfully
2	Connecting to the device…
3	The device has established a successful connection
4	All devices in the device chain have been connected successfully
5	Checking user name and password…
6	An authentication event has occurred
7	Another validation attempt with a new user was requested
8	Server has requested a callback number
9	The client has requested to change the password
10	Registering the computer on the network…
11	The connection speed calculation phase is starting…

Value	Description
12	An authentication request is being acknowledged
13	Reauthentication (after callback) is starting
14	The client has successfully completed authentication
15	The line is about to be disconnected for callback
16	Delaying to give the modem time to reset for callback
17	Waiting for an incoming call from the server
18	Projection result information is available
19	User authentication is being initiated or retried
20	Client has been called back and is about to resume authentication
21	Logging on to the network…
22	Subentry has been connected
23	Subentry has been disconnected
24	Terminal state supported by **RASPHONE.EXE**
25	Retry authentication state supported by **RASPHONE.EXE**
26	Callback state supported by **RASPHONE.EXE**
27	Change password state supported by **RASPHONE.EXE**
8192	Successfully connected to remote server!
8193	Disconnected

Examples

Tag name	Expression
Tag	DialStatus(0, "Office DialUp")
Tag	DialStatus(1, "USRobotics_SportsterFaxModem", "StatusMessage", second)
Tag	DialStatus(2, "DirectDial", "DialupError")

For further information, refer to: FindModem(), page 886

19.24.6 DialUp

Establishes a dial-up connection.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
DialUp	asynchronous	supported (see note)	supported	supported

Syntax

```
DialUp( numType, strPhonebookEntryOrModem, strUserName, strPassword, optStrDomain, strPhoneNumber )
```

Inputs

numType

A numeric flag that specifies the content of the **strPhonebookEntryorModem** parameter.

- 0: Phonebook name
- 1: Modem name
- 2: Direct connection name

strPhonebookEntryOrModem

The name of the phonebook, modem or direct connection to establish the connection. The **numType** parameter specifies which of these methods is used.

strUserName

The user name for logging on.

strPassword

The password for logging on.

optStrDomain

"Optional" The domain name for logging in.

strPhoneNumber

The phone number to be dialed (used only if parameter numType=1).

Returned value

Value	Description
0	OK: Dialing started
−1	Error: **INDRAS.DLL** not found
−2	Error: **INDRAS.DLL** damaged
−3	Error: Invalid number of parameters (minimum of 5)
-4	Invalid value for the **numType** parameter (0 or 1)
-5	Invalid value for the **strPhonebookEntryOrModem** parameter (String)
-6	Phonebook or modem does not exist
-7	Phonebook or modem in use
-8	Depending on the **numType** parameter: • If **numType** = 0: Could not read properties from phonebook. • If **numType** = 1: More than 1000 active connections simultaneously.
-9	Preliminary phonebook could not be created

The RAS server of the operating system automatically carries out dial-up for Windows PC stations.

Examples

Tag name	Expression
Tag	DialUp(0, "OfficeDialup", "Guest", "Password")
Tag	DialUp(1, "USRobotics_SportsterFaxModem", "HR12378", "HRPass", "15125554321")
Tag	DialUp(2, "DirectDial", "Rberton", "MyPassword", "156.48.25.0")

For further information, refer to: FindModem(), page 886

19.24.7 DialUpToCE

Runs the "DialUpToCE" program sending the information required for server call back by means of **CERasSvr.exe**.

☞ This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
DialUpToCE	asynchronous	supported (see note)	not supported	supported

Syntax

```
DialUpToCE(numModem,strDialPhone,strMyNumber,strUser,strPassword,optStrDomain,optAutoDial,optAutoClose)
```

Inputs

numModem

The modem used to dial to the Windows CE remote station.

strDialPhone

The phone number of the Windows CE remote station.

strMyNumber

The phone number sent to the Windows CE remote station. **CERasSvr.exe** calls back to this phone number.

strUser

The user name to be sent to the Windows CE remote station. **CERasSvr.exe** uses this name to connect to the Windows XP/Vista/7 computer after calling back.

strPassword

The password to be sent to the Windows CE remote station. **CERasSvr.exe** uses this password to connect to the Windows XP/Vista/7 computer after calling back.

optStrDomain

"Optional" The domain name for logging in.

optAutoDial

"Optional" tag with the following possible values:

- 1: Triggers the **DialupToCE** connection automatically if the function is executed
- 0: Requests confirmation before triggering the **DialupToCE** connection automatically if the function is executed

optAutoClose

"Optional" tag with the following possible values:

- 1: Closes the **DialUpToCE** dialog automatically after dialing the Windows CE remote station
- 0: Keeps open the **DialUpToCE** dialog

Returned value

Value	Description
0	Failed, unable to call "DialUpToCE".
1	Successful, "DialUpToCE" executed.



The **DialUpToCE** program was developed to dial up a remote Windows CE station. As Windows CE v.3.00 does not provide a RAS server, prior to answering a call the **CERasSvr.exe** program and a call back have to be executed on the Windows XP/Vista/7 computer using parameters sent by the **DialUpToCE** function. The RAS server service has to be configured on the Windows XP/Vista/7 computer to answer the call back from the Windows Embedded device and set the TCP/IP connection.

Examples

Tag name	Expression
Tag	DialUpToCE(0, "12344321", "98765432", "Administrator", "MyPass")
Tag	DialUpToCE(0, "12344321", "98765432", "Administrator", "MyPass", "", 1, 1)

19.24.8 FindAllDevices

Returns the list of all available modems and direct connection interfaces (COM ports) on the local station.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
FindAllDevices	synchronous	supported (see note)	supported	supported

Syntax

```
FindAllDevices( "tagArray" )
```

Inputs

tagArray

Name of a String array tag receiving the list of available modems and direct connection interfaces.

Returned value Returns the number of modems and/or interfaces found.

Examples

Tag name	Expression
Tag	FindAllDevices("SerialConnections[1]")

19.24.9 FindModem

Returns a list of all available modems in the local station.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
FindModem	synchronous	supported (see note)	supported	supported

Syntax

```
FindModem( "tagArray" )
```

Inputs

tagArray

Name of a String array tag receiving the list of available modems.

Returned value Returns the number of modems found.



This function can be used to get the names of the serial interface for a dial-up connection via modem. The determined information can be used for the "strPhonebookEntryOrModem" parameter in the DialError, page 871, DialStatus, page 880, DialUp, page 882 and HangUp, page 886 function.

Examples

Tag name	Expression
Tag	FindModem("Modems[1]")

19.24.10 HangUp

Terminates a dial-up connection.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
HangUp	synchronous	supported (see note)	supported	supported

Syntax

```
HangUp( numType, strPhonebookEntryOrModem )
```

Inputs

numType

A numeric flag specifying the meaning of the **strPhonebookEntryorModem** parameter.

- 0: Phonebook name
- 1: Modem name
- 2: Direct connection name

strPhonebookEntryOrModem

The name of the phonebook, modem or direct connection to establish the connection. The **numType** parameter specifies which of these methods is used.

Returned value

Value	Description
0	OK.
-1	Error: **INDRAS.DLL** not found
-2	Error: **INDRAS.DLL** damaged
-3	Invalid value for the **numType** parameter ("0" or "1")
-4	Phonebook or modem does not exist
-5	No configured modems exist

Examples

Tag name	Expression
Tag	HangUp(0, "OfficeDialup")
Tag	HangUp(1, "USRobotics_SportsterFaxModem")
Tag	HangUp(2, "DirectDial", "Rberton", "MyPassword")

For further information, refer to: FindModem(), page 886

19.24.11 PhoneDialUp

Dials a phone number using the Telephony Application Program Interface (TAPI).

☞ This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
PhoneDialUp	asynchronous	supported (see note)	not supported	supported

Syntax

```
PhoneDialUp( strPhoneNumber, optStrModemName )
```

Inputs

strPhoneNumber

Phone number to be called by the function.

optStrModemName

Name of the modem used to dial. If no modem is specified, the function uses the first modem found on the operating system.

Returned value

Value	Description
0	OK (dial triggered)
−1	Invalid number of parameters
−3	INDTAPI.DLL library not found
-4	PhoneDialUp () function not supported by the current INDTAPI.DLL library

Examples

Tag name	Expression
	PhoneDialUp ("512-123-4567")
	PhoneDialUp (StringPhoneNumberTag)
	PhoneDialUp (StringPhoneNumberTag, StringModemNameTag)

19.24.12 PhoneDisableListen

Terminates reception of the modem for incoming calls.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
PhoneDisableListen	synchronous	supported (see note)	not supported	supported

Syntax

```
PhoneDisableListen( optStrModemName )
```

Inputs

optStrModemName

Name of the modem used to dial. If no modem is specified, the function uses the first modem found on the operating system.

Returned value

Value	Description
1	OK (no reception of incoming calls)
−1	INDTAPI.DLL library not found
−2	PhoneDisableListen () function not supported by the current **INDTAPI.DLL** library

Examples

Tag name	Expression
	PhoneDisableListen()
	PhoneDisableListen("Hayes Compatible Modem on COM1")
	PhoneDisableListen(StringModemNameTag)

For further information, refer to: PhoneEnableListen(), page 888

19.24.13 PhoneEnableListen

Resumes reception of the modem for incoming calls.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
PhoneEnableListen	synchronous	supported (see note)	not supported	supported

Syntax

```
PhoneEnableListen( optStrModemName )
```

Inputs

optStrModemName
Name of the modem used to dial. If no modem is specified, the function uses the first modem found on the operating system.

Returned value

Value	Description
1	OK (reception of incoming calls)
0	Error executing the PhoneEnableListen() function
−1	INDTAPI.DLL library not found
−2	PhoneEnableListen() function not supported by the current INDTAPI.DLL library

Examples

Tag name	Expression
	PhoneEnableListen()
	PhoneEnableListen("Hayes Compatible Modem on COM1")
	PhoneEnableListen(StringModemNameTag)

For further information, refer to: PhoneDisableListen(), page 888

19.24.14 PhoneHangUp

Terminates a connection established with the "PhoneDialUp" function.



This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
PhoneHangUp	asynchronous	supported (see note)	not supported	supported

Syntax

```
PhoneHangUp( optStrModemName )
```

Inputs

optStrModemName
Name of the modem used to dial. If no modem is specified, the function uses the first modem found on the operating system.

Returned value

Value	Description
1	OK (dial-up was interrupted)
−1	INDTAPI.DLL library not found
−2	PhoneHangUp() function not supported by the current INDTAPI.DLL library

Examples

Tag name	Expression
	PhoneHangUp()
	PhoneHangUp("Hayes Compatible Modem on COM1")
	PhoneHangUp(StringModemNameTag)

19.24.15 PhoneStatus

Checks the status of the current connections.

This function is not supported by Windows 7.

Function	Execution	Windows	Embedded	Thin Client
PhoneStatus	synchronous	supported (see note)	not supported	supported

Syntax

```
PhoneStatus ( "strStatus", optStrModemName )
```

Inputs

strStatus

Name of the tag receiving the status description text

optStrModemName

Name of the modem used to dial. If no modem is specified, the function uses the first modem found on the operating system.

Returned value

Value	Description
0	Ready to make a call
1	Call was terminated
2	Line ringing
3	Dial tone
4	Dialing call
5	Call in process
6	Callback
7	Line is busy
8	Line is idle
9	Disconnected

Examples

Tag name	Expression
Tag Status Code	PhoneStatus("String Tag Status")
Tag Status Code	PhoneStatus("String Tag Status", "Hayes Compatible Modem on COM1")
Tag Status Code	PhoneStatus("String Tag Status", "StringModemNameTag")

19.25 ActiveX and .NET control functions

19.25.1 ActiveX and .NET control functions – General Information

These functions are used to execute functions in ActiveX and .NET objects integrated in WinStudio screens and to retrieve and change the properties of these objects.

Function	Description
XGet, page 891	The function XGet gets the current value of a Property on an ActiveX Control or .NET Control object.
XRun, page 892	The function XRun runs a Method on an ActiveX Control or .NET Control object.
XSet, page 893	The function XSet sets the value of a Property on an ActiveX Control or .NET Control object.

19.25.2 XGet

Diese Funktion ruft den aktuellen Wert einer Eigenschaft eines ActiveX- und .NET-Objekts ab.

Funktion	Ausführung	Windows	Embedded	Thin Client
XGet	asynchron	unterstützt	nur ActiveX-Objekte (siehe Hinweis)	unterstützt

Syntax

```
XGet ( strName , strProperties )
```

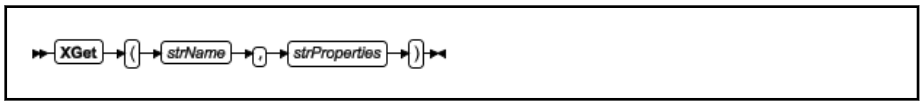


Fig. 19-79: Syntax

Eingaben

strName

Der eindeutige Name des ActiveX- und .NET-Objekts, der im Feld **Name** des Dialogs "Objekteigenschaften" konfiguriert wurde.

strProperties

Der Name der Eigenschaft, deren Wert abgerufen werden soll. Die verfügbaren Eigenschaften werden im Dialog "Konfiguration" (für ActiveX-Objekte) oder "Members" (für .NET-Objekte) angezeigt.

Rückgabewert

Gibt den Wert der angegebenen Eigenschaft zurück.

If the option "Enforce Web functionality" equivalence is selected in the project settings, this function cannot be called from global procedures, page 927 or Script worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens.

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Preferences tab, page 69.

Also, this function is not supported for .NET Control objects running on a Windows Embedded device.

Beispiele

Ruft den aktuellen Wert der Farbeigenschaft auf dem ActiveX-Objekt „ActXRec“ ab:

XGet("ActXRec", "Color")

19.25.3 XRun

Die Funktion XRun führt eine Methode auf einem ActiveX- oder .NET-Steuerobjekt aus.

Funktion	Ausführung	Windows	Embedded	Thin Client
XRun	asynchron	unterstützt	nur ActiveX-Objekte (siehe Hinweis)	unterstützt

Syntax

```
XRun ( strName , strMethod , Parameter1 ,ParameterN)
```

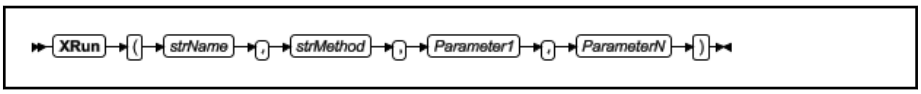


Fig. 19-80: Syntax

Eingaben

strName

Der eindeutige Name des ActiveX- und .NET-Objekts, der im Feld **Name** des Dialogs "Objekteigenschaften" konfiguriert wurde.

strMethod

Die auszuführende Methode. Die verfügbaren Methoden werden im Dialog "Konfiguration" (für ActiveX-Objekte) oder "Members" (für ein .NET-Objekte) angezeigt.

Parameter(1…N)

Verschiedene Datentypen, die für die Ausführung der Methode notwendig sind. Die Anzahl der Parameter kann zwischen 0 und 255 liegen und hängt von der angegebenen Methode ab. Die Datentypen (z.B. Boolean, Integer, Real oder String) von referenzierenden Tags müssen den Parametern in der Methode entsprechen.

Rückgabewert

Diese Funktion gibt das Methodenergebnis wie vom ActiveX- oder .NET-Steuerobjekt ausgegeben zurück. Nicht alle Methoden geben Ergebnisse zurück.

If the option "Enforce Web functionality" equivalence is selected in the project settings, this function cannot be called from global procedures, page 927 or Script worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens.

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Preferences tab, page 69.

Also, this function is not supported for .NET Control objects running on a Windows Embedded device.

Beispiele

Ausführen der Methode XPos auf dem ActiveX-Objekt „ActXCir“ mit vier Originalwerten für die Methode:

XRun("ActXCir", "XPos", FALSE, 12, 4.6, "Das ist mein Text.“)

Ausführen der Methode XPos auf dem ActiveX-Steuerelemente „ActXCir“ mit vier referenzierenden Tags für die Methode:

XRun ("ActXCir", "XPos", TagA, TagB, TagC, TagD)

19.25.4 XSet

Diese Funktion legt den Wert einer Eigenschaft in einem ActiveX- und .NET-Objekt fest.

Funktion	Gruppe	Ausführung	Windows	Embedded	Thin Client
XSet	ActiveX und .NET Control	asynchron	unterstützt	nur ActiveX-Objekte (siehe Hinweis)	unterstützt

Syntax

```
XSet(strName,strProperties,Value)
```

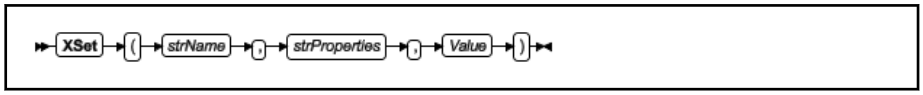


Fig. 19-81: Syntax

Eingaben

strName

Der eindeutige Name des ActiveX- und .NET-Objekts, der im Feld **Name** des Dialogs **Objekteigenschaften** konfiguriert wurde.

strProperties

Die Eigenschaft, deren Wert festgelegt werden soll. Die verfügbaren Eigenschaften werden im Dialog **Konfiguration** (für ActiveX-Objekte) oder **Members** (für .NET-Objekte) angezeigt.

Wert

Beliebiger Wert, der die vorgewählte Eigenschaft im ActiveX- bzw. .NET-Objekt überschreibt.

Rückgabewert

Die Funktion gibt keinen Wert zurück.

If the option "Enforce Web functionality" equivalence is selected in the project settings, this function cannot be called from global procedures, page 927 or Script worksheets, page 401.

This is because the function is meaningful only when it is executed on stations that display project screens.

It might cause unexpected behavior if it is called from background tasks that are executed on the project runtime server.

For more information, see Preferences tab, page 69.

Also, this function is not supported for .NET Control objects running on a Windows Embedded device.

Beispiele

Festlegen des Werts der Anzeigeeigenschaft des ActiveX-Steuerelements "ActXDisplay" auf "Status Normal":

XSet("ActXDisplay", "Display", "Status Normal")

19.26 Event Logger Functions

19.26.1 Event Logger Functions – General Information

These functions are used to send events and comments to the event logger.

Funktion	Beschreibung
SendEvent, page 894	Generates an entry in the event logger file. If the **Comment** option is enabled, the user is prompted to enter a comment after executing the SendEvent() function. This comment is saved in the event logger file.

19.26.2 SendEvent

Sends an event to the event logger.

The function generates an entry in the event logger file. If the **Comment** option is enabled, the user is prompted to enter a comment after executing the SendEvent() function. This comment is saved in the event logger file.

Function	Execution	Windows	Embedded	Thin Client
SendEvent	synchronous	supported	supported	supported

Syntax

```
SendEvent(strEvent,optBoolFlag,optStrComment,optCustom1,…,optCustom10)
```

Inputs

strEvent

The text to be saved in the event logger.

optBoolFlag

The event has no comment if the parameter is not specified or 0 (zero). Otherwise, a comment is saved with the event.

optStrComment

Comment text to be entered into the event logger file if the function is triggered (if optBoolFlag =1). This parameter is optional. Without specified text, the user is prompted to enter a comment by a default dialog.

optCustom1 … optCustom10

Custom fields that will be saved in Event History. Any type of data may be specified.

The number of custom fields can be set in Project Settings: Options, page 54.

Returned value

Value	Description
0	Successful
1	Event logger is disabled in the **Event Settings** dialog
2	Event logger is enabled, however, custom messages are disabled in the **Event Settings** dialog

This function is synchronous. Therefore, the execution of the function is only completed after the event data (including the optional comment) is saved to the database file. It is recommended not to configure this function in a Background task (e.g., "Math" or "Scheduler") if the comment is not to be used or directly entered into the function (without dialog).

Examples

Tag name	Expression
Tag	SendEvent("Valve Open") // Saves the event message.
Tag	SendEvent("Valve Open Oven No. " + OvenID) // Saves the event message concatenated with the value of the OvenID tag.
Tag	SendEvent("Valve Open" , 1) // Displays the dialog where the operator can type comments.
Tag	SendEvent("Valve Open" , 1 , TagComment) // Saves the event message with the comment configured in the TagComment tag.

19.27 FTP Functions

19.27.1 FTP Functions – General Information

These functions are used to configure the FTP settings and to upload and download files to/from a remote server.

Function	Description
CnfFTP, page 895	Configures the FTP settings used by other functions in the project that can transfer files, e.g., the "FTPGet" and "FTPPut" functions.
FTPGet, page 896	Gets a file from a remote server and saves it on the local computer.
FTPPut, page 898	Transfers a file to a remote server.
FTPStatus, page 899	Returns the current status of a file transfer started with "FTPGet" or "FTPPut".

19.27.2 CnfFTP

This function configures the FTP settings used by other functions in the project that can transfer files, e.g., the "FTPGet" and "FTPPut" functions.

Function	Group	Execution	Windows	Embedded	Thin Client
CnfFTP	FTP	synchronous	supported	supported	supported

Syntax

```
CnfFTP(strServer, optStrUser, optStrPwn, optNumPassiveMode, optNumPort)
```

Inputs

strServer

The address of the FTP server.

optStrUser

The user name for the FTP account.

This parameter is optional. If no value is specified, the user name "anonymous" is used by default.

optStrPwd

The password for the FTP account.

This parameter is optional. If no value is specified, the password field is left blank.

optNumPassiveMode

A numeric flag that specifies whether the passive FTP mode is enabled. (The passive FTP can be used to bypass some firewall configurations.)

Value	Description
0	Passive FTP mode disabled
1	Passive FTP mode enabled

This parameter is optional. If no value is specified, the passive FTP mode is disabled by default.

optNumPort

The TCP/IP port number.

This parameter is optional. If no value is specified, port 21 is used by default.

Returned value

Value	Description
0	Successful
-1	Invalid number of parameters
-2	Invalid server name
-3	Invalid user name

☞ This function does not actually connect to the server. For this reason, these error codes do not indicate the quality of the connection. They only indicate whether the FTP settings have been successfully configured.

☞ This function has to be called at least once to configure these settings before the FTPGet, page 896 and FTPPut, page 898 functions can be used to transfer files.

Example

Tag name	Expression
Tag	CnfFTP("ftp.mycompany.com", "admin", "12345", 1) // Configures the FTP server using passive mode.

19.27.3 FTPGet

Gets a file from a remote server and saves it on the local computer.

Function	Group	Execution	Windows	Embedded	Thin Client
FTPGet	FTP	asynchronous	supported	supported	supported

Syntax

```
FTPGet( strRemoteFile, strLocalFile, numTransferType, numOverwrite )
```

Inputs

strRemoteFile

The full path and name of the desired file on the remote server with the syntax **"/file path/file name.extension"**. Some FTP servers are case sensitive. For this reason, always use correct capitalization.

strLocalFile

Full path and name the file is to be saved to on the local computer with the syntax **"C:\file name.extension"**.

numTransferType

Numeric parameter to specify the file transfer type (refer to the table). This parameter is optional. If no value is specified, the transfer type "Unknown" (0) is used by default.

Value	Description
0	Unknown
1	ASCII
2	Binary
10	Unknown, without buffering
11	ASCII, without buffering
12	Binary, without buffering

It is recommended to use option "10". It automatically determines the format (ASCII or Binary) of the remote file and defines the transfer type accordingly. Additionally, it forces the runtime system to download the file from the actual FTP server instead of from an intermediate proxy or cache server.

numOverWrite

A numeric value that specifies whether the local file (specified by **strLocalFile**) may be overwritten if it already exists. This parameter is optional. The default value is 0.

Value	Description
0	Do not overwrite – an error is returned if the file already exists
1	Overwrite

Returned value

Value	Description
1	Failed to establish the FTP connection
0	Successful
-1	Invalid number of parameters

Value	Description
-2	Unknown system error
-3	Invalid remote file
-4	Invalid local file
-5	Invalid transfer type

Before this function can be called, the FTP settings (i.e., server address and login) have to be configured using the CnfFTP, page 895 function.

As the function is executed asynchronously, the FTPStatus, page 899 function has to be used to check the result of the transfer.

Example

Tag name	Expression
Tag	FTPGet("\Reports\040303.txt", "C:\Report.txt") // Retrieves the file **040303.txt** in the **Reports** directory, from the FTP server that was previously specified by CnfFTP(), page 895, it then saves the file locally at **C:\Report.txt**.

19.27.4 FTPPut

Transfers a file to a remote server.

Function	Group	Execution	Windows	Embedded	Thin Client
FTPPut	FTP	asynchronous	supported	supported	supported

Syntax

```
FTPPut( strLocalFile, strRemoteFile, numTransferType )
```

Inputs

strLocalFile

Fully qualified name of the local file (e.g., **"C:\file.extension"**)

strRemoteFile

Fully qualified name of the remote file (e.g., **"/Folder/File.extension"**). It has to be observed that some FTP servers are case sensitive. This has to be considered when entering the file name.

numTransferType

- 0 - Unknown
- 1 - ASCII
- 2 - Binary

Default is 0.

Returned value

Value	Description
1	Failed to create FTP thread
0	Successful

Value	Description
−1	Invalid number of parameters
−2	Unknown system error
−3	Invalid remote file
-4	Invalid local file
-5	Invalid transfer type



Before this function can be called, the FTP settings (i.e., server address and login) have to be configured using the CnfFTP, page 895 function.

As the function is executed asynchronously, the FTPStatus, page 899 function has to be used to check the result of the transfer.

Example

Tag name	Expression
Tag	FTPPut("C:\Report.txt", "\Reports\040303.txt") // Retrieves the file **C:\Report.txt** with the name **040303.txt**in the folder **Reports** in the FTP Server.

19.27.5 FTPStatus

This function returns the current status of a file transfer started with "FTPGet" or "FTPPut".

Function	Group	Execution	Windows	Embedded	Thin Client
FTPStatus	FTP	synchronous	supported	supported	supported

Syntax

```
FTPStatus("strStatusTag")
```

Inputs

strStatusTag

Name of the String tag that is to receive the status description (in plain text) on function feedback.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Value	Description
1	Transaction executed successfully
2	Resolving name
3	Name resolved
4	Connecting to server
5	Connected to server
6	Closing connection

7	Connection closed
8	Sending request
9	Request sent
10	Response received
11	Intermediate response received
12	Response received
13	Request completed
0	No transaction is being executed
−2	Invalid **opttagErrorDescription**
-6	Error opening the connection (see status String for details)
-7	Error establishing the connection (see status String for details)
-8	Error receiving the file (see status String for details)
-9	Transfer pending

Example

Tag name	Expression
StatusCode	FTPStatus("StatusDescription") // Retrieves the status of a current transfer. The return code is stored in the **StatusCode** tag and the description in the **StatusDescription** tag.

19.28 Database/ERP Functions

19.28.1 Database/ERP Functions – General Information

These functions are used to interact with external databases and ERP systems using SQL-like commands.

Function	Description
DBCursorClose, page 901	Closes the cursor and releases the result set.
DBCursorColumnCount, page 902	Gets the total number of columns in a SQL result set.
DBCursorColumnInfo, page 903	Gets information about a column in a SQL result set. The column is specified by a number instead of a name. For this reason, this function can be used to retrieve unknown column names.
DBCursorCurrentRow, page 904	Returns the row number of the current row (i.e., the cursor position) in a SQL result set.
DBCursorGetValue, page 905	Gets the value of the specified column of the current row (i.e., the cursor position) in a SQL result set.
DBCursorMoveTo, page 905	Moves the cursor to the specified row in a SQL result set and copies the values of that row to the mapped tags. If the specified row does not exist – i.e., if it is outside the range of the result set – the function returns an error code and does not change the mapped tags.

Function	Description
DBCursorNext, page 906	Moves the cursor to the next row in a SQL result set and copies the values of that row to the mapped tags. If there is no next row – i.e., if the current row is the last – the function returns an error code and does not change the mapped tags.
DBCursorOpen, page 907	Selects a set of rows and columns in a database table, initializes the cursor at the first row of the result set, copies the values of that row to mapped tags and returns a cursor handle that can be referenced by other DB/ERP functions.
DBCursorOpenSQL, page 909	Selects a set of rows and columns in a database table, initializes the cursor at the first row of the result set, copies the values of that row to mapped tags and returns a cursor handle that can be referenced by other DB/ERP functions. (This function corresponds to a SQL SELECT statement.)
DBCursorPrevious, page 911	Moves the cursor to the previous row of the result set and copies the values of that row to the mapped tags. If there is no previous row – i.e, if the current row is the first –, the function returns an error code and does not change the mapped tags.
DBCursorRowCount, page 912	Gets the total number of rows in an SQL result set.
DBDelete, page 913	Deletes selected rows from a database table. (This function corresponds to an SQL DELETE statement.)
DBExecute, page 914	"DBExecute" is an internal scripting function that executes a custom SQL statement on an external database. If the statement is a query (e.g., SELECT), the database values are copied to specified array tags.
DBInsert, page 916	Inserts one new row into a database table. (This function corresponds to an SQL INSERT statement.)
DBSelect, page 917	"DBSelect" is an internal scripting function that selects a result set from an external database (equivalent to an SQL SELECT statement), maps the columns to array tags in the project and copies the values from the result set to the array tags.
DBUpdate, page 919	Selects a result set and writes the same value to all rows of a specified column. (This function corresponds to an SQL UPDATE statement.)
SyncAlarm, page 920	Synchronizes the alarm database.
SyncAlarmStatus, page 921	Returns the status of a previously called "SyncAlarm" function.
SyncEvent, page 921	Synchronizes the event database.
SyncEventStatus, page 922	Returns the status of a previously called "SyncEvent" function
SyncTrend, page 923	Synchronizes the trend database.
SyncTrendStatus, page 924	Returns the status of a previously called "SyncTrend" function.

19.28.2 DBCursorClose

Closes the cursor and releases the result set.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorClose	Database/ERP	synchronous	supported	supported	supported



This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorClose( numCur, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

In case of errors, a negative value is returned. Otherwise, 0 is returned.

☞ If the cursor is closed, it is destroyed and cannot be used again. A new cursor has to be created using DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

Examples

As used in a math worksheet:

Tag name	Expression
nError-Code	DBCursorClose(nCursor)

19.28.3 DBCursorColumnCount

Gets the total number of columns in a SQL result set.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorCo-lumnCount	Database/ERP	synchronous	supported	supported	supported

☞ This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorColumnCount( numCur, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns the number of columns. In case of errors, a negative value is returned.



Also refer to DBCursorRowCount, page 912.

Examples

As used in a math worksheet:

Tag name	Expression
nError-Code	DBCursorColumnCount(nCursor)

19.28.4 DBCursorColumnInfo

Gets information about a column in a SQL result set. The column is specified by a number instead of a name. For this reason, this function can be used to retrieve unknown column names.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorColum-nInfo	Database/ERP	synchronous	supported	supported	supported



This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorColumnInfo(numCur, numColumn, numTypeInfo, "optStrErrorTag")
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

numColumn

The number of columns information is to be obtained about. Please note that a result set may include only some of the columns in the original database table.

numTypeInfo

The type of information to be retrieved about the column:

Value	Description
0	Column name
1	Column data type

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value In case of errors, a negative value is returned. Otherwise, 0 is returned.

Examples As used in a math worksheet:

Tag name	Expression
ErrorCode	DBCursorColumnInfo(nCursor, 2, 0) // Gets the column name of the second column in the result set.

19.28.5 DBCursorCurrentRow

Returns the row number of the current row (i.e., the cursor position) in a SQL result set.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorCurrentRow	Database/ERP	synchronous	supported	supported	supported

☞ This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorCurrentRow( numCur, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value Returns the number of the current row. In case of errors, a negative value is returned.

Examples As used in a math worksheet:

Tag name	Expression
nRow	DBCursorCurrentRow(nCursor)

19.28.6 DBCursorGetValue

Gets the value of the specified column of the current row (i.e., the cursor position) in a SQL result set.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorGetValue	Database/ERP	synchronous	supported	supported	supported



This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorGetValue(numCur, strColumn, "optStrErrorTag")
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

strColumn

The name of the column.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns the value of the specified column. If the value is NULL or the cursor is invalid, an empty string with quality BAD is returned.

Examples

As used in a math worksheet:

Tag name	Expression
Tag	DBCursorGetValue(nCursor, "Column1")

19.28.7 DBCursorMoveTo

Moves the cursor to the specified row in a SQL result set and copies the values of that row to the mapped tags. If the specified row does not exist – i.e., if it is outside the range of the result set – the function returns an error code and does not change the mapped tags.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorMoveTo	Database/ERP	synchronous	supported	supported	supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorMoveTo( numCur, numRow, "optStrErrorTag" )
```

Syntax

```
DBCursorMoveTo( numCur, numRow, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

numRow

The row of the result set to which the cursor is to be moved.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value In case of errors, a negative value is returned. Otherwise, 0 is returned.

Examples As used in a math worksheet:

Tag name	Expression
nErrorCode	DBCursorMoveTo(nCursor, 4) // Moves the cursor to the fourth row of the result set and copies those values.

19.28.8 DBCursorNext

Moves the cursor to the next row in a SQL result set and copies the values of that row to the mapped tags. If there is no next row – i.e., if the current row is the last – the function returns an error code and does not change the mapped tags.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorNext	Database/ERP	synchronous	supported	supported	supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorNext( numCur, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

In case of errors, a negative value is returned. Otherwise, 0 is returned.

Examples

As used in a math worksheet:

Tag name	Expression
nError-Code	DBCursorNext(nCursor)

19.28.9 DBCursorOpen

Selects a set of rows and columns in a database table, initializes the cursor at the first row of the result set, copies the values of that row to mapped tags and returns a cursor handle that can be referenced by other DB/ERP functions.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorOpen	Database/ERP	synchronous	supported	supported	supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorOpen(strDBConn, strTable, optStrCondition, optStrColumns, optStrTags, optStrOrder, "optStrErrorTag")
```

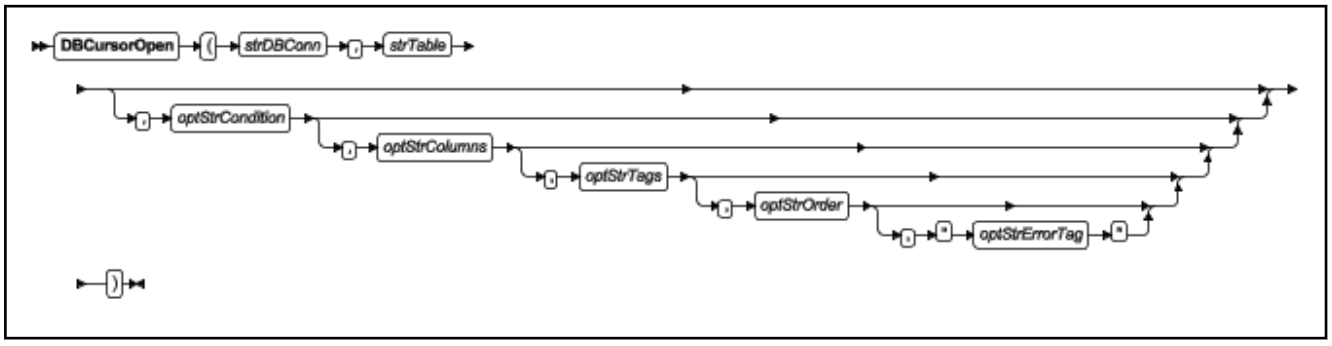


Fig. 19-82: Syntax

Inputs

strDBConn

The name of the database connection. Connections are configured in the Database/ERP, page 404 folder.

strTable

The name of the database table.

optStrCondition

A string that specifies which rows of the table are to be selected. This corresponds to the SQL WHERE clause and the string is to follow the same syntax.

This parameter is optional. If no rows are specified, all rows of the table are selected.

optStrColumns

A string specifying the columns of the table to be selected. This list of column names is to be comma-separated.

This parameter is optional. If no columns are specified, all columns of the table are selected.

optStrTags

A string specifying the project tags to which the columns are to be mapped. This list of tag names is to be comma-separated and in the same order as the columns specified by **optStrColumns**. If the cursor is moved through the result set, the values in the current row are copied to these tags.

This parameter is optional. If no tags are specified, no values are copied.

optStrOrder

The order in which the rows are to be sorted. This corresponds to the SQL ORDER BY clause and the string is to follow the same syntax.

This parameter is optional. If no order is specified, the rows are left in the default order of the table.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.



The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns a numeric value representing the cursor handle. In case of errors, a negative value is returned.

This function corresponds to a SQL SELECT statement, except that it breaks the clauses of the statement into separate function parameters. Users who are familiar with SQL and want to create SELECT statements from scratch can use DBCursorOpenSQL, page 909 instead.

Also refer to DBCursorClose, page 901.

Examples As used in a math worksheet:

Tag name	Expression
nCursor	DBCursorOpen("DB1", "Table1", "Column1 > 3", "Column1, Column2", "Tag1, Tag2", "Column1, Column2 DESC", "TagError") // Opens Table1 of DB1 and selects all rows where Column1 has a value greater than 3. Column1 is mapped to Tag1, and Column2 is mapped to Tag2. Lines are ordered first by Column1, then by Column2, in descending order. Error messages are written to TagError.

19.28.10 DBCursorOpenSQL

Selects a set of rows and columns in a database table, initializes the cursor at the first row of the result set, copies the values of that row to mapped tags and returns a cursor handle that can be referenced by other DB/ERP functions. (This function corresponds to a SQL SELECT statement.)

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorOpenSQL	Database/ERP	Synchronous	Supported	Supported	Supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorOpenSQL( strDBConn, strSQL, optStrTags, "optStrErrorTag" )
```

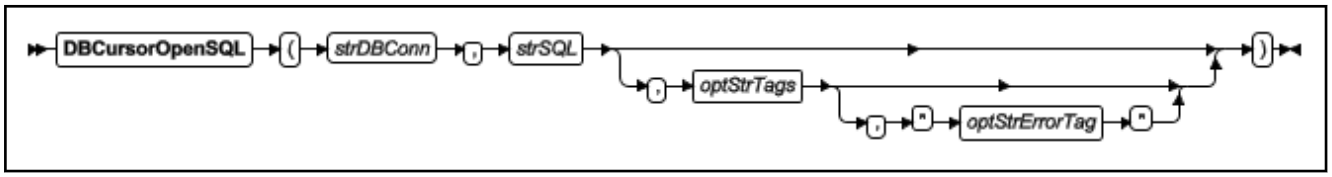


Fig. 19-83: Syntax

Inputs

strDBConn

The name of the database connection. Connections are configured in the Database/ERP, page 404 folder.

strSQL

A string as complete, syntactically correct SQL SELECT statement.

In SQL, curly brackets ("{}") are typically used to enclose an expression that has to be evaluated before the entire SQL statement is executed. For example:

```
SELECT * INTO inmates FROM OPENROWSET
('MSDASQL','Driver={Microsoft Text Driver (*.txt; *.csv)};
DEFAULTDIR=C:\;Extensions=CSV;
','SELECT * FROM flat.csv')
```

In "WinStudio", however, curly brackets are used to reference tags in text fields that are typically not evaluated (e.g., in the label of a Button object). If an SQL statement with this type of expression is given to **DBCursorOpenSQL**, the project tries to evaluate the expression as tag reference and the function fails. To provide an SQL statement in a way that the project can evaluate the expression properly, a new String tag is to be created with the text of the expression and referenced in the SQL statement. For example:

```
$AuxTag = "{Microsoft Text Driver (*.txt; *.csv)}"
$DBCursorOpenSQL( "inmates", "SELECT * INTO inmates FROM
OPENROWSET ('MSDASQL','Driver={AuxTag};
DEFAULTDIR=C:\;Extensions=CSV;
','SELECT * FROM flat.csv')")
```

optStrTags

A string that lists the project tags to which the columns are to be mapped. This list of tag names is to be comma-separated and in the same order as the columns specified by the WHERE clause of **strSQL**. If the cursor is moved through the result set, the values in the current row are copied to these tags.

This parameter is optional. If no tags are specified, no values are copied.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Return value

Returns the cursor handle as numeric value. In case of errors, a negative value is returned.

Usually, one database interface can have up to 1,000 database connections - including the cursor handle.

If this limit has been reached, the database interface closes the oldest connection automatically before it opens a new one. In such a case, use the function DBCursorClose, page 901 to close open cursor handles as soon as these are not required anymore.

It is also possible to increase the maximum number of database connections if required:

Use a text editor to open the project file (normally located under `[…]\Documents\WinStudio v7.4 Projects\project-name\projectname.app`) and edit the following entry:

```
[StDB]
MaxConnections=<from 1 to 32767>
```

Remark: If the number of database connections is (too) high, system performance might reduce.

The "Studio Database Gateway" can be used to directly display database connections (leaks and errors).

Also refer to Database interface, page 553.

Examples

As used in a math worksheet:

Open Table1 of DB1 and select all lines with a value > 3 in Column1. Map Column1 on Tag1 and Column2 on Tag2. Sort the lines in Column1 first, then in Column2; in descending order. Write the error messages to the TagError:

Tag name	Expression
nCursor	DBCursorOpenSQL("DB1", "SELECT Column1, Column2 FROM Table1 WHERE Column1 > 3 ORDER BY Column1, Column2 DESC", "Tag1, Tag2")

19.28.11 DBCursorPrevious

Moves the cursor to the previous row of the result set and copies the values of that row to the mapped tags. If there is no previous row - i.e, if the current row is the first -, the function returns an error code and does not change the mapped tags.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorPrevious	Database/ERP	synchronous	supported	supported	supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBCursorPrevious( numCur, "optStrErrorTag" )
```

Inputs

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value On successful execution, the value 0 is returned. In case of errors, a negative value is returned.

Examples As used in a math worksheet:

Tag name	Expression
ErrorCode	DBCursorPrevious(nCursor)

19.28.12 DBCursorRowCount

Gets the total number of rows in an SQL result set.

Function	Group	Execution	Windows	Embedded	Thin Client
DBCursorRow-Count	Database/ERP	synchronous	supported	supported	supported

☞ This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Also refer to DBCursorColumnCount(), page 902.

Syntax

```
DBCursorRowCount( numCur, "optStrErrorTag" )
```

Eingaben

numCur

The cursor handle for the result set. The cursor handle is returned by DBCursorOpen, page 907 or DBCursorOpenSQL, page 909.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value Returns the number of rows. In case of errors, a negative value is returned.

☞ Also refer to DBCursorColumnCount, page 902.

Examples As used in a math worksheet:

Tag name	Expression
nRow-Count	DBCursorRowCount(nCursor)

19.28.13 DBDelete

Deletes selected rows from a database table. (This function corresponds to an SQL DELETE statement.)

Function	Group	Execution	Windows	Embedded	Thin Client
DBDelete	Database/ERP	synchronous	supported	supported	supported

This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBDelete(strDBConn, strTable, strCondition, "optStrErrorTag")
```

Inputs

strDBConn

The name of the database connection. Connections are configured in the Database/ERP, page 404, folder.

strTable

The name of the database table.

strCondition

A string that specifies which rows of the table are to be selected. This corresponds to the SQL WHERE clause and the string is to follow the same syntax.

To delete all rows in the table, a single space (" ") is to be entered as condition statement.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value

Returns the number of rows deleted from the table. In case of errors, a negative value is returned.

Examples

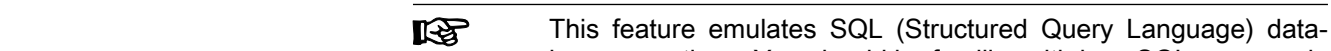
As used in a math worksheet:

Tag name	Expression
nRowsDeleted	DBDelete("DB1", "Table1", "Column1 > 1000", "TagError") // Deletes all rows in Table1 where the value of Column1 is greater than 1000. The returned value (i.e., the number of rows deleted) is written to **TagError.**
Tag	DBDelete("DB1", "Table1", " ") // Deletes all rows of Table1.

19.28.14 DBExecute

"DBExecute" is an internal scripting function that executes a custom SQL statement on an external database. If the statement is a query (e.g., SELECT), the database values are copied to specified array tags.

Function	Group	Execution	Windows	Embedded	Thin Client
DBExecute	Database/ERP	synchronous	supported	supported	supported



This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBExecute(strDBConn,strSQL{ | ,optStrTags,optNumMaxRows{ | ,optStrErrorTag }})
```

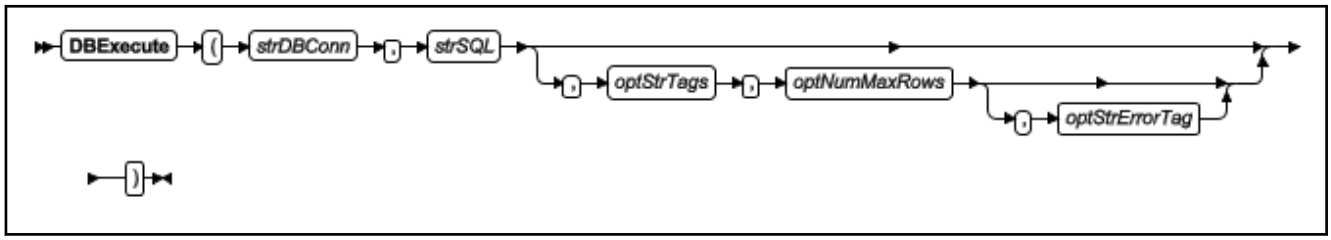


Fig. 19-84: Syntax

Inputs

strDBConn

The name of the database connection. Connections are configured in the **Database/ERP** folder in the "Project Explorer".

strSQL

A complete, syntactically correct SQL statement.



In SQL, curly brackets ("{}") are typically used to enclose an expression that has to be evaluated before the entire SQL statement is executed. For example:

```
SELECT * INTO inmates FROM OPENROWSET
('MSDASQL','Driver={Microsoft Text Driver (*.txt; *.csv)};
DEFAULTDIR=C:\;Extensions=CSV;','SELECT * FROM flat.csv')
```

In "WinStudio", however, curly brackets are used to reference project tags in text fields that are typically not evaluated (e.g., in the label of a Button object). If an SQL statement with this type of expression is given to "DBExecute", the project tries to evaluate the expression as tag reference and the function fails. To provide an SQL statement in a way that the project can evaluate the expression properly, a new String tag is to be created with the text of the expression and referenced in the SQL statement. For example:

```
AuxTag = "{Microsoft Text Driver (*.txt; *.csv)}"
DBExecute( "inmates", "SELECT * INTO inmates FROM
OPENROWSET ('MSDASQL','Driver={AuxTag};
DEFAULTDIR=C:\;Extensions=CSV;
','SELECT * FROM flat.csv')")
```

optStrTags

A comma-separated list of the names of array tags in the project to which the columns of an SQL SELECT result set is to be mapped. The database values are copied to these array tags with the first row of the result set being copied to array index 0. It has to be ensured that the arrays are sufficiently large to receive all rows of the result set.

This parameter is only required if **strSQL** contains an SQL SELECT statement. For all other types of statements, this parameter is ignored and does not need to be specified. However, if the syntax of the function has to be maintained to continue to **optStrErrorTag**, this parameter is to be assigned an empty string ("").

optNumMaxRows

The maximum number of rows to be copied from an SQL SELECT result set. In most cases, to copy all of the rows, a number greater than the expected number of rows in the result set is to be specified.

This parameter is only required if **strSQL** contains an SQL SELECT statement. For all other types of statements, this parameter is ignored and does not need to be specified. However, if the syntax of the function has to be maintained in order to continue to **optStrErrorTag**, this parameter has to be assigned a value of 0.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value This function returns the total number of rows affected by the SQL statement. In case of errors, a negative value is returned.

Please note that this is the value returned by the function itself. In case of an SQL SELECT statement, the database values are copied to the array tags specified by **optStrTags**.

Examples *Program:*

```
DBExecute( "DB1", "INSERT INTO Table1(Column1,Column2) values(1,1)" )
DBExecute("DB1","SELECT max(Column1),max(Column2) FROM Table1","Tag1,Tag2",1,"TagError")
```

19.28.15 DBInsert

Inserts one new row into a database table. (This function corresponds to an SQL INSERT statement.)

☞ This function emulates SQL (Structured Query Language) database operations. Before using this function, users have to familiarize themselves with the structure and execution of SQL commands.

Function	Group	Execution	Windows	Embedded	Thin Client
DBInsert	Database/ERP	synchronous	supported	supported	supported

Syntax

```
DBInsert( strDBConn, strTable, strValues, optStrColumns, "optStrErrorTag" )
```

Inputs

strDBConn

The name of the database connection. Connections are configured in the **Database/ERP** folder in the "Project Explorer".

strTable

The name of the database table.

strValues

A string with the values to be written in the new row. This list of values is to be comma-separated and String values have to be enclosed in quotation marks.

optStrColumns

A string with a list of the columns into which the values are to be written. This list of column names is to be comma-separated and in the same order as the values specified by **strValues**.

This parameter is optional. If no columns are specified, the values are written in the default column order of the database table.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value In case of errors, a negative value is returned. Otherwise, 1 is returned.

Examples As used in a math worksheet:

Tag name	Expression
nError-Code	DBInsert("DB1", "Table1", "1, 'one'", "Column1, Column2")

19.28.16 DBSelect

"DBSelect" is an internal scripting function that selects a result set from an external database (equivalent to an SQL SELECT statement), maps the columns to array tags in the project and copies the values from the result set to the array tags.

Function	Group	Execution	Windows	Embedded	Thin Client
DBSelect	Database/ERP	synchronous	supported	supported	supported

☞ This feature emulates SQL (Structured Query Language) database operations. You should be familiar with how SQL commands are formed and executed before you use this feature.

Syntax

```
DBSelect(strDBConn,strTable,strTags,strColumns,strCondition,strOrder{ | ,optNumMaxRows{ | ,optStrErrorTag}})
```

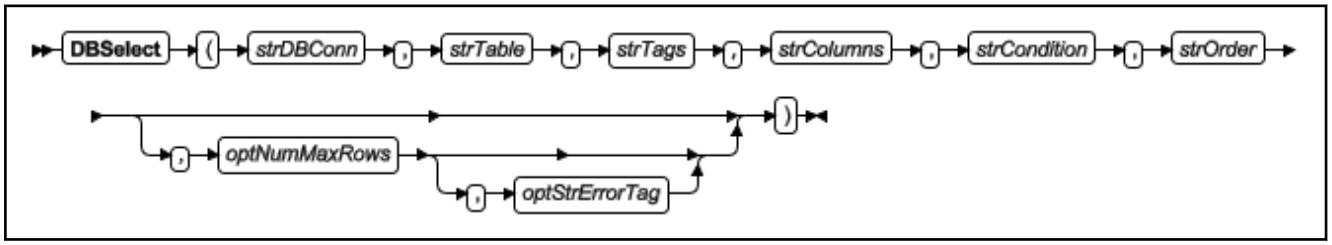


Fig. 19-85: Syntax

Inputs

strDBConn

The name of the database connection. Connections are configured in the **Database/ERP** folder in the "Project Explorer".

strTable

The name of the database table the result set is to be selected from.

strTags

A comma-separated list of the names of array tags in the project the columns of the database table are to be mapped to. The database values are copied to these array tags with the first row of the result set being copied to index 0. It has to be ensured that the arrays are sufficiently large to receive all rows of the result set.

strColumns

A comma-separated list with the columns to be selected from the database table. The list order is to correspond to the list in **strTags**.

To select all columns in the table in their original order, this parameter is to be assigned an empty string ("""").

strCondition

A statement specifying which rows in the database table are to be selected. This corresponds to the SQL WHERE clause and has to follow the same syntax.

To select all of the rows in the table, this parameter is to be assigned an empty string ("").

strOrder

A statement specifying the order the rows are to be sorted in. This corresponds to the SQL ORDER BY clause and has to follow the same syntax.

To keep the rows in their original order, this parameter is to be assigned an empty string ("").

optNumMaxRows

The maximum number of rows to be copied. In most cases, to copy all of the rows, a number greater than the expected number of rows in the result set is to be specified.

This is an optional parameter. If no value is specified, only the first row of the result set is copied.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value This function returns the total number of rows in the result set. In case of errors, a negative value is returned.

It has to be observed that this is the value returned by the function itself. The database values are copied to the array tags specified by **strTags**.

Examples *Program:*

```
DBSelect( "DB1", "Table1", "Array1,Array2", "Column1,Column2", "", "" )
```

```
DBSelect("DB1","Table1","Array1,Array2","Column1,Column2","Column2 < Column1","Column1",4,"TagError")
```

19.28.17 DBUpdate

Selects a result set and writes the same value to all rows of a specified column. (This function corresponds to an SQL UPDATE statement.)

☞ This function emulates SQL (Structured Query Language) database operations. Before using this function, users have to familiarize themselves with the structure and execution of SQL commands.

Function	Group	Execution	Windows	Embedded	Thin Client
DBUpdate	Database/ERP	synchronous	supported	supported	supported

Syntax

```
DBUpdate( strDBConn, strTable, strValues, strColumns, optStrCondition, "optStrErrorTag" )
```

Inputs

strDBConn

The name of the database connection. Available connections are displayed in the "Database Connections" dialog that can be accessed via the context menu of the **Tasks ▸ External database** folder.

strTable

The name of the database table.

strValues

A string that lists the values to be written to the columns. This list of values is to be comma-separated and String values have to be enclosed in quotation marks.

strColumns

A string with a list of the columns into which the values are to be written. This list of column names is to be comma-separated and in the same order as the values specified by **strValues**.

optStrCondition

A string that specifies which rows of the table are to be selected. This corresponds to the SQL WHERE clause and the string is to follow the same syntax.

This parameter is optional. If no rows are specified, all rows of the table are selected.

optStrErrorTag

The name of a String tag that is to receive detailed error messages if errors occur during runtime.

This parameter is optional.

☞ The tag name has to be enclosed in quotation marks as illustrated in the syntax diagram. Otherwise, the project tries to get the value of the specified tag.

Returned value Returns the number of rows updated. In case of errors, a negative value is returned.

Examples As used in a math worksheet:

Tag name	Expression
Tag	DBUpdate("DB1", "Table1", "'X'", "Column2", "Column1 = 1", "TagError") // In Table1 of DB1, for all rows where Column1 equals 1, writes "X" to Column2.

19.28.18 SyncAlarm

Synchronizes the alarm database.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncAlarm	Database/ERP	asynchronous	supported	supported	executed on the server

Syntax

```
SyncAlarm( optStrStartDate, optStrEndDate )
```

Inputs

optStrStartDate

String with the start date. If this parameter is not specified, the current date is used.

optStrEndDate

String with the end date. If this parameter is not specified, the function uses the same value as for the start date.

Returned value

Value	Description
1	Failed to start synchronization; the database is probably being synchronized
0	Successful
-1	Invalid group number

Value	Description
-2	The format is not set to "Database"
-4	The start date specified is invalid
-5	The end date specified is invalid
-6	The start date is greater than the end date



This function is executed asynchronously to ensure that it does not return the result of the synchronization. To get this information, the SyncAlarmStatus, page 921 function is to be used.

Examples

Tag name	Expression
Tag	SyncAlarm() // Synchronizes the database using the current date
Tag	SyncAlarm("10/20/2004") // Synchronizes the database only for the day 10/20/2004
Tag	SyncAlarm("10/20/2004", "10/28/2004") // Synchronizes the database from 10/20/2004 to 10/28/2004

19.28.19 SyncAlarmStatus

Returns the status of a previously called "SyncAlarm" function.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncAlarmStatus	Database/ERP	synchronous	supported	supported	executed on the server

Syntax

```
SyncAlarmStatus()
```

This function does not have any parameters.

Returned value

Value	Description
3	Synchronization complete
2	Synchronization failed
1	Synchronization in progress
0	No synchronization is being executed
-1	The format is not set to "Database"

Examples

Tag name	Expression
Tag	SyncAlarmStatus()

19.28.20 SyncEvent

Synchronizes the event database.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncEvent	Database/ERP	asynchronous	supported	supported	executed on the server

Syntax

```
SyncEvent( optStrStartDate, optStrEndDate )
```

Inputs

optStrStartDate

String with the start date. If this parameter is not specified, the current date is used.

optStrEndDate

String with the end date. If this parameter is not specified, the function uses the same value as for the start date.

Returned value

Value	Description
1	Failed to start synchronization; the database is probably being synchronized
0	Successful
-1	Invalid group number
-2	The format is not set to "Database"
-4	The start date specified is invalid
-5	The end date specified is invalid
-6	The start date is greater than the end date



This function is executed asynchronously to ensure that it does not return the result of the synchronization. To get this information, the SyncEventStatus, page 922 function is to be used.

Examples

Tag name	Expression
Tag	SyncEvent() // Synchronizes the database using the current date
Tag	SyncEvent("10/20/2004") // Synchronizes the database only for the day 10/20/2004
Tag	SyncEvent("10/20/2004", "10/28/2004") // Synchronizes the database from 10/20/2004 to 10/28/2004

19.28.21 SyncEventStatus

Returns the status of a previously called "SyncEvent" function.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncEventStatus	Database/ERP	synchronous	supported	supported	executed on the server

Syntax

```
SyncEventStatus()
```

This function does not have any parameters.

Returned value

Value	Description
3	Synchronization complete
2	Synchronization failed
1	Synchronization in progress

Value	Description
0	No synchronization is being executed
-1	The format is not set to "Database"

Examples

Tag name	Expression
Tag	SyncEventStatus()

19.28.22 SyncTrend

Synchronizes the trend database.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncTrend	Database/ERP	asynchronous	supported	supported	executed on the server

Syntax

```
SyncTrend( numGroup, optStrStartDate, optStrEndDate )
```

Inputs

numGroup

Number of the trend group (number of the trend worksheet).

optStrStartDate

String with the start date. If this parameter is not specified, the current date is used.

optStrEndDate

String with the end date. If this parameter is not specified, the function uses the same value as for the start date.

Returned value

Value	Description
1	Failed to start synchronization; the database is probably being synchronized
0	Successful
-1	Invalid group number
-2	The format is not set to "Database"
-4	The start date specified is invalid
-5	The end date specified is invalid
-6	The start date is greater than the end date

This function is executed asynchronously to ensure that it does not return the result of the synchronization. To get this information, the SyncTrendStatus, page 924 function is to be used.

Examples

Tag name	Expression
Tag	SyncTrend(1) // Synchronizes the group 1 database using the current date
Tag	SyncTrend(1, "10/20/2004") // Synchronizes the group 1 database only for the day 10/20/2004
Tag	SyncTrend(1, "10/20/2004", "10/28/2004") // Synchronizes the group 1 database from 10/20/2004 to 10/28/2004

19.28.23 SyncTrendStatus

Returns the status of a previously called "SyncTrend" function.

Function	Group	Execution	Windows	Embedded	Thin Client
SyncTrendStatus	Database/ERP	synchronous	supported	supported	executed on the server

Syntax

```
SyncTrendStatus(numGroup)
```

Inputs

numGroup
Number of the trend group (trend worksheet)

Returned value

Value	Description
3	Synchronization complete
2	Synchronization failed
1	Synchronization in progress
0	No synchronization is being executed
-1	The format is not set to "Database"

Examples

Tag name	Expression
Tag	SyncTrendStatus(1)

20 Appendix: VBScript language

20.1 Overview of VBScript

VBScript is a simple, standardized and flexible scripting language that enables implementing logics and algorithms within the project.

"WinStudio" implements Visual Basic Scripting Edition 5.5 or higher.

Advantages of the "VBScript" language:

- Syntax, operators and functions.
- The ability to create new variables and procedures (functions and/or sub-routines).
- Access to properties, methods and/or events from COM objects, including ActiveX control elements.
- The ability to execute logic operations in any platform that supports VBScript, including Microsoft Windows-based PCs (running as "WinStudio" server station), Microsoft Windows Embedded devices (via CEView) and Internet Explorer (via Thin Client).

☞ In case of doubt whether the image loaded on the device supports VBScript, please contact the hardware manufacturer. Hardware manufacturers have to enable the support for VBScript on Windows CE devices to enable CEView to execute the scripts configured in the VBScript language on the device.

The purpose of this documentation is to provide an overview for the use of VBScript commands with "Rexroth WinStudio". Furthermore, it can be used as a quick reference for the most used features of the language. For a full description of the language as well as its interfaces and functions, please contact Microsoft. (When this documentation was created, the VBScript documentation could be accessed directly at the Microsoft Developer Network. This link, however, is beyond our control and may change without notice.)

20.2 VBScript interfaces in the software

20.2.1 VBScript interfaces in the software - General information

The following table provides a summary of the VBScript interfaces supported by "Rexroth WinStudio":

Interface	Range of procedures and variables	Execution	Functionality
Global procedures	Graphics and tasks	-	Declaration of procedures
Graphic script	Graphic script interface only	Server (Viewer) + Thin Clients	Declaration of variables, declaration of procedures, execution
Screen script	Screen in which the script is configured	Server (Viewer) + Thin Clients	Declaration of variables, declaration of procedures, execution
Command animation	Object in which the script is configured	Server (Viewer) + Thin Clients	Declaration of variables, execution
ActiveX events	Object in which the script is configured	Server (Viewer) + Thin Clients	Declaration of variables, execution

Interface	Range of procedures and variables	Execution	Functionality
Start script	All script sheets from tasks	Server (BGTask)	Declaration of variables, declaration of procedures, execution
Script groups	Script group only	Server (BGTask)	Declaration of variables, execution

The following figure illustrates the range of each VBScript interface and the order they are scanned by "WinStudio":

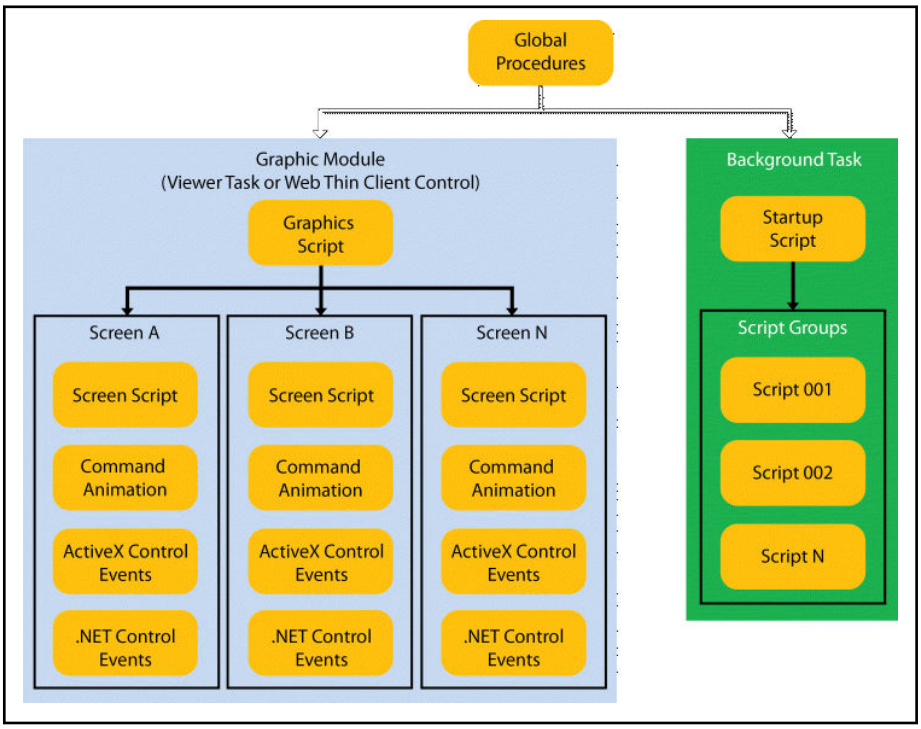


Fig. 20-1: Range of VBScript interfaces

The figure illustrates that the global procedures are shared by the graphics module and the background task. However, the other VBScript interfaces are either from the graphics module or from the background task and they do not share variables or procedures between them. They are independent of each other.

Although the graphic script is scanned by "WinStudio" before the screen scripts, procedures and variables declared in the graphic script interface are NOT available for any script interface configured on the screens. To implement procedures that have to be available to all screens, these have to be specified in the global procedures.

When writing the code into a VBScript interface, any tag from the "WinStudio" tags database or any function from the integrated scripting language can be accessed by applying the "$" prefix to the tag/function name, as shown in the following examples:

Code in a VBScript interface

```
$Time 'Returns the value of the tag Time from the tags database
$MyTag 'Returns the value of the tag MyTag from the tags database
$Open("main") 'Executes the Open() built-in function to open the "main" screen
```

In this way, scripts can be created using built-in functions from WinStudio, tags from the WinStudio tags database, VBScript functions, VBScript varia-

bles, ActiveX properties, methods or events, and any other interface available.

"WinStudio" tags are shared by all modules from "WinStudio", including the graphics module and the background task.

20.2.2 Global procedures

The procedures interface is used create a library of VBScript functions and sub-routines on page 956 that can be called by any other scripting interface in Rexroth WinStudio. The procedures declared here are never directly executed during runtime. They have to be called explicitly by another script.

NOTICE **WinStudio does not keep you from assigning the same name to two or more functions**

Rexroth WinStudio does not keep you from assigning the same name to two or more functions. (This also applies to functions which were imported from external files, see the following section: “Importing functions from an external file”.) In this case, however, the project may behave unexpectedly during runtime. Ensure that all of your functions are named correctly.

Any number of procedure worksheets can be created and each worksheet can contain as many functions and sub-routines as required.

Using the procedures interface:

1. Open the default **Main procedures** worksheet in folder **WinStudio** ▸ **Tasks** ▸ **Global procedures**.

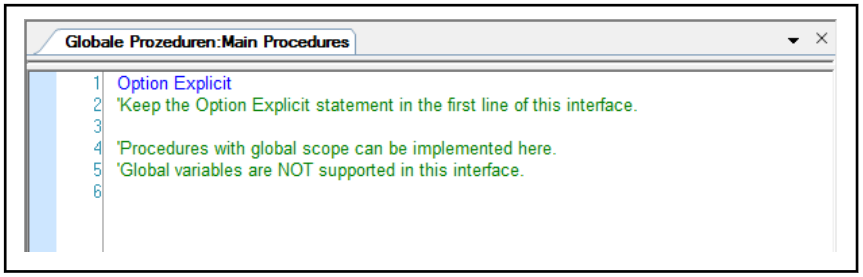


Fig. 20-2: "Global procedures - Main procedures" worksheet

   In this worksheet, any number of procedures can be declared. This worksheet is an integral part of any project and cannot be deleted. To structure procedures, however, additional worksheets can be created as follows.

2. Open the context menu of the **Global procedures** folder and select "New script".

   Double-click on the inserted worksheet to open it in the editor.

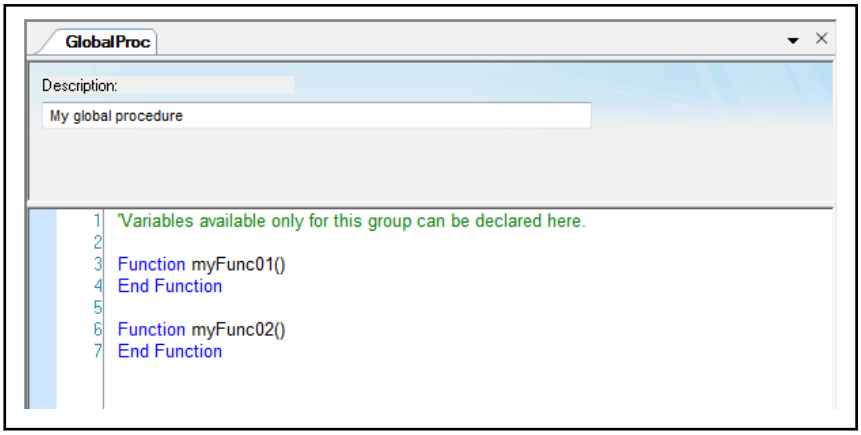


Fig. 20-3: A newly inserted "Global procedures" worksheet

3. Declare the functions and sub-routines in the interface.

Example:

```
Option Explicit
'Keep the Option Explicit statement in the first line
'of this interface.

'Procedures with global scope can be implemented here
'Global variables are NOT supported in this interface

Sub MyMessage(message)
  MsgBox message, 0
End Sub

Function MyAdd(number1, number2)
  MyAdd = number1 + number2
  Call MyMessage("The sum is" & MyAdd & ".")
End Function
```

Local variables can be declared within each procedure, however, global variables cannot be declared in this interface. In most cases, use tags instead.

4. Save the changes. The worksheet is stored in the Procedures folder of the Project Explorer.

Filing procedures in sub-folders

If you have many procedures in one single procedures worksheet, you can file them in sub-folders.

To organize procedures:

1. In the **Procedures** folder, open the worksheet that you want to organize.
2. Enter the following line in the worksheet in front of the procedure to be grouped:

   `,$region:foldername`

   ...where foldername is the name of the subfolder.

Example:

```
'$region:My Subroutines
Sub MyMessage(message)
  MsgBox message, 0
End Sub

'$region:My Functions
Function MyAdd(number1, number2)
  MyAdd = number1 + number2
  Call MyMessage("The sum is" & MyAdd & ".")
End Function
```

3. Save changes. The procedures are stored in the subfolder of the procedures worksheet in the Project Explorer.

Importing functions from an external file

You can also import functions from an external file and add them to the procedures worksheet. This is useful if you have a library of existing functions that you want to reuse.

To import functions into a worksheet:

1. Save and close all open screens and worksheets.
2. In the Procedures folder, click right on the procedures worksheet into which functions are to be imported and then click Import in the shortcut menu.

   The Import - Global procedures dialog window is displayed.

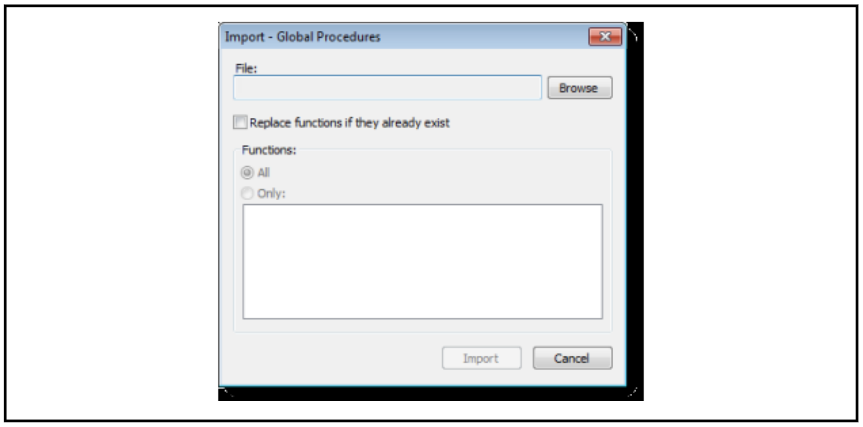


Fig. 20-4: Import - Global procedures dialog window

3. Click on **Browse** right next to the **File** window. A default Windows file browser is displayed. Use it to find and select a procedures worksheet file. (This is a plain text file that has been saved with the `.gis` file extension.)
4. Select **Replace functions if they already exist** to overwrite functions in the worksheet with functions imported from the file, if the functions have the same name.
5. *Do one of the following in the Functions section:*
   - Click All to import all functions from the file
   - or click Only to only import selected functions from the files and then select these functions from the list.
6. Click on “Import”.

If the functions have been imported, they should be shown in the worksheet.

Worksheet password protection

It is possible to create a password for any procedures worksheet to prevent it from being edited or analyzed by other users. To protect a procedures worksheet:

1. In the "Global procedures" folder, right-click on the desired worksheet and select **Password protection** from the context menu. The "Password protection" dialog is displayed.
2. Enter the password into the **New password** field.
3. Enter the password again into the **Confirm password** field.
4. Click on **OK**.

Afterwards, the password is requested every time the worksheet is opened.

20.2.3 Graphics module

Graphic Script

The "graphic script" worksheet is used to declare functions that are to be executed on certain events (see below) of the Graphic module. To open the graphic script, double click on the "Graphic Script" element in the "Scripts" folder.

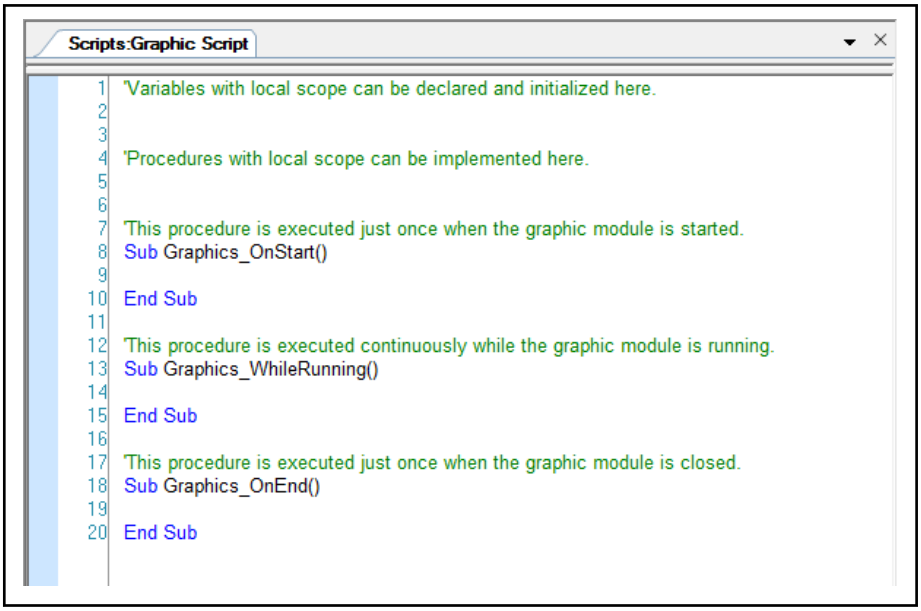


Fig. 20-5: Project Explorer: Graphic script interface

This worksheet contains predefined functions to be executed on the following events:

- **Graphics_OnStart()**: The code configured within this sub-routine is automatically executed once when the Graphic module is started. This interface is useful for initializing variables or executing logics that have to be executed once when starting the runtime system.
- **Graphics_WhileRunning()**: The code configured within this sub-routine is automatically executed continuously while the Graphic module is running. The rate at which this sub-routine is called depends on the performance of the platform where the project is running.
- **Graphics_OnEnd()**: The code configured within this sub-routine is automatically executed once when the Graphic module is closed.
- **Graphics_OnScreenResize(width, height)**: The code configured within this sub-routine is automatically executed once when the screen resolution of the runtime station changes. The new height and width of the screen (in pixels) are passed to the sub-routine as parameters.

☞ The **Graphics_OnScreenResize()** sub-routine is intended for projects running in CEView on Windows Embedded devices that can switch between portrait and landscape display modes. It is "not" supported for projects running on Windows PC or in the Thin Client.

☞ Do not change the name of the predefined sub-routines. Otherwise, the system cannot execute them automatically.

Example:

```
'Variables with local scope can be declared and initialized here
Dim MyDate
MyDate = Date()
Dim MyValue
MyValue = 100

'Procedures with local scope can be implemented here
Function MyNewProcedure(nCount)
    MyNewProcedure = nCount + 1
End Function

Function AreaRec(side1, side2)
  AreaRec = side1 * side2
End Function

Sub CheckHiLimit(myValue, myHiLimit)
  If myValue > myHiLimit Then
    MsgBox("Value out of range")
  End If
End Sub

'This procedure is executed just once when the graphic module is started
Sub Graphics_OnStart()
  MsgBox("Welcome to the system!")
End Sub

'This procedure is executed continuously while the graphic module is running
Sub Graphics_WhileRunning()
  If $UserName = "Guest" Then
    $MyFlag = 0
  End If
End Sub

'This procedure is executed just once when the graphic module is closed
Sub Graphics_OnEnd()
  $LogOff()
End Sub
```

The following has to be observed when executing the graphic script:

On the server (where "Rexroth WinStudio" or CEView is running):

- The Graphic module is the Viewer task.
- The **Graphics_OnStart()** sub-routine is executed once on the server when the Viewer task is launched.
- The **Graphics_WhileRunning()** sub-routine keeps being executed on the server while the Viewer task is running. The **Graphics_OnEnd()** sub-routine is executed once on the server when the Viewer task is shut down.

On the Thin Client or Secure Viewer:

- The Graphic module is the ISSymbol control element.
- The **Graphics_OnStart()** sub-routine is executed once on the Thin Client station after logging in successfully.
- The **Graphics_WhileRunning()** sub-routine keeps being executed on the Thin Client station while the ISSymbol control element is hosted by the web browser.
- The **Graphics_OnStart()** sub-routine is executed once on the Thin Client station when the web browser is shut down (or when the ISSymbol control element is no longer hosted by the web browser).

The execution of the graphic script sub-routines on the server is completely independent of the execution on the Thin Client and Secure Viewer stations.

Calling graphic script procedures in other VBScript interfaces

The three predefined sub-routines are exclusively available to the Graphic module and are only executed on above mentioned events. In the graphic script, however, further functions can be declared that can be accessed by

other interfaces (e.g., in screen scripts or command objects) by means of the **Graphics.procedure_name** syntax.

In the **MyNewProcedure** function declared in the example above, a Button object could be placed in the WinStudio screen and a command animation could be applied to it with the following line:

$NewTag = Graphics.MyNewProcedure($oldTag)

Screen Script

To edit the screen script of a WinStudio script, open it in the editor and select **Screen Script** from the context menu of the editor.

Screen scripts can be used to execute logics on the following events based on preconfigured sub-routines:

- **Screen_OnOpen()**: The code configured within this sub-routine is automatically executed once when the screen is opened.
- **Screen_WhileOpen()**: The code configured within this sub-routine is automatically executed continuously while the screen is open. The rate at which this sub-routine is called depends on the performance of the platform where the project is running.
- **Screen_OnClose()**: The code configured within this sub-routine is executed once when the screen is closed.
- **Screen_IsClosedByReplace()**: This procedure defines whether the screen is automatically closed if another screen is opened (this is the default behavior). If the function is assigned the value "0" or "FALSE", automatic closing is disabled. If the function is assigned a positive value (e.g., "1") or "TRUE" or if the procedure is not declared at all, automatic closing is enabled.

The variables and procedures declared in this interface are available to the VBScript interfaces of the screen where the screen script is configured.

NOTICE **Do "not" change the names of the preconfigured sub-routines described above.**

Otherwise, they cannot be called by the system automatically.

- The execution of the screen script sub-routines on the server is completely independent of the execution on the Thin Client stations. In other words, these sub-routines are executed asynchronously.
- The procedures and/or variables declared in the screen script interface have local validity.

  They can be called only from the specific screen on which they are declared.

Example:

```
'Variables available on this screen can be declared and initialized here
Dim Counter

'Procedures available on this screen can be implemented here
Function AreaCircle(radius)
  AreaCircle = Sqr(radius) * $Pi()
End Function

Sub CheckLoLimit (myValue, myLoLimit)
  If myValue < myLoLimit Then
    MsgBox("Value out of range")
  End If
```

```
End Sub

'This procedure determines whether the screen is automatically closed
Function Screen_IsClosedByReplace()
  Screen_IsClosedByReplace = $ReplaceModeTag
End Function

'This procedure is executed just once when this screen is open
Sub Screen_OnOpen()
  MsgBox("The screen was open!")
End Sub

'This procedure is executed continuously while this screen is open
Sub Screen_WhileOpen()
  If Counter < 100 Then
    Counter = Counter + 1
  Else
    Counter = 0
  End If
  $SimulationTag = Counter
End Sub

'This procedure is executed just once when this screen is closed
Sub Screen_OnClose()
  MsgBox("The screen will be closed!")
End Sub
```

Command animation

To edit scripts in command animations, proceed as follows:

1. Select the object.
2. Click on the "Command on/off" icon in the **WinStudio dynamic function** toolbar.

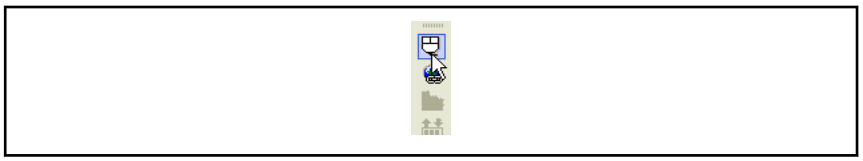


Fig. 20-6: Command animations icon

3. Double-click on the object. The "Object properties" dialog for the command animation is displayed.

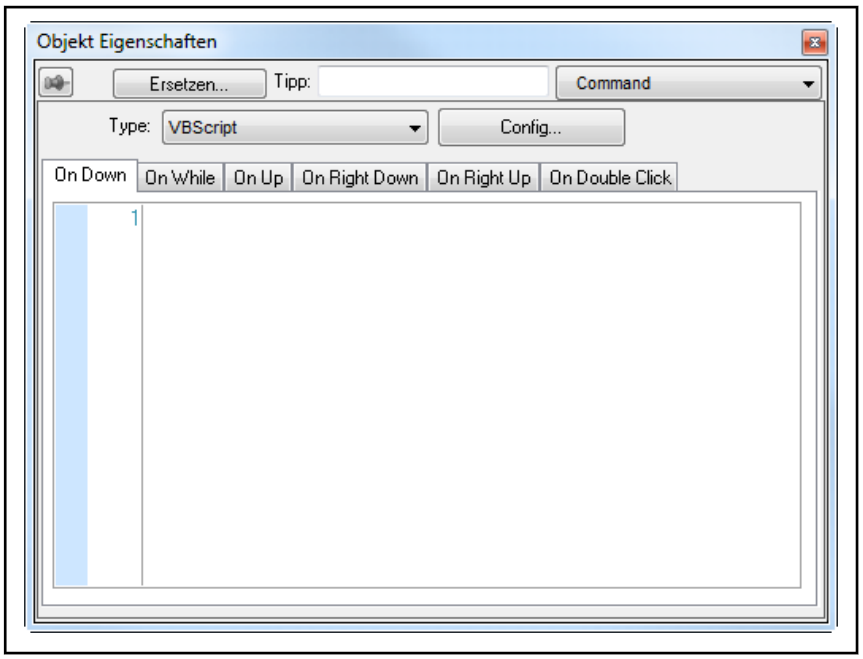


Fig. 20-7: "Object properties" dialog for the command animation

4. Select **VBScript** as **Type**.

5. To edit the script in a larger editor window or to set additional options, click on the **Config...** button.

Use this interface to execute functions when clicking on the object during runtime or pressing the shortcut configured for the object.

Variables declared in this interface are available for this interface only (local range). In other words, they are not available for any other object in the project. No procedures can be implemented in this interface. However, procedures can be called that are implemented in the global procedures or the screen script of the same screen in which the screen object can be found.

For further information, refer to Command animation on page 246.

Example:

```
'The script below will be executed when the user clicks on the object
'where this animation is configured
$MyValue = InputBox("Please enter the new set-point", "Set-point")
```

ActiveX events

To edit the ActiveX events interface, select the Script option from the Events tab of the ActiveX object in the screen.

1. Click on "ActiveX control" in the "WinStudio screen tools" tool window.

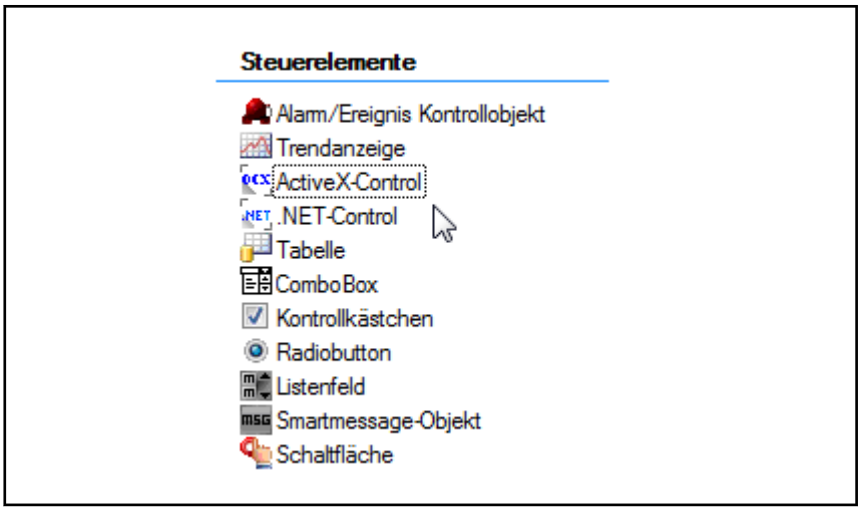


Fig. 20-8: ActiveX control icon in the active objects toolbar

The "Insert ActiveX control" dialog is displayed.

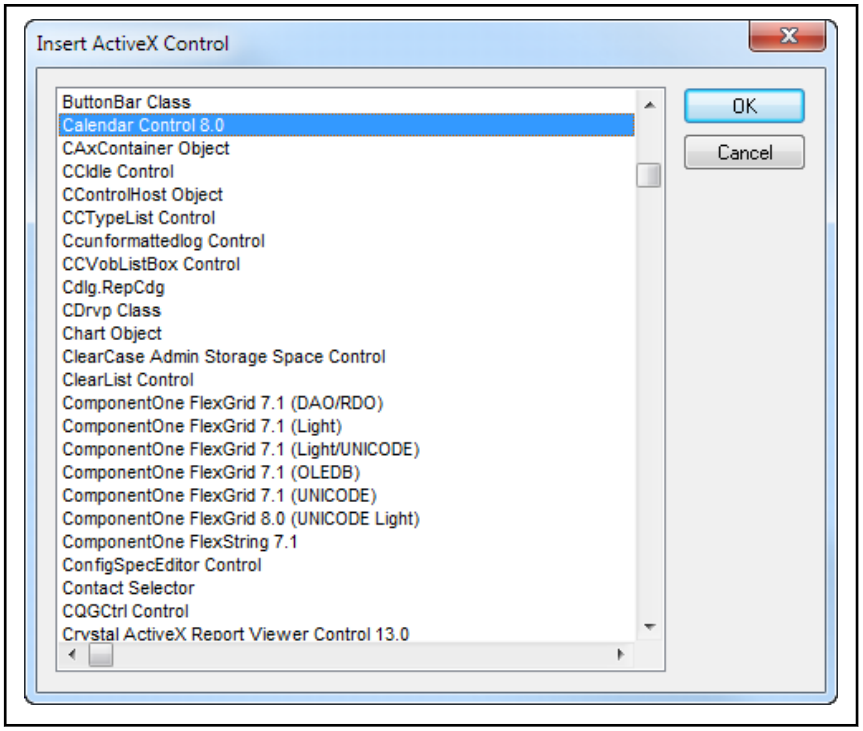


Fig. 20-9: Insert ActiveX control dialog

2. Select the ActiveX control to be used and click on **OK**.
3. The object is created and inserted in the top-left corner of the screen. Double-click on the object to open the "Object properties" dialog.

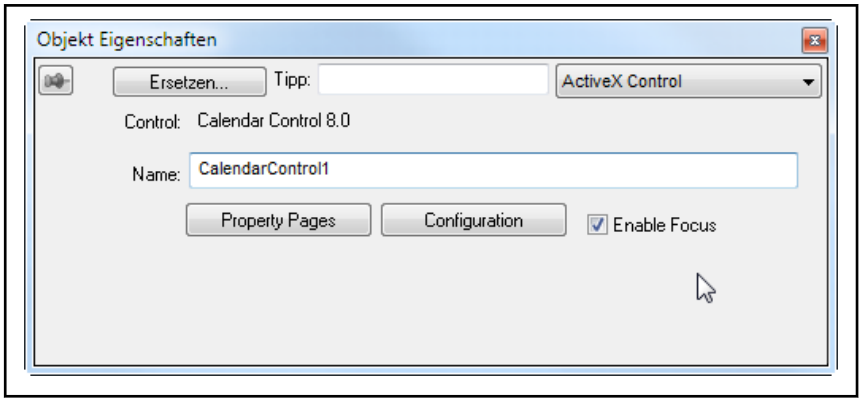


Fig. 20-10: Object Properties dialog

4. Click on the **Configuration** button. The "Configuration" dialog is displayed. Click on the “Event” tab.

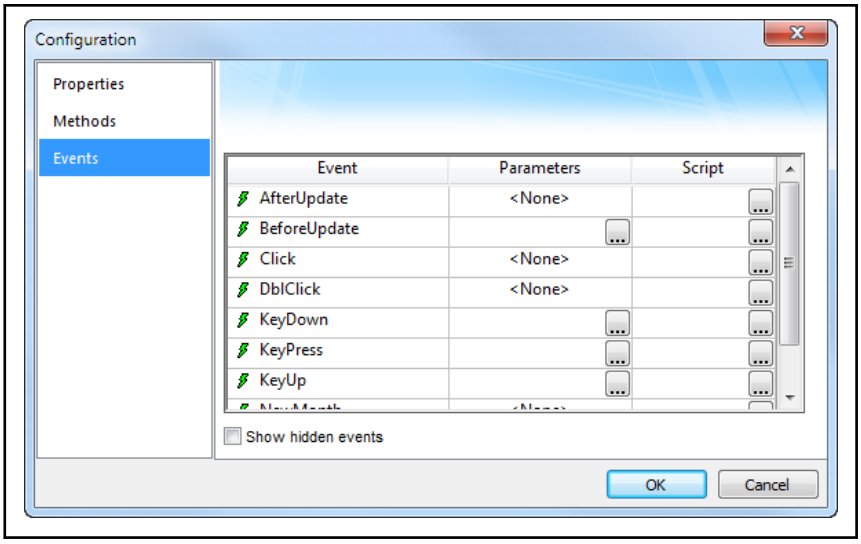


Fig. 20-11: "Event" tab

5. Click on the ... button in the Script column.



Fig. 20-12: Script column

This interface is used to execute logics if an ActiveX object triggers an event.

Variables declared in this interface are available for this interface only (local range). They are not included in other objects used in the project.

No procedures can be implemented in this interface. However, procedures can be called that are implemented in the global procedures or the screen script of the same WinStudio screen in which the ActiveX object can be found.

For further information, refer to ActiveX control object, page 231.

Example:

```
'The script below will be executed when the Calendar Control ActiveX
'triggers its "AfterUpdate" event
$MyYear = CalendarControl1.Year
$MyMonth = CalendarControl1.Month
$MyDay = CalendarControl1.Day
```

20.2.4 Background task

For further details, refer to:

- Script worksheet startup on page 402,
- Script worksheet on page 401.

20.3 Language reference

20.3.1 Language reference – Overview

- Constants, page 938

- Functions, page 946
- Errors, page 947
- Keywords, page 946
- Methods, page 945
- Objects and collections, page 941
- Operators, page 937
- Properties, page 942
- Statements, page 943
- System tags, page 949

20.3.2 Operators

Arithmetic operators

Symbol	Name	Description
^	Exponentiation	Raises a number to the power of an exponent
-	Unary negation	Finds the difference between two numbers or indicates the negative value of a numeric expression
*	Multiplication	Multiplies two numbers
/	Division	Divides two numbers and returns a floating point result
\	Integer division	Divides two numbers and returns an integer result
Mod	Modulus arithmetic	Divides two numbers and returns only the remainder
+	Addition	Finds the sum of two numbers
-	Subtraction	Finds the difference between two numbers or indicates the negative value of a numeric expression
&	String concatenation	Forces string concatenation of two expressions

Comparison operators

Symbol	Name	Description
=	Equality	Comparison is True if the first expression is equal to the second expression
<>	Inequality	Comparison is True if the first expression is different from the second expression
<	Less than	Comparison is True if the first expression is less than the second expression
>	Greater than	Comparison is True if the first expression is greater than the second expression
<=	Less than or equal to	Comparison is True if the first expression is less than or equal to the second expression
>=	Greater than or equal to	Comparison is True if the first expression is greater than or equal to the second expression
Is	Object equivalence	Compares two object reference variables. Comparison is True if both object names refer to the same object

Logical operators

Symbol	Name	Description
Not	Logical negation	Performs logical negation on an expression
And	Logical conjunction	Performs a logical conjunction on two expressions
Or	Logical disjunction	Performs a logical disjunction on two expressions
Xor	Logical exclusion	Performs a logical exclusion on two expressions
Eqv	Logical equivalence	Performs a logical equivalence on two expressions
Imp	Logical implication	Performs a logical implication on two expressions

Assignment operators

Symbol	Name	Description
=	Assignment	Assigns a value to a variable or property

20.3.3 Constants

Color constants

Constant	Value	Description
vbBlack	&h00	Black
vbRed	&hFF	Red
vbGreen	&hFF00	Green
vbYellow	&hFFFF	Yellow
vbBlue	&hFF0000	Blue
vbMagenta	&hFF00FF	Magenta
vbCyan	&hFFFF00	Cyan
vbWhite	&hFFFFFF	White

Comparison constants

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison
vbTextCompare	1	Perform a textual comparison

Date & time constants

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday

Constant	Value	Description
vbSaturday	7	Saturday
vbUseSystem-DayOfWeek	0	Uses the day of the week specified in the system settings for the first day of the week
vbFirstJan1	1	Uses the week in which January 1 occurs (default)
vbFirstFourDays	2	Uses the first week that has at least four days in the new year
vbFirstFullWeek	3	Uses the first full week of the year

Date format constants

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. For real numbers, date and time are displayed. If there is no fractional part, only a date is displayed. If there is no integer part, only the time is displayed. Date and time display depends on the system settings
vbLongDate	1	Displays a date using the long date format specified in the regional settings of the computer
vbShortDate	2	Displays a date using the short date format specified in the regional settings of the computer
vbLongTime	3	Displays a time using the long time format specified in the regional settings of the computer
vbShortTime	4	Displays a time using the short time format specified in the regional settings of the computer.

Miscellaneous constants

Constant	Value	Description
vbObjectError	-2147221504	User-defined error numbers have to be greater than this value

Window constants – Buttons and icons

Constant	Value	Description
vbOKOnly	0	Displays the OK button only
vbOKCancel	1	Displays OK and Cancel buttons
vbAbortRe-tryIgnore	2	Displays Abort, Retry and Ignore buttons
vbYesNoCancel	3	Displays Yes, No and Cancel buttons
vbYesNo	4	Displays Yes and No buttons
vbRetryCancel	5	Displays Retry and Cancel buttons
vbCritical	16	Displays a Critical Message icon
vbQuestion	32	Displays a Warning Query icon
vbExclamation	48	Displays a Warning Message icon
vbInformation	64	Displays an Information Message icon
vbDefaultBut-ton1	0	First button is the default

Constant	Value	Description
vbDefaultBut-ton2	256	Second button is the default
vbDefaultBut-ton3	512	Third button is the default
vbDefaultBut-ton4	768	Fourth button is the default
vbApplication-Modal	0	Application modal. The user has to respond to the message box before continuing to work in the current application
vbSystemModal	4096	System modal. On Win16 systems, all programs are suspended until the user responds to the message box. On Win32 systems, this constant provides a program modal message box that always remains on top of any other running programs

Box constants – Selected buttons

Constant	Value	Description
vbOK	1	OK button was clicked
vbCancel	2	Cancel button was clicked
vbAbort	3	Abort button was clicked
vbRetry	4	Retry button was clicked
vbIgnore	5	Ignore button was clicked
vbYes	6	Yes button was clicked
vbNo	7	No button was clicked

String constants

Constant	Value	Description
vbCr	Chr(13)	Return
VbCrLf	Chr(13) & Chr(10)	Return…line feed combination
vbFormFeed	Chr(12)	Line feed; not useful in Microsoft Windows
vbLf	Chr(10)	Line feed
vbNewLine	Chr(13) & Chr(10) or Chr(10)	Platform-specific new line character; according to the platform
vbNullChar	Chr(0)	Characters with the value 0
vbNullString	String with the value 0	Not the same as a zero-length string (""); used for calling external procedures
vbTab	Chr(9)	Horizontal tab
vbVerticalTab	Chr(11)	Vertical tab; not useful in Microsoft Windows

Tristate constants

Constant	Value	Description
vbUseDefault	−2	Uses the default value from the regional settings of the computer
vbTrue	−1	TRUE
vbFalse	0	FALSE

VarType constants

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	Integer subtype
vbLong	3	Long subtype
vbSingle	4	Single subtype
vbDouble	5	Double subtype
vbCurrency	6	Currency subtype
vbDate	7	Date subtype
vbString	8	String subtype
vbObject	9	Object
vbError	10	Error subtype
vbBoolean	11	Boolean subtype
vbVariant	12	Variant (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	Decimal subtype
vbByte	17	Byte subtype
vbArray	8192	Array

20.3.4 Objects and Collections

Class object

The object created using the class statement. Provides access to the events of the class.

Debug object

An intrinsic global object that can send output to a script debugger like the Microsoft Script Debugger.

Err object

Contains information about runtime errors. Accepts the Raise and Clear methods for generating and clearing runtime errors.

Match object

Provides access to the write-protected properties of a regular expression match.

Matches collection

Collection of regular expression match objects.

Regular expression object (RegExp)

Provides simple regular expression support.

SubMatches collection

Collection of regular expression submatch strings.

20.3.5 Properties

Description

Returns or sets a descriptive string associated with an error.

FirstIndex

Returns the position in a search string where a match occurs.

Global

Sets or returns a Boolean value that indicates whether a pattern is to match all occurrences in an entire search string or just the first one.

HelpContext

Sets or returns a context ID for a topic in a help file.

HelpFile

Sets or returns a fully qualified path to a help file.

IgnoreCase

Sets or returns a Boolean value that indicates if a pattern search is case-sensitive or not.

Length

Sets or returns a Boolean value that indicates if a pattern search is case-sensitive or not.

Number

Returns or sets a numeric value specifying an error. Number is the default property of the Err object.

Pattern

Sets or returns the regular expression pattern being searched for.

Source

Returns or sets the name of the object or application that originally generated the error.

Value

Returns the value or text of a match found in a search string.

20.3.6 Statements

Call

Transfers the control to a sub or function procedure.

Class

Declares the name of a class and a definition of the variables, properties and methods that comprise the class.

Const

Declares constants for use instead of literal values.

Dim

Declares variables and allocates storage space.

Do…Loop

Repeats a block of statements while a condition is True or until a condition becomes True.

Erase

Reinitializes the elements of fixed-size arrays and removes the allocation of dynamic-array storage space.

Execute

Executes one or more specified statements.

ExecuteGlobal

Executes one or more specified statements in the global namespace of a script.

Exit

Exits a block of Do…Loop, For…Next, Function, or Sub code.

For Each…Next

Repeats a group of statements for each element in an array or collection.

For … Next

Repeats a group of statements a specified number of times.

Function

Declares the name, arguments and code that form the body of a function procedure.

If…Then…Else

Conditionally executes a group of statements, depending on the value of an expression.

Option Explicit

Forces explicit declaration of all variables in a script.

Private

Declares private variables and allocates storage space. Declares a private variable in a class block.

Property Get

Declares the name, arguments and code in a class block that form the body of a property procedure that gets (returns) the value of a property.

Property Let

Declares the name, arguments and code in a class block that form the body of a property procedure that assigns (sets) the value of a property.

Property Set

Declares the name, arguments and code in a class block that form the body of a property procedure that sets a reference to an object.

Public

Declares public variables and allocates storage space. Declares a private variable in a class block.

Ramdomize

Initializes the random generator for numbers.

ReDim

Declares dynamic-array variables and allocates or reallocates storage space at procedure level.

Rem

Includes explanatory remarks in a program.

Select

Executes one of several groups of statements, depending on the value of an expression.

Set

Assigns an object reference to a variable or property or associates a procedure reference with an event.

Stop

Suspends execution.

Sub

Declares the name, arguments and code that form the body of a sub-procedure.

While

Executes a series of statements as long as a given condition is True.

With

Executes a series of statements on a single object.

20.3.7 Methods

Clear

Deletes all property settings of the Err object.

Execute

Executes a regular expression search with a specified String.

Raise

Generates a runtime error.

Replace

Replaces text found in a regular expression search.

Test

Executes a regular expression search with a specified String and returns a Boolean value that indicates if a pattern match was found.

Write

Sends Strings to the script debugger.

WriteLine

Sends Strings to the script debugger followed by a "New Line" character.

20.3.8 Functions

Function Names			
Abs	Array	Asc	Atn
CBool	CByte	CCur	CDate
CDbl	Chr	CInt	CLng
Conversions	Cos	CreateObject	CSng
CStr	Date	DateAdd	DateDiff
DatePart	DateSerial	DateValue	Day
Derived Math	Escape	Eval	Exp
Filter	FormatCurrency	FormatDateTime	FormatNumber
FormatPercent	GetLocale	GetObject	GetRef
Hex	Hour	InputBox	InStr
InStrRev	Int, Fix	IsArray	IsDate
IsEmpty	IsNull	IsNumeric	IsObject
Join	LBound	LCase	Left
Len	LoadPicture	Log	LTrim; RTrim; and Trim
Maths	Mid	Minute	Month
MonthName	MsgBox	Now	Oct
Replace	RGB	Right	Rnd
Round	ScriptEngine	ScriptEngineBuildVersion	ScriptEngineMajorVersion
ScriptEngineMinorVersion	Second	SetLocale	Sgn
Sin	Space	Split	Sqr
StrComp	Character string	StrReverse	Tan
Time	Timer	TimeSerial	TimeValue
TypeName	UBound	UCase	Unescape
VarType	Weekday	WeekdayName	Year

20.3.9 Keywords

Empty

The keyword Empty is used to indicate an uninitialized variable value. This does not correspond to Null.

False

The keyword False has a value equal to 0.

Nothing

The keyword Nothing in VBScript is used to disassociate an object variable from any actual object.

Null

The keyword Null is used to indicate that a variable contains no valid data. This does not correspond to Empty.

True

The keyword True has a value equal to -1.

20.3.10 Errors

VBScript runtime errors

Error number	Description
5	Invalid procedure call or argument
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
17	Cannot perform requested operation
28	Out of stack space
35	Sub or function not defined
48	Error in loading DLL
51	Internal error
91	Object variable not set
92	"For" loop not initialized
94	Invalid use of Null
424	Object required
429	ActiveX component cannot create object
430	Class does not support automation
432	File name or class name not found during automation operation
438	Object does not support this property or method
445	Object does not support this action
447	Object does not support current local setting
448	Named argument not found
449	Argument not optional

Error number	Description
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
458	Variable uses an automation type not supported in VBScript
462	The remote server device does not exist or is not available
481	Invalid picture
500	Variable is undefined
502	Object not safe for scripting
503	Object not safe for initializing
504	Object not safe for creating
505	Invalid or unqualified reference
506	Class not defined
507	An exception occurred
5008	Invalid assignment
5017	Syntax error in regular expression
5018	Unexpected bracket characters
5019	Expected ']' in regular expression
5020	Expected ')' in regular expression
5021	Invalid range in character set

VBScript syntax errors

Error number	Description
1001	Out of memory
1002	Syntax error
1005	Expected '('
1006	Expected ')'
1010	Expected identifier
1011	Expected '='
1012	Expected 'If'
1013	Expected 'To'
1014	Expected 'End'
1015	Expected 'Function'
1016	Expected 'Sub'
1017	Expected 'Then'
1018	Expected 'Wend'
1019	Expected 'Loop'
1020	Expected 'Next'

Error number	Description
1021	Expected 'Case'
1022	Expected 'Select'
1023	Expected expression
1024	Expected statement
1025	Expected end of statement
1026	Expected integer constant
1027	Expected 'While' or 'Until'
1028	Expected 'While', 'Until' or end of statement
1029	Expected 'With'
1030	Identifier too long
1037	Invalid use of 'Me' keyword
1038	'loop' without 'do'
1039	Invalid 'exit' statement
1040	Invalid 'for' loop control variable
1041	Name redefined
1042	Has to be first statement on the line
1044	Cannot use parentheses when calling a Sub
1045	Expected literal constant
1046	Expected 'In'
1047	Expected 'Class'
1048	Has to be defined inside a class
1049	Expected Let or Set or Get in property declaration
1050	Expected 'Property'
1051	Number of arguments has to be consistent for properties specification
1052	Invalid multiple default properties/methods in a class
1053	No arguments to initialize or terminate class
1054	Property Set or Let requires at least one argument
1055	Unexpected 'Next'
1057	'Default' specification has to also specify 'Public'
1058	'Default' specification can only be on property Get

20.3.11 List of system tags

List of system tags

Tag name	Data type	Description	Range
AckAll	Boolean	Toggle to "1" to acknowledge all alarms "1)"	Server
AckAlr	Boolean	Toggle to "1" to acknowledge the unacknowledged alarm with the highest priority; see "Alarm" below "1)"	Server

Tag name	Data type	Description	Range
Alarm	Character string	The name of the unacknowledged alarm with the highest priority "1)"	Server
AnalogValue_	Real	A simulated analog value that steadily increases from "0" to "96" at the rate of 1.6/sec. If it reaches "96", it starts over at "0".	Local
Beep	Boolean	The current state of the audible alarm: "0" = OFF, "1" = ON "1)"	Server
BeepOff	Boolean	Toggle to "1" to disable the audible alarm "1)"	Server
BlinkFast	Boolean	Continuous, two-second blinking – one second on, one second off	Local
BlinkSlow	Boolean	Continuous, six-second blinking – three seconds on, three seconds off	Local
CrispDisplay	Integer	Legacy tags used by the CrispView custom interface. They are kept in "WinStudio" only for backward compatibility and should not be use in new projects. For more information, please contact technical support	Local
CrispInput	Character string		Local
CrispOutput	Character string		Local
Datum	Character string	The current date on the local station presented as a string in MM/DD/YYYY format. For example, "05/15/2008". (**Note**: This may differ from the current date on the project server; see "ServerDate_" below)	Local
Day	Integer	The current day of the month, between "1" and "31"	Local
DayOfYear	Integer	The current day of the year, between "0" and "365"	Local
DigitalValue_	Boolean	A simulated digital value that alternates between "0" and "1" each second	Local
Goto	Character string	Goto, math function	Local
GroupCNFHiLevel	Integer	End of the current security level range to CNF "3)"	Local
GroupCNFLoLevel	Integer	Start of current security level range to CNF "3)"	Local
GroupHiLevel	Integer	The maximum security level of the current user group "2)" "3)"	Local
GroupLoLevel	Integer	The minimum security level of the current user group "2)" "3)"	Local
GroupName	Character string	The name of the group to which the current user belongs "2)"	Local
Hint	Character string	The hint text for the current object. For further information, refer to Object properties, page 178	Local
Hour	Integer	The current hour of the day, between "0" and "23"	Local
InputMaxRange	Real	The maximum value (Max) for the current tag or object; see "InputOutOfRange" below	Local
InputMinRange	Real	The minimum value (Min) for the current tag or object; see "InputOutOfRange" below	Local
InputOutOfRange	Boolean	This tag toggles to "1" when the user input is outside the Max/Min range of the current tag or object	Local
Label	Character string	Label, math function	Local
LastCodeChar_	Integer	Last code character in the Viewer (???)	Local
LptOff	Boolean	Toggle to "1" to disable the printing of alarms to the default printer "1)"	Local

Tag name	Data type	Description	Range
Minute	Integer	The current minute of the hour, between "0" and "59"	Local
Month	Integer	The current month of the year, between "1" and "12"	Local
Next	Integer	Next, math function	Local
Reserved___1	Boolean	Reserved for future use	Local
Reserved___2	Boolean	Reserved for future use	Local
Reserved___8	Boolean	Reserved for future use	Local
Reserved___9	Boolean	Reserved for future use	Local
Reserved___10	Boolean	Reserved for future use	Local
Second	Integer	The current second of the minute, between "0" and "59"	Local
ServerDate_	Character string	The current date on the project server, presented as a string in MM/DD/YYYY format. For example, "05/15/2008".	Server
ServerTime_	Character string	The current time on the project server, presented as a string in HH:MM:SS format. For example, "23:45:03".	Server
Time	Character string	The current time on the local station, presented as a string in HH:MM:SS format. For example, "23:45:03". (**Note**: This may be different than the current time on the project server; see "ServerTime_" above.)	Local
Tomorrow	Integer	The next day of the month, between "1" and "31"	Local
UserName	Character string	The name of the user that is currently logged in "2)"	Local
Weekday	Integer	The current day of the week on the local station, presented as an integer: "0" = Sunday, "1" = Monday, "2" = Tuesday, "3" = Wednesday, "4" = Thursday, "5" = Friday, "6" = Saturday	Local
Year	Integer	The current, four-digit year (e.g., "2008").	Local
Yesterday	Integer	The previous day of the month, between "1" and "31"	Local

"1)" For further information on configuring alarms, please refer to Alarms folder, page 313.

"2)" For further information on configuring users, groups and security levels, please see Security folder, page 461.

"3)" This system tag is discontinued and only available for reasons of backward compatibility. If the user is assigned to more than one group, this tag has the value -1. Please use the CheckSecurityLevel function instead.

20.4 Tips & tricks

20.4.1 Tips & tricks – Overview

- VBScript editor IntelliSense, page 952
- VBScript compared to VBA, page 954
- Screen events, page 955
- MsgBox() and InputBox() functions, page 956
- Support for ActiveX objects, page 966
- Logical operator NOT, page 961

- Boolean variables, page 959
- Windows CE support, page 966
- Scope and lifetime of variables, page 958
- Declaring variables, page 958
- Creating constants, page 957
- Precedence of VBScript operators, page 960
- Using conditional statements, page 962
- Looping through code, page 963
- VBScript procedures, page 956

20.4.2 VBScript Editor IntelliSense

IntelliSense provides an array of options that make language references easily accessible. When coding, the Code Editor or Immediate Mode command window does not need to be left to perform searches on language elements. The context can be kept to find the required information, insert language elements directly into the code and even have IntelliSense complete typing.

IntelliSense comprises the following options…

List members

A list of valid members from class tags, fields from any tag, properties/methods from an ActiveX object or functions from the integrated scripting language can be displayed. Selecting from the list inserts the member into the code.

Type the $ character on any VBScript interface to automatically open a list box with all tags available for the current project as well as all functions from the integrated scripting language.

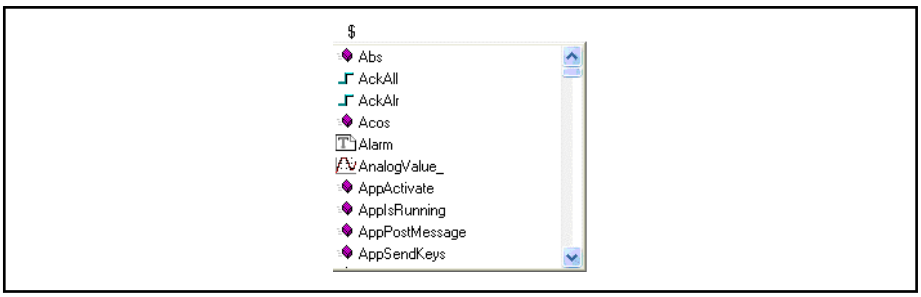


Fig. 20-13: Opening the tag/function list box with the $ character

By entering the name of a class tag followed by a point (.) into any VBScript interface, a list box with all members from the class tags is opened automatically:

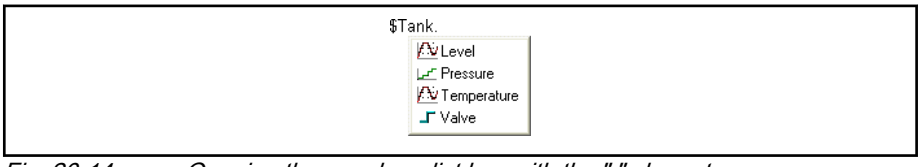


Fig. 20-14: Opening the members list box with the "." character

By entering the name of a tag followed by a hyphen and a greater than character (->) into any VBScript interface, a list box with all fields available for this tag is opened automatically:

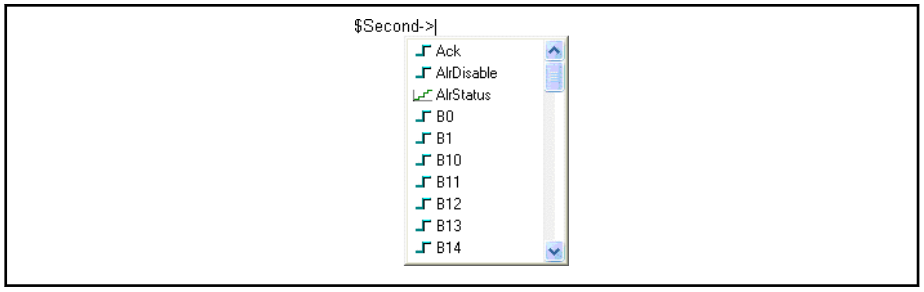


Fig. 20-15: Opening the list box with all fields available for a tag with "->"

The elements are displayed in alphabetic order and each element has one of the following icons identifying its main type:

Symbol	Type
	Boolean tag
	Integer tag
	Real tag
	String tag
	Class tag
	Function from the integrated scripting language

By entering the name of an ActiveX control element followed by a point (.) into any VBScript interface from the screen inserted into the ActiveX object, a list box with all properties and methods of this object is automatically opened:

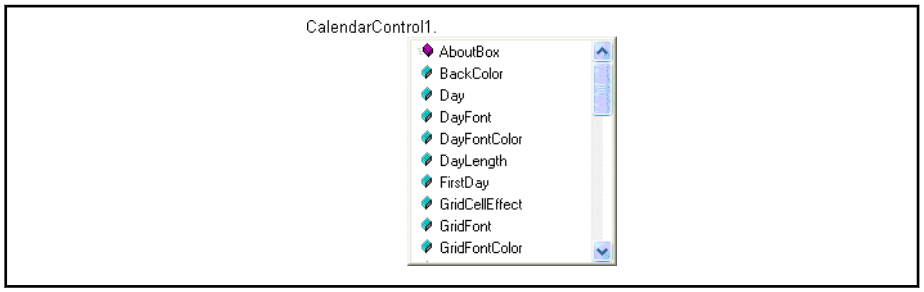


Fig. 20-16: Opening the properties and methods list box with the "." character

The elements are displayed in alphabetic order and each element has one of the following icons identifying its main type:

Symbol	Type
	Property from the ActiveX object
	Method from the ActiveX object

Parameter Quick info

The Quick info option displays pop-up boxes with information about the functions from the integrated scripting language. The information includes all parameters supported by this function with the currently configured one in bold text.

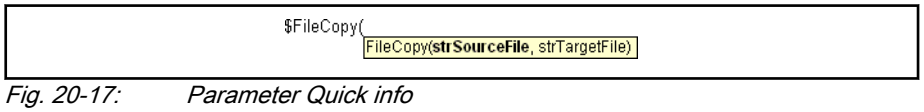


Fig. 20-17: Parameter Quick info

Complete word

Complete word finishes a tag, member, field, function or ActiveX property/method name once enough unique characters for this term are entered. After

entering the first few characters of the name, it can be completed automatically by pressing <Ctrl>+<Space>.

20.4.3 VBScript compared to VBA

While VBScript and Visual Basic for Applications (VBA) are similar and are both based on the Visual Basic standard language, there are advantages to using VBScript for "WinStudio" users:

- VBScript is supported by the Microsoft Windows Embedded operating system and VBA is not.
- VBScript brings active scripting to a wide variety of environments including web client scripting in Microsoft Internet Explorer. This prevents operations that may present risks for Thin Client users such as direct access to local files.
- VBScript was designed to be simple and easy to learn with some VBA standards simplified in VBScript to make it more straightforward. For example, in VBScript, users do not have to worry about the type of each variable when declaring them as VBScript automatically assumes the proper type for each variable.

The following table lists VBScript features that are not offered by VBA.

Category	Feature/Keyword
Declarations	Class
Miscellaneous	Eval
	Execute
Objects	RegExp
Script Engine Identification	ScriptEngine
	ScriptEngineBuildVersion
	ScriptEngineMajorVersion

The following table lists VBA features that are not offered by VBScript.

Category	Not Included Feature/Keyword
Array handling	Option Base
	Declaring arrays with lower limit <> 0
Collection	Add, Count, Item, Remove
	Access to collections using ! characters
Conditional compilation	#Const
	#If...Then...#Else
Control flow	DoEvents
	GoSub...Return, GoTo
	On Error GoTo
	On...GoSub, On...GoTo
	Line numbers, line labels
Conversion	CVar, CVDate
	Str, Val

Category	Not Included Feature/Keyword
Data types	All intrinsic data types except Variant
	Type...End Type
Date/Time	Date statement, time statement
DDE	LinkExecute, LinkPoke, LinkRequest, LinkSend
Troubleshooting	Debug.Print
	End, Stop
Declaration	Declare (for declaring DLLs)
	Optional
	ParamArray
	Static
Error handling	Erl
	Error
	Resume, Resume Next
File Input/Output	All conventional basic files I/O
Financial	All financial functions
Object handling	TypeOf
Objects	Clipboard
	Collection
Operators	Like
Options	Deftype
	Option Base
	Option Compare
	Option Private Module
Select case	Expressions containing the "Is" keyword or any comparison operators
	Expressions containing a range of values using the "To" keyword
Strings	Fixed-length Strings
	LSet, RSet
	Mid Statement
	StrConv
Using objects	Collection access using !

20.4.4 Screen Events

In addition to the screen script, logics can be configured using the integrated scripting language for the On Open, While Open and On Close events for the screen (refer to the screen logic interface from the Screen Attributes dialog). For configuration of the screen script (VBScript language) and the screen logic (integrated scripting language), "WinStudio" respects the following order of execution:

Event	Order of execution
On opening the screen	• Screen_OnOpen() sub-routine from the screen script interface (VBScript language) • On opening from the screen logics interface (internal scripting language)
On closing the screen	• On closing from the screen logics interface (internal scripting language) • Screen_OnClose() sub-routine from the screen script interface (VBScript language)

20.4.5 MsgBox and InputBox Functions

The **MsgBox()** and **InputBox()** functions of the VBScript language enable displaying of pop-up messages during runtime. These functions are synchronous. If one of these functions is executed, the remaining instructions from the code are not executed before the pop-up messages launched by the function are closed.



The text displayed in these pop-up messages is not affected by the translation tool of "WinStudio", unless the text is explicitly configured using the $Ext() function from the integrated scripting language.

20.4.6 VBScript Procedures

In VBScript, there are two kinds of procedures: Sub procedures and Function procedures.

Sub procedures

Sub procedures are series of VBScript statements (enclosed by **Sub** and **End Sub** statements) that perform actions, however, do not return a value. Sub procedures can contain arguments (constants, variables or expressions that are transferred by a calling procedure). If a Sub procedure has no arguments, its **Sub** statement has to include an empty set of parentheses ().

The following Sub procedure uses two intrinsic (integrated) VBScript functions, MsgBox and InputBox on page 956 to prompt a user input. Afterwards, the results of a calculation based on that information are displayed. The calculation is performed in a Function procedure created with VBScript. The Function procedure is displayed after the following discussion.

Sub procedure

```
Sub ConvertTemp()
  temp = InputBox("Please enter the temperature in degrees F.", 1)
  MsgBox "The temperature is " & Celsius(temp) & " degrees C."
End Sub
```

Function procedures

Function procedures are a series of VBScript statements enclosed by the **Function** and **End Function** statements. Function procedures are similar to Sub procedures, however, they can also return a value. Function procedures can contain arguments (constants, variables or expressions that are transferred to it by a calling procedure).

If a Function procedure has no arguments, its **Function** statement has to include an empty set of parentheses ().

A function returns a value by assigning a value to the function name in one or more statements of the procedure. The return value of a function is always a variant.

In the following example, the **Celsius** function calculates degrees Celsius from degrees Fahrenheit. If the function is called from the **ConvertTemp** Sub procedure, a variable containing the argument value is transferred to the

function. The result of the calculation is returned to the calling procedure and displayed in a message box.

Function procedure

```
Sub ConvertTemp()
  temp = InputBox("Please enter the temperature in degrees F.", 1)
  MsgBox "The temperature is " & Celsius(temp) & " degrees C."
End Sub

Function Celsius(fDegrees)
  Celsius = (fDegrees - 32) * 5 / 9
End Function
```

Input and output of procedure data

Each data element is passed into the procedures using an argument. Arguments serve as placeholders for data to be passed into procedures. Arguments can be assigned any valid variable name. If a procedure is created using either the **Sub** statement or the **Function** statement, parentheses have to be included after the name of the procedure. Any arguments are placed inside these parentheses, separated by commas. In the following example, **fDegrees** is a placeholder for the value being passed into the **Celsius** function for conversion.

Input and output of procedure data

```
Function Celsius(fDegrees)
  Celsius = (fDegrees - 32) * 5 / 9
End Function
```

To receive data from a procedure, functions have to be used. Function procedures can return values while Sub procedures cannot.

Using Sub and Function procedures in code

Functions in codes have to be always used on the right side of variable assignments or in expressions. For example:

Using Function procedures in code

```
Temp = Celsius(fDegrees)
 // or
MsgBox "The Celsius temperature is " & Celsius(fDegrees) & " degrees."
```

To call a Sub procedure from another procedure, enter the name of the procedure along with values for any required argument, each separated by a comma. The **Call** statement is not required. However, if it is used, all arguments have to be enclosed in parentheses.

The following example illustrates two calls for the **MyProc** procedure. One uses the **Call** statement in code, the other does not. Both have exactly the same effect.

Using Sub procedures in code

```
Call MyProc(firstarg, secondarg)

MyProc firstarg, secondarg
```

Please note that the parentheses are not specified in the call if the **Call** statement is not used.

20.4.7 Creating Constants

Constants are convenient names that replace numbers or Strings and never change. VBScript defines a number of intrinsic constants.

User-defined constants are created in VBScript using the Const statement.

By means of the Const statement, String or numeric constants with meaningful names can be created and get literal values assigned to them.

For example:

```
Const MyString = "This is my text."
Const MyAge = 49
```

Please note that the string is enclosed in quotation marks (" "). Quotation marks are the most obvious way to differentiate String values from numeric values. Date literals and time literals can be specified by enclosing them in number signs ("#").

For example:

```
Const EndDate = #6-1-97#
```

It is useful to apply a naming scheme to differentiate constants from variables. This helps to prevent reassignment of constant values while the script is running. For example, "vb" or "con" can be used as prefixes for constant names or constants may be named in capital letters. Clearly differentiating constants from variables helps to eliminate confusion when developing more complex scripts.

20.4.8 Declaring Variables

Variables are convenient placeholders that refer to a storage location on the computer where program information that may change during execution of the script can be stored. In VBScript, variables are always of the basic data type Variant.

Variables are explicitly declared in the script using the Dim statement, the Public statement, and the Private statement.

For example:

```
Dim DegreesFahrenheit
```

Multiple variables are declared by separating each variable name with a comma.

For example:

```
Dim Top, Bottom, Left, Right
```

A variable can also be declared implicitly by simply using its name in the script. In general, this is not good practice as the variable name could be misspelled in one or several places causing unexpected results executing the script. For this reason, the option Explicit statement is configured by default in the global procedures interface to require explicit declaration of all variables. Unless this statement is deleted, all variables have to be declared explicitly. Otherwise, VBScript generates errors during runtime indicating that the variable does not exist.

In an expression, the variable is to stand on the left and the value to be assigned to the variable on the right side.

For example:

```
MyVar = 100
```

20.4.9 Scope and Lifetime of Variables

The scope of a variable is determined by where it is declared. If a variable is declared within a procedure, only code within that procedure can access or change the value of the variable. It has local scope and is a procedure-level variable. If a variable is declared outside a procedure, it can be recognized by all procedures in the script. This is a script-level variable with script-level scope.

The lifetime of a variable depends on how long it exists. The lifetime of a script-level variable extends from the time it is declared until the time the script is finished running. At procedure level, a variable exists only as the pro-

cedure runs. The variable is destroyed on exiting the procedure. Local variables are ideal as temporary storage space when a procedure is running. Local variables with the same name can be declared to several different procedures as they are only recognized by the respective procedure in which they are declared.

20.4.10 Boolean tags and Boolean variables

By default, Boolean variables in VBScript are handled differently than Boolean, tags page 111 in "WinStudio". The Boolean states FALSE and TRUE have the same meaning, however, the numeric values of TRUE are different as illustrated in the table below.

Boolean state	Numeric value in…	
	Project tag	VBScript variable
FALSE	0	0
TRUE	1 (or not zero)	−1



In VBScript, "FALSE" and "TRUE" are reserved as keywords on page 946.

This difference in how Booleans are handled can seriously affect runtime behavior if VBScript is used in the project. Logical and arithmetic operations – especially the NOT operator on page 961 – can change tag values in unexpected ways.

To prevent this kind of problem and to ensure that Boolean tags have correct values at any time, the VBScript interface in Rexroth WinStudio was modified to process Boolean tags and treat them as Boolean variables. In other words: Even though a Boolean tag has an actual value of 1 in the tag database, the VBScript treats it as if it had a value of -1 to enable you to perform operations with the help of Boolean tags as well as Boolean variables.

This change was introduced in an earlier version of Rexroth WinStudio - Rexroth WinStudio v7.1. Projects which were developed using earlier versions of Rexroth WinStudio and were then upgraded to the newest version, were not changed to maintain backward compatibility. If your project was developed using an earlier version of Rexroth WinStudio, make the change yourself by manually editing your project file (`project name.APP`) and adding the following entry:

```
[Script]
VBBoolean=1
```

After that, however, test your project thoroughly to ensure that your VBScript code still behaves as expected.



Custom property on a linked symbol on page 223 using the `#Label:@Pointer` syntax are to be defined with care.

Example:

```
'The following statements are valid
If $MyBoolean = 1 Then
End If

If $MyBoolean = True Then
End If

If #Mne:@MyPointer = True Then
End If
```

```
'The following statement is invalid
If #Mne:@MyPointer = 1 Then
End If
```

20.4.11 Writing Real values to Integer tags

By default, a Real (i.e., floating point) value is cut off at the decimal point when it is written to an Integer tag on page 111. This behavior is the same in both the integrated scripting language and in VBScript.

However, this behavior can be changed in VBScript by disabling the "TruncRealToInt" runtime setting. To do so, change the following line in the project file (**project_name.app**):

[Script]

TruncRealToInt=value

If **TruncRealToInt** is set to "1", the project behaves as described above:

Real values are truncated at the decimal point without rounding. (For example: a value of "5.56" is written as "5" to an Integer tag.) This is the default setting for projects created with "Rexroth WinStudio v7.1" or earlier and then updated to "Rexroth WinStudio v7.2" in order to maintain backward compatibility.

If "TruncRealToInt" is set to "0", VBScript functions and operations round Real values to the nearest integer.

(For example: a value of "5.56" is written as "6" to an Integer tag.) This is the default setting for projects created with "Rexroth WinStudio v7.2" or later.



This setting only affects the behavior of VBScript in "WinStudio". The behavior of the integrated scripting language is not affected.

20.4.12 Precedence of VBScript Operators

VBScript has a full range of operators, including arithmetic operators, comparison operators, concatenation operators and logical operators.

If several operations occur in an expression, each part is evaluated and resolved in a predetermined order called "operator precedence." Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed first. Within the parentheses, however, standard operator precedence is maintained.

If expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next and logical operators are evaluated last. All comparison operators have equal precedence, i.e., they are evaluated in a left-to-right order. Arithmetic and logical operators are evaluated in the following order of precedence.

Arithmetic	Comparison	Logical
Negation (-)	Equality (=)	Not
Exponentiation (^)	Inequality (<>)	And
Multiplication and division (*, /)	Less than (<)	Or
Integer division (\)	Greater than (>)	Xor
Modulus arithmetic (Mod)	Less than or equal to (<=)	Eqv

Arithmetic	Comparison	Logical
Addition and subtraction (+, -)	Greater than or equal to (>=)	Imp
String concatenation (&)	Is	&

If multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, if addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

The string concatenation (&) operator is not an arithmetic operator. In precedence, however, it is set after all arithmetic operators and before all comparison operators. The Is operator is an object reference comparison operator. It does not compare objects or their values. It checks only whether two object references refer to the same object.

20.4.13 Logical operator NOT

The logical operator NOT behaves differently in VBScript than it does in the integrated scripting language.

NOT operator in VBScript

In VBScript, the NOT operator inverts the bits of a given numeric value producing its complement number according to the "two's complement" system of signed numbers that is used by computers. The following table illustrates the behavior of the NOT operator in VBScript for the syntax…

Result = NOT expression

If "expression" is…	Then "result" is…
−3	2
−2	1
−1	0
0	−1
1	−2
2	−3
3	−4



If a numeric value not equal to 0 is to be written to a Boolean tag on page 111, the tag is set to the value 1 by default.

Thus, if the NOT operator of VBScript is applied to a Boolean tag with the value 1, the tag value does not change.

The operator returns the value -2. The tag can, however, not accept this value and is reset to 1. "WinStudio" can be configured to handle Boolean tags in VBScript similar to Boolean variables to ensure that the NOT operator in VBScript works as expected. For further information, please refer to Boolean tags and Boolean variables on page 959.

NOT operator in integrated language

In contrast, the NOT operator on page 611 in the integrated scripting language toggles the given numeric value as if it is a natural Boolean value. The following table illustrates the behavior of the NOT operator in the integrated scripting language for the syntax…

Result = NOT expression

If "expression" is…	Then "result" is…
0	1
≠0	0

20.4.14 Using Conditional Statements

The flow of scripts can be controlled with conditional and looping statements. Using conditional statements, VBScript code can be written that makes decisions and repeats actions. The following conditional statements are available in VBScript:

- **If…Then…Else** statement
- **Select Case** statement

Making decisions using If… Then…Else

The **If…Then…Else** statement is used to evaluate whether a condition is **True** or **False** and, depending on the result, to specify one or more statements to run. Usually, the condition is an expression that uses a comparison operator to compare one value or variable with another. For information about comparison operators, refer to Comparison operators.

If…Then…Else statements can be nested to as many levels as required.

Running statements if a condition is True

To run only one statement if a condition is True, the single-line syntax is used for the **If…Then…Else** statement. The following example illustrates the single-line syntax. Please note that this example does not include the **Else** keyword:

*Single-line syntax for the **If…Then…Else***

```
Sub FixDate()
  Dim myDate
  myDate = #2/13/95#
  If myDate < Now Then myDate = Now
End Sub
```

To run more than one line of code, the multiple-line (or block) syntax is to be used. This syntax includes the **End If** statement, as illustrated in the following example:

Multiple-line (or block) syntax

```
Sub AlertUser(value)
  If value = 0 Then
    AlertLabel.ForeColor = vbRed
    AlertLabel.Font.Bold = True
    AlertLabel.Font.Italic = True
  End If
End Sub
```

Running certain statements if a condition is True and running others if a condition is False

A **If…Then…Else** statement can be used to define two blocks of executable statements: One block to run if the condition is **True** and the other block to run if the condition is **False**:

If…Then…Else statement to define two blocks...

```
Sub AlertUser(value)
  If value = 0 Then
    AlertLabel.ForeColor = vbRed
    AlertLabel.Font.Bold = True
    AlertLabel.Font.Italic = True
  Else
    AlertLabel.Forecolor = vbBlack
    AlertLabel.Font.Bold = False
```

```
    AlertLabel.Font.Italic = False
  End If
End Sub
```

Deciding between several alternatives

A variation on the **If…Then…Else** statement allows selecting between several alternatives. Adding **ElseIf** clauses expands the functionality of the **If… Then…Else** statement. This way, the program flow can be controlled based on different possibilities. For example:

If…Then…Else statement with ElseIf

```
Sub ReportValue(value)
  If value = 0 Then
    MsgBox value
  ElseIf value = 1 Then
    MsgBox value
  ElseIf value = 2 then
    Msgbox value
  Else
    Msgbox "Value out of range!"
  End If
End Sub
```

ElseIf clauses can be added as required to provide alternative choices. However, extensive use of **ElseIf** clauses often becomes cumbersome. A better way to choose between several alternatives is the **Select Case** statement.

Making decisions with Select Case

The "Select Case" structure provides an alternative to **If…Then…ElseIf** for selectively executing one block of statements from multiple blocks of statements. A **Select Case** statement provides capability similar to the **If…Then… Else** statement. However, it makes the code more efficient and readable.

A **Select Case** structure works with a single test expression that is evaluated once at the top of the structure. The result of the expression is then compared to the values for each **Case** in the structure. If there is a match, the block of statements associated with that **Case** is executed, as in the following example:

Select Case structure

```
Select Case Document.Form1.CardType.Options(SelectedIndex).Text
  Case "MasterCard"
    DisplayMCLogo
    ValidateMCAccount
  Case "Visa"
    DisplayVisaLogo
    ValidateVisaAccount
  Case "American Express"
    DisplayAMEXCOLogo
    ValidateAMEXCOAccount
  Case Else
    DisplayUnknownImage
    PromptAgain
End Select
```

Please note that the **Select Case** structure evaluates an expression once at the top of the structure. In contrast, the **If…Then…ElseIf** structure can evaluate a different expression for each "ElseIf" statement. A **If…Then…ElseIf** structure can only be replaced by a **Select Case** structure if each **ElseIf** statement evaluates the same expression.

20.4.15 Looping with Code

Looping enables running a group of statements repeatedly. Some loops repeat statements until a condition is False. Other loops repeat statements until a condition is True. There are also loops that repeat statements a specific number of times.

The following looping statements are available in VBScript:

- **Do…Loop**: Loops while or until a condition is True
- **While…Wend**: Loops while a condition is True
- **For…Next**: Using a counter to run statements a specified number of times

Using Do Loops

"Do…Loop" statements can be used to run a block of statements an indefinite number of times. The statements are repeated either while a condition is True or until a condition becomes True.

Repeating statements while a condition is True

The While keyword can be used to check a condition in a **Do…Loop** statement. The condition can be checked before entering the loop (as shown in the following **ChkFirstWhile** example) or after the loop has run at least once (as shown in the **ChkLastWhile** example). In the **ChkFirstWhile** procedure, if **myNum** is set to 9 instead of 20, the statements inside the loop are never executed. In the **ChkLastWhile** procedure, the statements inside the loop run only once as the condition is already False.

Repeating statements while a condition is True

```
Sub ChkFirstWhile()
  Dim counter, myNum
  counter = 0
  myNum = 20
  Do While myNum > 10
    myNum = myNum - 1
    counter = counter + 1
  Loop
  MsgBox "The loop made " & counter & " repetitions."
End Sub

Sub ChkLastWhile()
  Dim counter, myNum
  counter = 0
  myNum = 9
  Do
    myNum = myNum - 1
    counter = counter + 1
  Loop While myNum > 10
  MsgBox "The loop made " & counter & " repetitions."
End Sub
```

Repeating a statement until a condition becomes True

There are two ways to use the **Until** keyword to check a condition in a **Do…Loop** statement. The condition can be checked before entering the loop (as shown in the following **ChkFirstUntil** example) or after the loop has run at least once (as shown in the **ChkLastUntil** example). As long as the condition is **False**, the looping occurs.

Repeating a statement until a condition becomes True

```
Sub ChkFirstUntil()
  Dim counter, myNum
  counter = 0
  myNum = 20
  Do Until myNum = 10
    myNum = myNum - 1
    counter = counter + 1
  Loop
  MsgBox "The loop made " & counter & " repetitions."
End Sub

Sub ChkLastUntil()
  Dim counter, myNum
  counter = 0
  myNum = 1
  Do
    myNum = myNum + 1
    counter = counter + 1
```

```
  Loop Until myNum = 10
  MsgBox "The loop made " & counter & " repetitions."
End Sub
```

Exiting a Do…Loop statement from inside the loop

A **Do…Loop** can be exited by using the **Exit Do** statement. As loops are typically only exited in certain situations, for example, to avoid an endless loop, the **Exit Do** statement is to be used in the **True** statement block of an **If… Then…Else** statement. If the condition is **False**, the loop runs as usual.

In the following example, "myNum" is assigned a value that creates an endless loop. The **If…Then…Else** statement checks for this condition, preventing the endless repetition.

Exiting a Do…Loop statement from inside the loop

```
Sub ExitExample()
  Dim counter, myNum
  counter = 0
  myNum = 9
  Do Until myNum = 10
    myNum = myNum - 1
    counter = counter + 1
    If myNum < 10 Then Exit Do
  Loop
  MsgBox "The loop made " & counter & " repetitions."
End Sub
```

Using While…Wend

The **While…Wend** statement is provided in VBScript for experienced users. However, due to the lack of flexibility of **While…Wend**, it is recommended to use **Do…Loop** instead.

Using For…Next

For…Next statements can be used to run a block of statements a specific number of times. For loops, a counter variable is to be used whose value increases or decreases with each repetition of the loop.

The following example causes a procedure called **MyProc** to execute 50 times. The For statement specifies the counter variable **x** and its start and end values. The "Next" statement increments the counter variable by 1.

Using For…Next (1)

```
Sub DoMyProc50Times()
  Dim x
  For x = 1 To 50
    MyProc
  Next
End Sub
```

Using the **Step** keyword, the counter variable can be increased or decreased by the specified value. In the following example, the counter variable **j** is increased by 2 each time the loop repeats. If the loop is finished, the total is the sum of 2, 4, 6, 8, and 10.

Using For…Next (2)

```
Sub TwosTotal()
  Dim j, total
  For j = 2 To 10 Step 2
    total = total + j
  Next
  MsgBox "The total is " & total
End Sub
```

To decrease the counter variable, a negative **Step** value is to be used. An end value that is less than the start value is to be specified. In the following example, the counter variable **myNum** is decreased by 2 each time the loop repeats. If the loop is finished, the total is the sum of 16, 14, 12, 10, 8, 6, 4, and 2.

Using For…Next (3)

```
Sub NewTotal()
  Dim myNum, total
  For myNum = 16 To 2 Step -2
    total = total + myNum
  Next
  MsgBox "The total is " & total
End Sub
```

The **For…Next** statement can be exited at any time before the counter reaches its end value by using the **Exit For** statement. As loops are typically only exited in certain situations, for example, in case of an error, the **Exit For** statement is to be used in the **True** statement block of an **If…Then…Else** statement. If the condition is **False**, the loop runs as usual.

20.4.16 Support for ActiveX Control Elements

Using the VBScript interfaces for the Graphic module (graphic script, screen script, command animation and ActiveX events), this syntax can be applied to access properties and methods directly from any ActiveX control object in the screen, in which the object is configured.

"WinStudio" assigns a unique name to the object on the screen. The Name property can be used (in the **Object Properties** dialog) to modify this name.

After inserting an ActiveX control object on the screen, the properties and methods of this object can be accessed from any VBScript interface associated with this screen. Use the syntax **Object_Name.Properties_or_Method_Name**.

Examples:

```
//Access the value of the property Day from the CalendarControl1 ActiveX object
CalendarControl1.Day

//Triggers the method AboutBox from the CalendarControl1 ActiveX object
CalendarControl1.AboutBox
```

20.4.17 Windows Embedded Support

CEView also supports VBScript. The hardware manufacturer of the Microsoft Windows Embedded device has to enable support of VBScript on the device to enable CEView to execute the scripts configured in the VBScript language on the device.

The **MsgBox()** and **InputBox()** functions can be specifically enabled/disabled by the hardware manufacturer when the image for the Windows Embedded device is created.

In case of doubt whether the image loaded on the device supports VBScript, please contact the hardware manufacturer.

20.5 Troubleshooting VBScript

20.5.1 Troubleshooting VBScript, General information

The development application provides additional tools for troubleshooting in your VBScript code.



These tools are based on debugging tools in Microsoft Visual Studio. This means that if you have experience with Visual Studio, you should be familiar with these tools.

The tools are available in the following VBScript interfaces in the development application (VBScript interfaces):

- Procedures on the **Global** tab of the Project Explorer;
- Graphic script on the **Graphics** tab of the Project Explorer;
- Script groups including the startup script on the **Tasks** tab of the Project Explorer; and
- Screen script which is attached to each screen worksheet.

At the moment, the following VBScript interfaces do not support troubleshooting:

- Command animation, which is part of every button object, and
- ActiveX and .NET control objects.

Generally speaking, you can debug your code by running your project in **Debug** mode (in contrast to the common **Run** mode) and observing how the values of the project tags, locally declared variables and complete functions change if you step through the code. It is possible to control the steps using the **Debug** tab of the ribbon and to monitor the changing values in the Database spy window.

Also refer to:

- The Debug tab on page 967. Shows the troubleshooting tools in the VBScript code of your project.
- Setting breakpoints in your VBScript code on page 968, set breakpoints in your VBScript code to indicate where the runtime is to be stopped.
- Running the project in Debug mode on page 970, let your project run in Debug mode to use the other debugging tools.
- Monitor the current status in the Database spy window on page 972, use the Database spy window to monitor the current state of the project if it is in break mode.
- Stepping through your VBScript code on page 973, use the tools **Step into**, **Step over** and **Step out** to step through your VBScript code line by line.

20.5.2 The Debug tab

The development application provides additional tools for troubleshooting in your VBScript code (**WinStudio** ▸ **Debug application**)

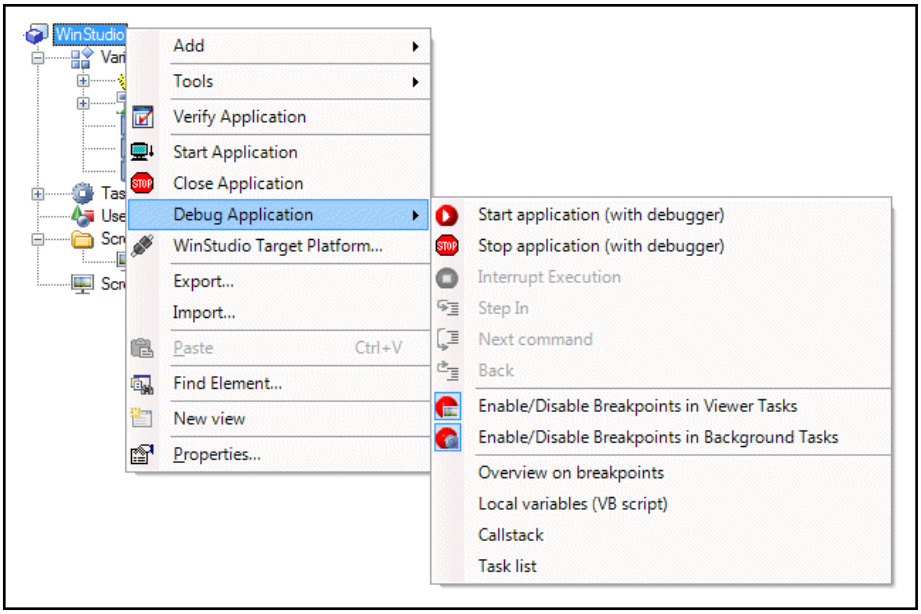


Fig. 20-18: Debugging tools

See also

- Troubleshooting VBScript on page 966
- WinStudio Debug Functions Toolbar on page 44

20.5.3 Setting breakpoints in your VBScript code

Set breakpoints in your VBScript code to indicate where the runtime is to be stopped.

While developing the code, you will identify important sections on which to concentrate during troubleshooting. If your code for example contains an important function which updates the values of several project tags, you may want to interrupt the execution at the beginning of this function to directly observe the value changes. Set a breakpoint here.

A breakpoint is a signal which commands the debugger to temporarily interrupt the execution of your project. If the execution is interrupted at a breakpoint, it means that your project is in break mode. The break mode does not stop or finish the execution of your project. The execution can be resumed at any time.

Consider the break mode as a timeout. All elements are maintained (functions, variables and objects, for example, remain in the memory) but their movement and activities are interrupted. In break mode, it is possible to check the positions and states of the elements to find violations and errors. In addition, you can change the project during break mode, for example the value of a variable.

Breakpoints are a powerful tool which enables you to interrupt the execution wherever and whenever necessary. Instead of going through your code line by line, it is possible to let your project run until it reaches a breakpoint and then start troubleshooting. This accelerates the troubleshooting process. Without this option, it would be almost impossible to debug a large project.

Setting one or several breakpoints in your VBScript code:

1. Open the VBScript worksheet you want to debug. The worksheet must be in one of the VBScript interfaces which support troubleshooting. For further information, refer to Troubleshooting VBScript, General information on page 966.

The worksheet is opened for editing.

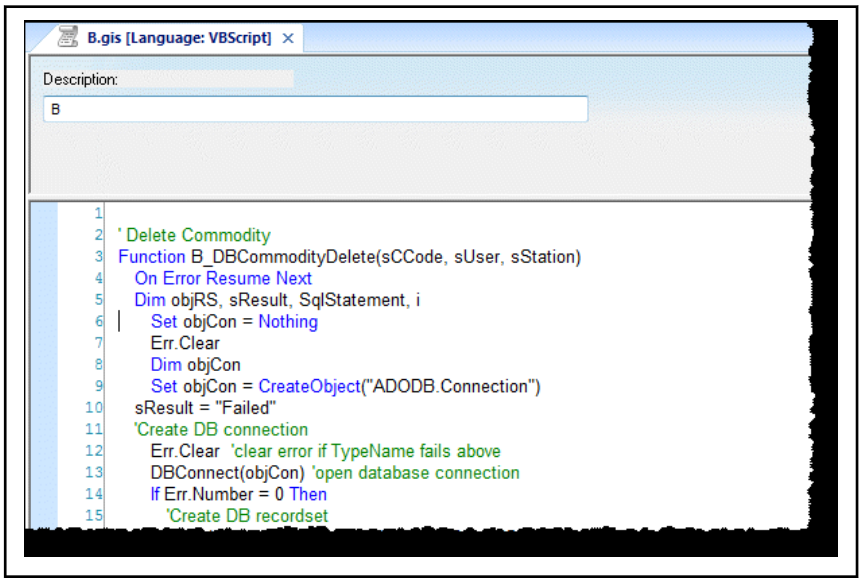


Fig. 20-19: Example for a VBScript interface

2. Find the code line on the worksheet in which you want to insert a breakpoint.

Tip: To show/hide the line numbers in the VBScript interface select the line number on the View tab of the ribbon.

The option is set independently for each worksheet. There is no global setting.

---

3. *Do one of the following:*
   - Click on the light blue gap between the columns on the left of the relevant line or
   - Click on the line to position the cursor and then click on **Breakpoint** in the ribbon.

   A red breakpoint icon is inserted. If it is not possible to insert a breakpoint at the place you clicked - for example if it is a Dim statement which does not really change any values - then it is automatically inserted in the next possible line.

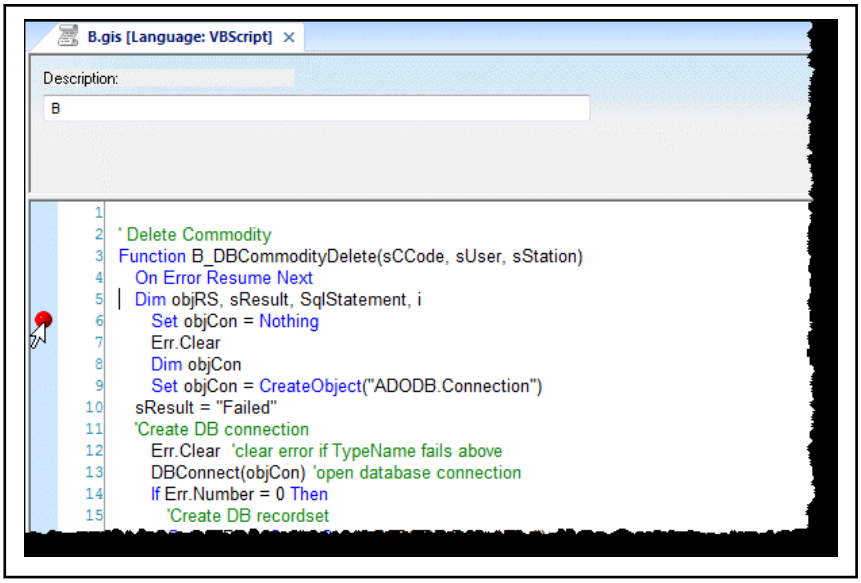


Fig. 20-20: Setting a breakpoint

4. Repeat these steps for each breakpoint you want to set.
5. Save and close all open VBScript worksheets.

   All breakpoints you set are saved together with the worksheet(s).

20.5.4 Running the project in Debug mode

Run the project in Debug mode to use the other debugging tools.

A project can either be executed in the normal execution mode or in debug mode. You have to run your project in debug mode to be able to use the other debugging tools such as breakpoints, stepping and the specific VBScript tabs in the database spy window.

Running the project in debug mode:

1. On the **Debug** tab of the ribbon, in the **Local Management** group, click **Debug**. The project runtime is started in debug mode. If the **Debug viewer** option (on the **Debug** tab in the **Debug options** group) is selected and the Viewer Task is started, the Secure Viewer is started as well.



If a project is executed locally, the Standard Viewer is usually used. If a project is executed in debug mode, however, the Secure Viewer is used for better thread and process management.

The VBScript code is executed until the first breakpoint set by you is reached if you have set breakpoints. (The "first" breakpoint is the first in the logical, sequential execution of the code, not the first one you set.)

At this point, the project automatically changes to break mode: the execution is interrupted and a yellow arrow is displayed in the VBScript interface which indicates where exactly in the code the execution was interrupted.

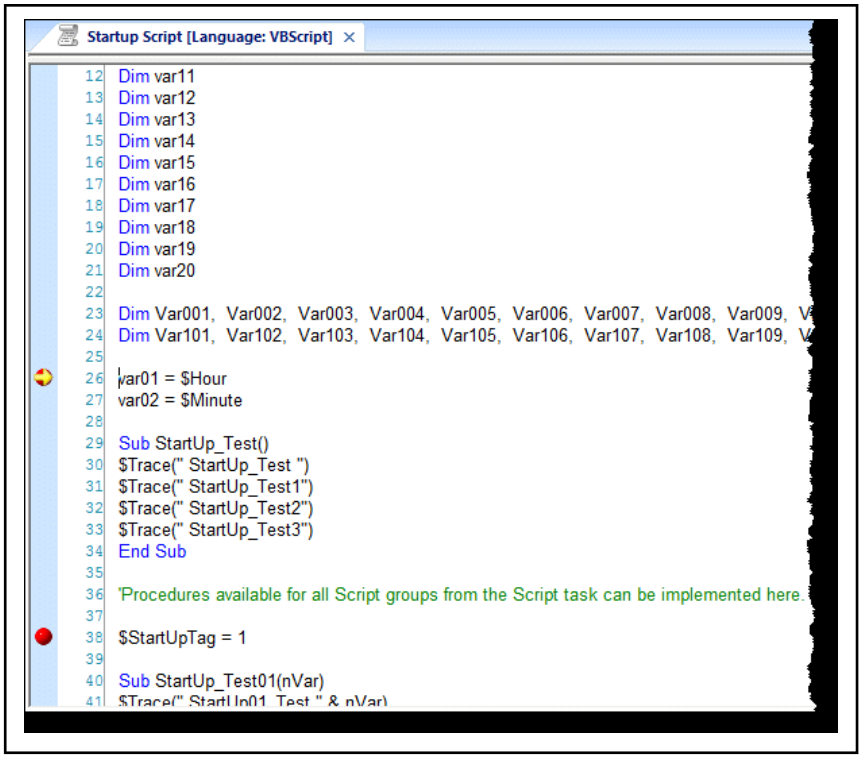


Fig. 20-21: The yellow arrow shows where the execution was interrupted

If no breakpoints were set, the project will continue running either until you click **Stop** (to stop the project runtime) or **Break** (to manually activate the break mode) on the **Debug** tab.

2. *If the project is in break mode, select one of the following methods:*
   - Click **Next** on the **Debug** tab to resume the execution and continue until the next breakpoint is reached.
   - In the Database spy window, the current state of the project is displayed (Monitor the current status in the Database spy window on page 972).
   - Click on Step into, Step over and Step out on the Debug tab to step through the code line by line. (Stepping through your VBScript code on page 973).

☞ If your project is in break mode, move the cursor over any available VBScript variable in the code to view the current value of this variable. The value is displayed in a popup window.

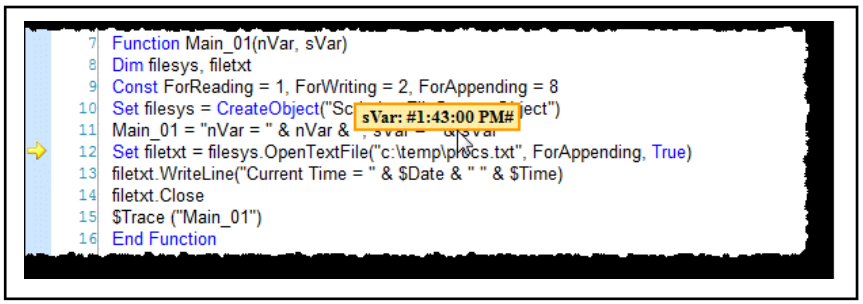


Fig. 20-22: Current value of the VBScript variable in a popup window

3. If applicable, repeat the previous step.

4. If you have finished troubleshooting your code, click on **Stop** on the **Debug** tab.

NOTICE **Running the project in debug mode**

If you run a project in debug mode, you cannot stop it before the startup script of the project has been executed completely.

20.5.5 Monitor the current status in the Database spy window

Use the Database spy window to monitor the current status of the project if it is in break mode.

To be able to use these tools, your project must already be in break mode, either because it reached a previously set breakpoint or by using the break tool for manual activation of the break mode.

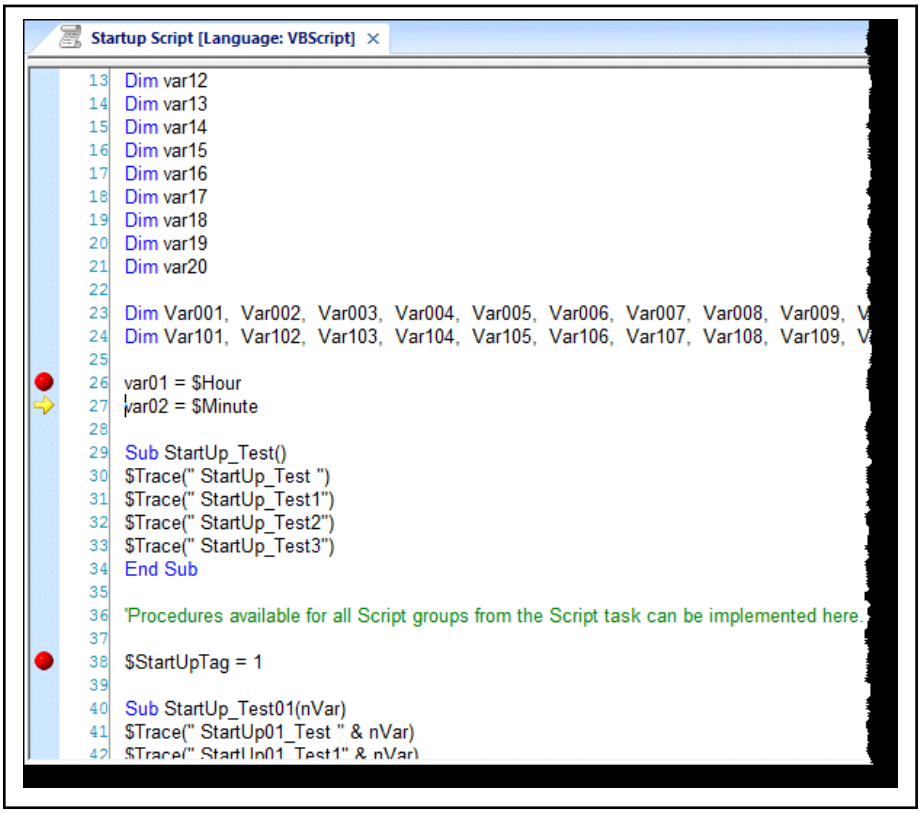


Fig. 20-23: Project in break mode. The yellow arrow shows where the execution was interrupted.

You can enter the names of the VBScript functions (but not of the subroutines) in the Database spy window, to get the return values of these functions.

In addition, there are three new tabs which only work during VBScript troubleshooting:

Local On this tab, all locally declared VBScript variables as well as the current values of these variables are shown.

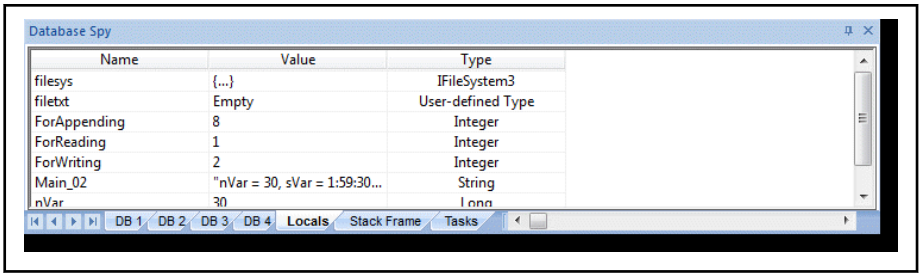


Fig. 20-24: The Local tab of the Database spy window

Stack frame

On this tab, additional information on the VBScript interfaces which are currently opened for troubleshooting are shown. The interfaces are sorted in the sequence of their most recent execution so that you can see how one script calls functions in another script. In the **Line** column, the most recently executed lines for each interface are displayed.

The screen shot below therefore shows that the execution was interrupted at line 24 in the `Main_02` function in the Global procedures and that the Main_02 function was most recently called at line 97 of the startup script.

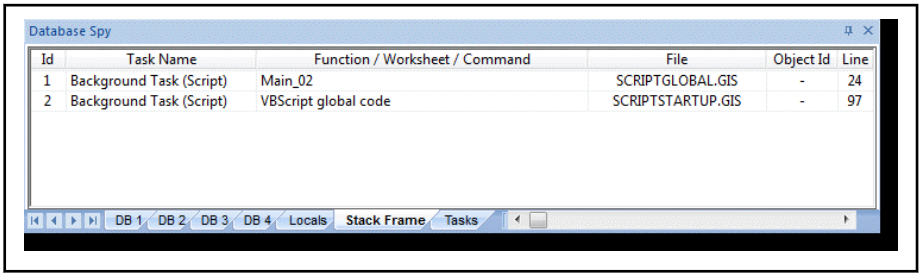


Fig. 20-25: The Stack frame tab of the Database spy window

Tasks

On this tab, all runtime tasks which are currently being debugged are displayed as well as the current status of each task.

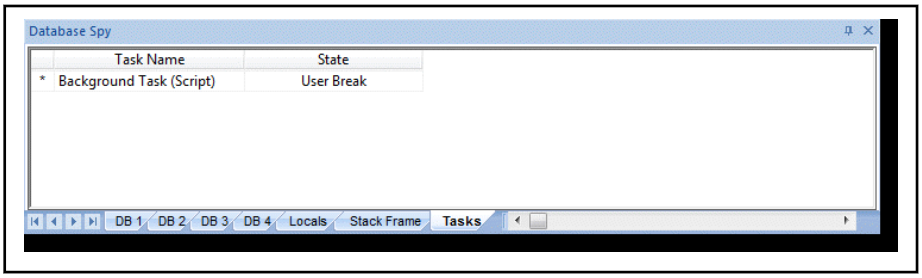


Fig. 20-26: The Tasks tab of the Database spy window



At the moment, only the background task (script) can be debugged. This is why it is the only task which can be displayed on this tab. In the future, additional tasks may follow if the debugging function is activated for more VBScript interfaces.

Please note that these tabs only display information if a project is executed in debug mode and the project is interrupted.

20.5.6 Stepping through your VBScript code

Use the tools Step into, Step over and Step out to go through your VBScript code step by step, line by line.

To be able to use these tools, your project must already be in break mode, either because it reached a previously set breakpoint or by using the **break** tool for manual activation of the break mode.

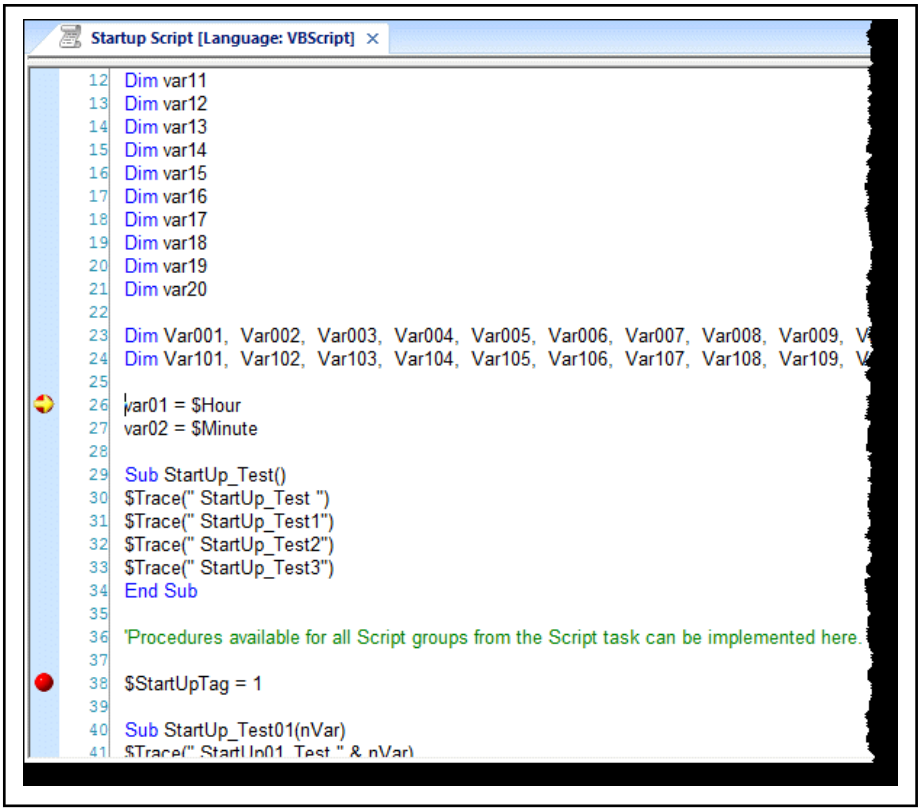


Fig. 20-27: Project in break mode. The yellow arrow shows where the execution was interrupted.

1. To go one step forward in the code, no matter what kind of step it is, click on **Step into** on the **Debug** tab. This moves the debugger one individual step forward. If the next step is a function call, the function is called and the execution is interrupted again at the first step of this function. Meaning of "Step into": The debugger enters the called function. The yellow arrow moves to indicate that the step has been executed.

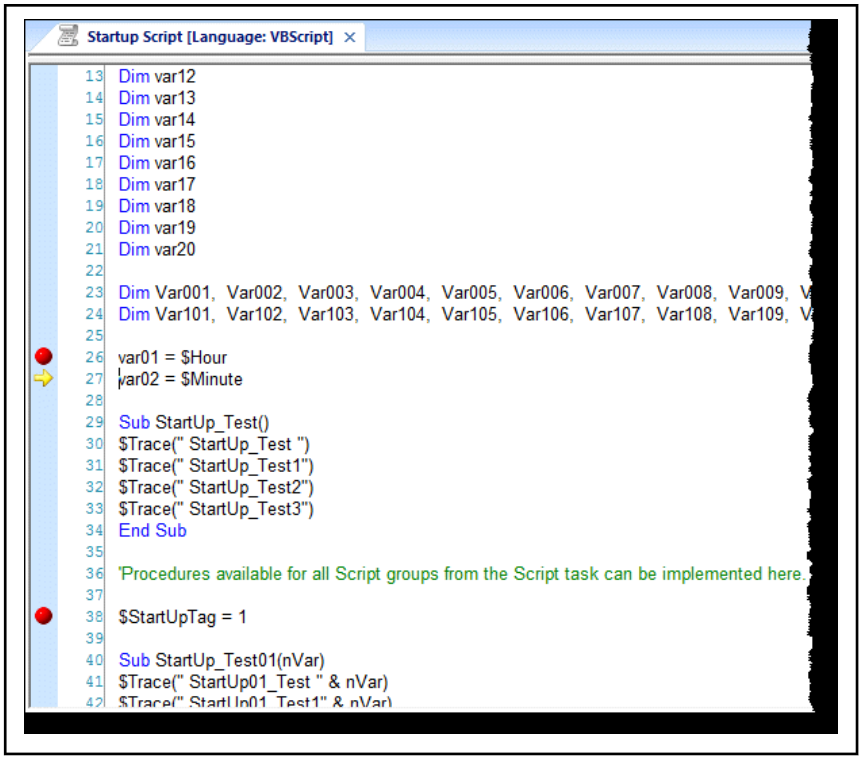


Fig. 20-28: The yellow arrow points one step forward

2. To move one step forward only in the main script, click on **Step over** on the **Debug** tab. If the next step is a function call, the entire function is executed and the debugger continues with the main script. Meaning of "Step over": The debugger leaves out the entire called function. The function is not skipped - it is still executed as written - but no additional time is used to go through it step by step.



Only use this tool if you have already debugged the function and are sure that it is executed correctly.

The yellow arrow moves to indicate that the step has been executed.

3. To finish the execution of a function and continue with the main script click on **Step out** on the **Debug** tab. Meaning of "Step out": The debugger leaves a function it has already entered. The function is not stopped - it is still executed as written - but no additional time is used to go through it step by step.



Only use this tool if you have already debugged the function and are sure that it is executed correctly.

The yellow arrow moves to indicate that the step has been executed.

After each step, the Database spy window is updated and shows the current status of the project (Monitor the current status in the Database spy window on page 972).

21 Appendix: Driver error codes

The **Read status** and **Write status** fields return error codes to the tags specified for these fields. These error codes can be default codes or customized for each driver. For a description of customized error codes, please refer to the documentation for the driver.

The following values are default error codes, which are always negative values:

Value	Description
0	OK
-1	Invalid serial port
-2	Invalid baud rate
-3	Invalid number of bits
-4	Invalid number of stop bits
-5	Invalid parity
-6	Invalid irq
-7	Serial port already in use
-8	Invalid buffer size
-9	Memory not enough
-10	Tx buffer empty
-11	Tx buffer full
-12	Rx buffer empty
-13	Rx buffer full
-14	Timeout waiting CTS
-15	Timeout waiting start a message
-16	Timeout waiting a message to finish
-17	Timeout between rx char
-18	Timeout between tx char
-19	No carrier detected
-20	No DSR detected
-21	Could not find a 8250 in address
-22	Tx line is busy
-23	User abort
-24	Function not supported
-25	Overrun
-26	Parity
-27	Overrun and parity
-28	Framing
-29	Framing and overrun

Value	Description
-30	Framing and parity
-31	Framing, overrun and parity
-32	Timeout waiting a tx message to finish

The `UNICOMM.MSG` file in the `[…]\WinStudio\Bin` directory also contains descriptions of the default error codes.

The `DriverName.MSG` file in the `[…]\WinStudio\Drv` directory contains descriptions of the customized error codes for each driver.

22 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts offer you advice and assistance should you have any queries. You can contact us **24/7**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	**+49 9352 40 5060**
Fax:	**+49 9352 18 4941**
E-mail:	service.svc@boschrexroth.de
Internet:	http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

Index

Symbols

.NET controls ... 237

A

About this documentation ... 1
  Validity of this documentation ... 1
Abs ... 611
ACos ... 661
Active objects
  Button ... 195
  Checkbox ... 201
  Combo box ... 208
  List box ... 212
  Pushbutton ... 197
  Radio button ... 204
  Smart message ... 216
  Text ... 192
  Text input ... 193
ActiveX and .NET control functions ... 890
ActiveX control ... 231
ActiveX events ... 934
Add a link to an external image file ... 242
Add classes using the class wizard ... 425
Adding a touch event to a project screen ... 300
Adding a touch event to a screen object ... 299
Adjust position of screen object ... 274
Adjust the size of a screen object ... 275
Advanced settings for alarm worksheet ... 318
Alarm history
  Format ... 324
Alarm worksheet ... 313
  Advanced settings ... 318
  Email settings ... 315
Alarm worksheet body ... 319
Alarm worksheet header ... 314
Alarm/event control object ... 331
Alarms, events and trends ... 313
Align, center and distribute ... 271
Animations ... 178
  Bargraph ... 250
  Color ... 253
  Command ... 246
  Hyperlink ... 249
  Position ... 255
  Resize ... 257
  Rotation ... 259
  Text data link ... 251
ANSI Z535.6-2006 ... 3
Apache for Windows ... 530
AppActivate ... 692
Appendix
  Built-in scripting language ... 595
  Driver error codes ... 977
  VBScript ... 925
Appendix_A ... 374
Appendix_B ... 377
AppIsRunning ... 693
application
  Stop ... 51
Application
  Configuration and test ... 50
  Start ... 51
  Troubleshooting ... 52
Apply animations to screen objects ... 246
AppPostMessage ... 694
AppSendKeys ... 695
Arithmetic functions ... 610
Arithmetic operators ... 611
Array tags ... 112
Asc2Str ... 631
ASin ... 662
Asynchronous
  versus synchronous ... 724
Auto declaration
  WinStudio tags ... 120
AutoFormat ... 756
Avg ... 622

B

Background color of a screen ... 173
Background picture ... 173
Background task
  Execute ... 17
  Switch ... 17
Background tasks ... 391
Bargraph animation ... 250
Batch creation of history files ... 346
Batch file
  Calling an external file ... 725
Block or unblock users ... 483
BlockUser ... 675
Boolean tags ... 959
Boolean variables ... 959
Button object ... 195

C

CEWebServer ... 527
Change handling of tag values outside the validity range ... 141
Changing properties of multiple screen objects (static objects) ... 190
Changing the properties of multiple screen objects (active objects) ... 221
Changing the properties of multiple screen objects (formatting) ... 265
Changing the target system of an existing project ... 88
CharToValue ... 632
CharToValueW ... 633
Checkbox

Operating modes ... 207
Checkbox object ... 201
CheckESign ... 676
CheckSecurityLevel ... 677
Classes
Create ... 121
Tag ... 115
Classes folder ... 106
ClassMembersToStrVector ... 633
CleanReadQueue ... 696
Client/server architecture ... 519
ClockGetDate ... 648
ClockGetDayOfWeek ... 649
ClockGetTime ... 651
Close ... 665
Closed polygon object ... 183
CloseSplashWindow ... 696
CnfEmail ... 865
CnfFTP ... 895
Color animation ... 253
Color interface ... 277
Combo box object ... 208
Command animation ... 246, 933
Communication tab ... 49, 67
Communication with a remote device ... 413
Communication with other devices ... 413
Communication with remote device
Default driver sheets ... 421
Comparing the Rexroth WinStudio software components ... 21
Conditional statements ... 962
Configuring a DDE client connection to a DDE server ... 455
Configuring a TCP/IP client connection to another project ... 453
Configuring an OPC client connection to an OPC server ... 427
Configuring an OPC UA client connection to an OPC UA server ... 433
Configuring the Remote Agent to automatically run a project ... 551
Constants ... 938
Creating ... 957
Control layer
ISSymbol ... 516
Conventions in this documentation ... 5
Convert screen resolution ... 307
Convert WinStudio screens ... 307
Converting trend history files
From binary to text ... 345
From text to binary ... 346
Cos ... 662
Cot ... 663
Create and configure groups ... 474
Create and configure users ... 478
Create tags
Datasheet ... 119
CreateUser ... 678
Creating a new project ... 85
Creating tags
"On-the-fly" ... 120
Modification ... 120
Cross reference list
WinStudio ... 82

D

Data input ... 282
Data server
Configuration ... 523
Data type
WinStudio ... 111
Database debugging ... 568
Database interface ... 553
Database spy ... 78, 500
Remote ... 502
Database/ERP functions ... 900
Database/ERP worksheet ... 404
Date and time functions ... 647
DateTime2Clock ... 652
DateTime2UTC ... 653
DBCursorClose ... 901
DBCursorColumnCount ... 902
DBCursorColumnInfo ... 903
DBCursorCurrentRow ... 904
DBCursorGetValue ... 905
DBCursorMoveTo ... 905
DBCursorNext ... 906
DBCursorOpen ... 907
DBCursorOpenSQL ... 909
DBCursorPrevious ... 911
DBCursorRowCount ... 912
DBDelete ... 913
DBExecute ... 914
DBInsert ... 916
DBSelect ... 917
DBUpdate ... 919
DbVersion ... 778
Debugging ... 585
DeleteOlderFiles ... 728
Deleting a tag
from the project database ... 143
Development environment
WinStudio ... 39
Dial-up functions ... 871
DialError ... 871
DialGetClientIP ... 877
DialGetServerIP ... 879
DialStatus ... 880
DialUp ... 882
DialUpToCE ... 884
DirCreate ... 728
DirDelete ... 729
DirLength ... 730
DirRename ... 730
DisableMath ... 697

Div ... 612
Documentation
  Change record ... 1
Documentation structure ... 1
Driver error codes ... 977

E

Edit group
  Arrange objects ... 180
  Deactivate dragging into a screen ... 180
  Grid settings ... 181
  Object properties ... 178
  Replace project tags ... 146, 178
  Undo ... 180
Edit screen objects ... 178
Editing the tag database during runtime ... 124
Ellipse object ... 187
email
  configure ... 865
Email functions ... 864
Email settings for alarm worksheet ... 315
EnableMath ... 697
EndTask ... 697
Error codes
  Driver ... 977
Errors ... 947
Event log
  Format ... 329
Event logger ... 326
Event logger functions ... 894
Events and event logs ... 326
Execute ... 51
ExecuteAlarmAck ... 811
ExecuteCustomCall ... 757
Execution modes ... 34
ExitWindows ... 698
Exp ... 625
ExportSecuritySystem ... 680
Ext ... 770

F

False ... 628
File functions ... 727
FileCopy ... 731
FileDelete ... 732
FileLength ... 732
FileReadFields ... 733
FileReadMessage ... 734
FileRename ... 736
FileWrite ... 736
FileWriteFields ... 737
FileWriteMessage ... 739
Fill color ... 276
FindAllDevices ... 885
FindFile ... 740
FindModem ... 886
FindPath ... 741
Folder
  Classes ... 106
  Project tags ... 104
  Shared database ... 107
  System tags ... 108
Fonts for a target language
  Configuration ... 490
Fonts tool ... 280
For … Next ... 846
ForceTagChange ... 812
Format ... 612
Format tab ... 265
Frequently asked questions ... 586
FTP functions ... 895
FTPGet ... 896
FTPPut ... 898
FTPStatus ... 899
Functional descriptions
  How to read ... 608
Functions ... 946

G

GetAlarmCount ... 812
GetAlarmInfo ... 813
GetAppHorizontalResolution ... 778
GetAppPath ... 779
GetBit ... 616
GetClock ... 655
GetComputerIP ... 779
GetComputerName ... 780
GetCursorX ... 780
GetCursorY ... 780
GetDisplayHorizontalResolution ... 781
GetDisplayVerticalResolution ... 781
GetFileAttributes ... 742
GetFileTime ... 742
GetFreeMemoryCE ... 781
GetHardKeyModel ... 782
GetHardKeySN ... 782
GetHstInfo ... 743
GetIPAll ... 783
GetLastESignUser ... 681
GetLicenseGracePeriod ... 784
GetLine ... 744
GetMemoryCE ... 784
GetNetMACID ... 785
GetOS ... 786
GetPerformanceMetric ... 787
GetPrivateProfileString ... 791
GetProductPath ... 791
GetRegValue ... 791
GetRegValueType ... 792
GetScrInfo ... 758
GetSecuritySystemStatus ... 681
GetServerHostName ... 793
GetStatusSendEmailExt ... 867
GetTagValue ... 815

GetTickCount ... 794
GetTimeZone ... 655
GetTimeZoneCount ... 657
GetUserFullName ... 683
GetUserNames ... 683
GetUserPwdAging ... 684
GetUserState ... 685
GetUTC ... 657
Global procedures ... 927
Global replace tool ... 146
Global tab ... 46
Graphic functions ... 756
Graphics
  Active objects ... 191
  Animations ... 246
  Shapes ... 181
Graphics tab ... 170
Grid object ... 381
  Advanced dialog ... 388
  Columns dialog ... 385
  Data dialog ... 383
Group and ungroup objects ... 270

H
HangUp ... 886
Hazard warnings ... 3
Helpdesk ... 979
Hotline ... 979
Hour2Clock ... 657
HST2TXT ... 746
HST2TXTIsRunning ... 748
Hyperlink animation ... 249

I
If ... 628
IIS ... 528
Import from CSV databases ... 157
Import from ODBC databases ... 158
Import from other project databases ... 154
Import Wizard ... 151
Importing from OPC server databases ... 156
Importing Studio XML masks ... 310
Importing Studio XML screens ... 159
ImportSecuritySystem ... 685
ImportXML ... 749
Indirect tags (pointers) ... 117
InfoAppAlrDir ... 794
InfoAppHSTDir ... 794
InfoDiskFree ... 795
InfoResources ... 795
Information tab ... 53
Insert tags using the tag wizard ... 423
Installation ... 21
Installing EmbeddedView or CEView on a target device ... 27
IntelliSense
  VBScript editor ... 952
IsActiveXReg ... 796
IsAppChangedOnServer ... 796
IsScreenOpen ... 699
ISSymbol control layer ... 516
IsTaskRunning ... 699
IsViewerInFocus ... 700
IT security
  Password principles ... 480
  WinStudio project security ... 461
  WinStudio projects ... 85
  WinStudio security system ... 480

K
Keyboard ... 66
  Virtual ... 66
KeyPad ... 700
Keywords ... 946

L
Language reference ... 936
  Constants ... 938
  Errors ... 947
  Functions ... 946
  Keywords ... 946
  Methods ... 945
  Objects and collections ... 941
  Properties ... 942
  Statements ... 943
  System tags ... 949
Language Reference
  Operators ... 937
Layout tool ... 177
Libraries
  .NET controls ... 237
  ActiveX control ... 231
  Symbols ... 222, 223
License
  invalid ... 38
License protection types ... 33
License settings ... 33
Licensing ... 33
Lifetime of variables ... 958
Line color ... 276
Line object ... 181
Linking the database through a remote DB provider ... 555
List box object ... 212
List of available functions ... 595
List of system tags ... 949
Log ... 626
Log message functions ... 609
Log on/off ... 482
Log10 ... 626
Logarithmic functions ... 625
Logical functions ... 627
Logical operator NOT ... 961
Logical operators ... 627

LogOff ... 702
LogOn ... 702
LogWin
Remote ... 503
LogWin output window ... 79
LookupContains ... 751
LookupGet ... 751
LookupLoad ... 752
Loop functions ... 846
Looping with code ... 963

M
Math ... 703
Math worksheet ... 400
Max ... 623
Methods ... 945
Microsoft Access databases
Use ... 575
Microsoft Excel
Use ... 580
Microsoft SQL server
Use ... 573
Min ... 624
Mirror horizontally ... 274
Mirror vertically ... 273
Mobile access ... 527
Mod ... 617
Module activity functions ... 690
MsgBox and InputBox functions ... 956
Multi-touch ... 285
Multi-touch gestures ... 292
Multi-touch gestures in data objects ... 294
Multi-touch gestures in object animations ... 297
Multi-touch gestures in project screens ... 292
Multi-touch settings for project screens ... 286
Project settings ... 286
Screen attributes ... 289
Multimedia functions ... 775
MySQL
Use ... 583

N
NCopy ... 634
Next ... 846
NoInputTime ... 797
NTWebServer ... 527
Num ... 635

O
Object Finder tool ... 149
Object properties
.NET controls ... 237
ActiveX control ... 231
Animation ... 259
Bargraph ... 250
Button ... 195
Checkbox ... 201
Closed polygon ... 183
Colors ... 253
Combo box ... 208
Command ... 246
Ellipse ... 187
Hyperlink ... 249
Line ... 181
List box ... 212
Open polygon ... 182
Position ... 255
Pushbutton ... 197
Radio button ... 204
Rectangle ... 184
Rounded rectangle ... 186
Size ... 257
Smart message ... 216
Text ... 192
Text data link ... 251
Text input ... 193
Object properties dialog ... 178
Objects and collections ... 941
ODBC databases
Use ... 571
ODBC functions ... 847
ODBC worksheet ... 397
ODBCBeginTrans ... 848
ODBCBindCol ... 849
ODBCCanAppend ... 850
ODBCCanUpdate ... 851
ODBCClose ... 851
ODBCCommitTrans ... 852
ODBCDelete ... 852
ODBCExecuteSQL ... 853
ODBCInsert ... 853
ODBCIsBOF ... 854
ODBCIsDeleted ... 854
ODBCIsEOF ... 855
ODBCIsFieldNULL ... 856
ODBCIsFieldNullable ... 856
ODBCMove ... 857
ODBCMoveFirst ... 857
ODBCMoveLast ... 858
ODBCMoveNext ... 858
ODBCMovePrev ... 859
ODBCOpen ... 859
ODBCQuery ... 860
ODBCRollBack ... 861
ODBCSetFieldNULL ... 861
ODBCSetFilter ... 862
ODBCSetSort ... 863
ODBCUnbindCol ... 863
ODBCUpdate ... 864
On touch ... 301
OnTouchCompleted ... 306
OnTouchDelta ... 304
OnTouchStart ... 304
Open ... 666
Open polygon object ... 182

OpenPrevious ... 670
Operators ... 937
  Arithmetic ... 611
  Logical ... 627
Option tab
  Configure database for all task histories ... 56, 567
Options tab ... 54
  Alarm history ... 55, 322
  Database configuration ... 56, 563
  Events ... 328
ORACLE databases
  Use ... 574
Output (LogWin) ... 497

P
Password protection
  Screens, symbols and worksheets ... 484
PDFCreate ... 752
PhoneDialUp ... 887
PhoneDisableListen ... 888
PhoneEnableListen ... 888
PhoneHangUp ... 889
PhoneStatus ... 890
Pi ... 663
Play ... 775
Pointers ... 117
Port Usage ... 542
Position animation ... 255
PostKey ... 703
Pow ... 617
Practise
  View you project in the web ... 543
Preferences tab ... 69
Print ... 753
PrintSetup ... 760
PrintWindow ... 761
Product features ... 8
Product overview ... 7
Product type ... 88
Product versions ... 36
ProductVersion ... 798
Project execution
  Windows services ... 93
Project explorer
  Communication folder ... 49
  Tasks folder ... 47
  Variables folder ... 46
Project Explorer ... 45
  Folders ... 46
  Graphics tab ... 170
Project language
  Setting at program start ... 493
  Setting at runtime ... 494
Project localization ... 487
Project security ... 461
  WinStudio projects ... 461
Project settings
  Additional configuration ... 53
  Communication tab ... 67
  E-Mail ... 88
  FTP configuration ... 90
  Information tab ... 53
  Options tab ... 54
  Preferences tab ... 69
  Viewer tab ... 61
  Web tab ... 71
Project status
  Execution tasks dialog ... 91
Project tags datasheet
  Extend ... 105
Project tags folder ... 104
Properties ... 942
Properties of Boolean tags ... 131
Properties of Integer and Real tags ... 126
Properties of String tags ... 134
Pushbutton object ... 197

R
Radio button
  Operating modes ... 207
Radio button object ... 204
Rand ... 625
RDFileN ... 754
Real values to Integer tags ... 960
Recipe ... 704
Recipe worksheet ... 395
Rectangle object ... 184
Referencing a tag property ... 123
RegSaveCE ... 798
ReloadAppFromServer ... 799
Remote database spy ... 502
Remote LogWin ... 503
Remote management ... 52, 545
Remove unused tags ... 147
RemoveUser ... 687
Replace project tags in a document or screen object ... 146, 178
Report ... 705
Report worksheet ... 396
ResetBit ... 618
ResetDecimalPointsTable ... 762
Resize animation ... 257
RGBColor ... 763
RGBComponent ... 763
Rotating ... 273
Rotation animation ... 259
Round ... 619
Rounded rectangle object ... 186
Run project ... 551
RunGlobalProcedureAsync ... 706
RunGlobalProcedureAsyncGetStatus ... 708
RunGlobalProcedureOnFalse ... 710
RunGlobalProcedureOnTrigger ... 712

RunGlobalProcedureOnTrue ... 714
RunVBScript ... 715

S
Safety instructions ... 3
Save own project symbols ... 223
SaveAlarmFile ... 800
SaveScreenShot ... 764
Scheduler worksheet ... 403
Scope of variables ... 958
Screen attributes dialog ... 170
Screen events ... 955
Screen functions ... 665
Screen groups folder ... 175
Screen objects ... 178
  Active objects ... 191
  Animations ... 246
  Disable translation ... 495
  Shapes ... 181
Screen script ... 932
Screen/worksheet Editor ... 77
Screens and graphics ... 169
Screens, symbols and worksheets
  Password-protected ... 484
Script worksheet ... 401
  Startup ... 402
SecureViewerReload ... 716
Security access levels ... 462
Security for web-based applications
  Implementation ... 537
Security functions ... 675
Security in the remote agent ... 547
Security modes ... 461
  Server settings ... 466
Security rights
  Saving ... 469
Security system
  Configuration wizard ... 463
  Password principles ... 480
Security system configuration
  Data backup ... 481
Security system dialog ... 480
SendEmail ... 868
SendEmailExt ... 869
SendEvent ... 894
SendKeyObject ... 716
Service hotline ... 979
SetAppAlarmPath ... 800
SetAppHSTPath ... 801
SetAppPath ... 719
SetBit ... 619
SetDateFormat ... 801
SetDecimalPoints ... 767
SetDisplayUnit ... 768
SetKeyboardLanguage ... 802
SetLanguage ... 770
SetPassword ... 688
SetRegValue ... 802
SetSystemDate ... 658
SetSystemTime ... 658
SetTagDisplayUnit ... 768
SetTagValue ... 816
SetTimeZone ... 659
SetTranslationFile ... 771
SetViewerInFocus ... 720
SetViewerPos ... 720
SetWebConfig ... 804
Shapes group ... 181
  Add graphics to a screen ... 188
  Closed polygon object ... 183
  Ellipse object ... 187
  Line object ... 181
  Open polygon object ... 182
  Rectangle object ... 184
  Rounded rectangle object ... 186
Shared database folder ... 107
Sharing tags
  IndraLogic devices ... 162
ShowInplaceInput ... 671
ShowMessageBox ... 673
ShutDown ... 721
Signal alert symbol ... 3
Signal words ... 3
Sin ... 664
SMA ... 527
Smart message object ... 216
SNMPGet ... 806
SNMPSet ... 807
SQL relational databases ... 554
SQL server CE
  Use ... 578
SQL server on a Windows Embedded device ... 579
Sqrt ... 620
StartTask ... 721
Statements ... 943
Statistical functions ... 622
Status bar
  Winstudio ... 84
Status codes and messages for OPC UA ... 445
Status codes for external database operations ... 407
Stop ... 51
Str ... 636
Str2Asc ... 636
StrCompare ... 636
StrCompareNoCase ... 637
StrFromInt ... 638
StrFromReal ... 638
StrFromTime ... 639
StrGetElement ... 640
String functions ... 630
StrLeft ... 641
StrLen ... 641
StrLower ... 642

StrRChr ... 642
StrRight ... 642
StrSetElement ... 643
StrStr ... 643
StrStrPos ... 644
StrTrim ... 644
StrTrimAll ... 645
StrUpper ... 646
Studio database gateway ... 556
Support ... 979
Swap16 ... 620
Swap32 ... 621
Sybase
  Use ... 580
Symbols ... 5, 222
SyncAlarm ... 920
SyncAlarmStatus ... 921
SyncEvent ... 921
SyncEventStatus ... 922
Synchronous
  versus asynchronous ... 724
SyncTrendStatus ... 924
System information functions ... 776
System tags ... 949
System tags folder ... 108

T
Tab order of screen objects ... 267
Tag data type
  Choice ... 111
Tag integration ... 161
Tag name syntax ... 109
Tag properties ... 122
  List ... 136
Tag properties tool ... 150
Tag scope
  Choice ... 118
Tag type
  Choice ... 109
Tags and the project database ... 103
Tags database functions ... 809
Tags database reset ... 149
Tags in the project
  Use ... 142
TagsDBAddClass ... 817
TagsDBAddClassMember ... 817
TagsDBAddTag ... 819
TagsDBBeginEdit ... 820
TagsDBEndEdit ... 821
TagsDBGetAlarm ... 822
TagsDBGetClassMember ... 825
TagsDBGetClassMemberCount ... 826
TagsDBGetFirstClass ... 826
TagsDBGetFirstClassMember ... 827
TagsDBGetFirstTag ... 828
TagsDBGetNextClass ... 829
TagsDBGetNextClassMember ... 829
TagsDBGetNextTag ... 830
TagsDBGetTagCount ... 831
TagsDBGetTagProperty ... 832
TagsDBGetTrend ... 833
TagsDBRemoveAlarm ... 834
TagsDBRemoveClass ... 836
TagsDBRemoveClassMember ... 836
TagsDBRemoveTag ... 837
TagsDBRemoveTrend ... 838
TagsDBSetAlarm ... 839
TagsDBSetTagProperty ... 842
TagsDBSetTrend ... 844
Tan ... 664
Target device, execute or stop ... 549
Target platforms and product types ... 86
Tasks tab ... 47
Text data link animation ... 251
Text input object ... 193
Text object ... 192
Thin client
  Configuration ... 532
Thin client program ... 507
Thin clients ... 505
Tips & tricks ... 951
To back, by one level ... 269
To front, by one level ... 269
To the foreground and to the background ... 269
Toggle ... 629
Toolbar
  WinStudio alignment functions ... 41
  WinStudio debug functions ... 44
  WinStudio dynamic functions ... 44
  WinStudio execution functions ... 40
  WinStudio object properties ... 42
  WinStudio table functions ... 43
  WinStudio tag properties ... 42
  WinStudio, customize ... 39
Touch events ... 299
Trace ... 610
Translation functions ... 769
Translation settings
  Advanced ... 495
TranslationLoad ... 772
TranslationLookupClose ... 773
TranslationLookupGet ... 774
TranslationLookupLoad ... 774
Trend control
  Advanced ... 371
  Axes ... 360
  Legend ... 369
  SPC ... 360
  Toolbar ... 365
Trend control object ... 349
Trend control runtime interface ... 350
Trend object
  Data source ... 366
  Object properties ... 353
  Options ... 357

Pen style ... 355
Points ... 354
Use a database as data source ... 377
Use a text file as data source ... 374
Trend worksheet ... 340
Trigonometric functions ... 661
Troubleshooting ... 52
Troubleshooting VBScript
Breakpoints in the VBScript code ... 968
Database spy window ... 972
General Information ... 966
Running the project in Debug mode ... 970
Stepping through a VBScript code ... 973
True ... 630
Trunc ... 621

U
UnblockUser ... 689
Use the LogWin module ... 501
Using user-defined property values when opening image objects ... 261
UTC2DateTime ... 660

V
ValueToChar ... 646
ValueWToChar ... 647
Variables
Declaring ... 958
VBScript ... 925
Calling an external file ... 725
VBScript editor IntelliSense ... 952
VBScript interfaces in the Software ... 925
VBScript operators
Precedence of ... 960
VBScript procedures ... 956
Verify application ... 50
Verify project ... 50
Viewer tab ... 61
ViewerPostMessage ... 722
Virtual keyboard ... 66

W
Wait ... 723
Warnings ... 3
Web application
Page server configuration ... 527
Web server
Apache for Windows ... 530
Installation and configuration ... 527
Web server as host for project pages
Configuration ... 527
Web servers
CEWebServer ... 527
Microsoft IIS ... 528
NTWebServer ... 527
Web tab ... 71
Web tunneling gateway
Installation ... 530
Web Tunneling Gateway ... 527
WebGetFile ... 755
Windows
Processes ... 867
Windows command
Execution ... 724
Windows Embedded support ... 966
WinExec ... 724
WinExecIsRunning ... 726
WinStudio
Activate OPC UA trace log ... 451
Communication folder ... 49
Tasks folder ... 47
Tools ... 76
Variables folder ... 46
WinStudio folder ... 46, 170
WinStudio libraries ... 222
WinStudio projects ... 85
WinStudio software
Installation ... 22
WinStudio toolbar
Alignment functions ... 265
WinStudio toolbars ... 39
Worksheet
Sort and filtering lines ... 143, 320, 343, 410, 458
WritePrivateProfileString ... 808

X
XGet ... 891
XRun ... 892
XSet ... 893

Notes

Notes

rexroth
A Bosch Company

Bosch Rexroth AG
Electric Drives and Controls
P.O. Box 13 57
97803 Lohr, Germany
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr, Germany
Phone +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics

R911341585

DOK-IWORKS-WINSTUD*V14-AP08-EN-P